Eiter, Thomas; Kaminski, Tobias; Redl, Christoph; Weinzierl, Antonius

Exploiting Partial Assignments for Efficient Evaluation of Answer Set Programs with External Source Access

Published in:
Journal of Artificial Intelligence Research

DOI:
10.1613/jair.1.11221

Published: 30/07/2018

Document Version
Publisher's PDF, also known as Version of record

Please cite the original version:
Eiter, T., Kaminski, T., Redl, C., & Weinzierl, A. (2018). Exploiting Partial Assignments for Efficient Evaluation of Answer Set Programs with External Source Access. Journal of Artificial Intelligence Research, 62, 665-727. https://doi.org/10.1613/jair.1.11221

Journal of Artificial Intelligence Research 62 (2018) 665-727 Submitted 1/18; published 7/18

Exploiting Partial Assignments for Efficient Evaluation of Answer Set Programs with External Source Access

Thomas Eiter EITER@KR.TUWIEN.AC.AT
Tobias Kaminski KAMINSKI@KR.TUWIEN.AC.AT
Christoph Redl REDL@KR.TUWIEN.AC.AT
Institute of Logic and Computation
Knowledge-Based Systems Group
Technical University of Vienna (TU Wien)
Favoritenstraße 9-11, A-1040 Vienna, Austria

Antonius Weinzierl ANTONIUS.WEINZIERL@AALTO.FI
Department of Computer Science, Aalto University
FI-00076 Aalto

Abstract

Answer Set Programming (ASP) is a well-known declarative problem solving approach based on nonmonotonic logic programs, which has been successfully applied to a wide range of applications in artificial intelligence and beyond. To address the needs of modern applications, HEX-programs were introduced as an extension of ASP with *external atoms* for accessing information outside programs via an API style bi-directional interface mechanism. To evaluate such programs, conflict-driving learning algorithms for SAT and ASP solving have been extended in order to capture the semantics of external atoms. However, a drawback of the state-of-the-art approach is that external atoms are only evaluated under complete assignments (i.e., input to the external source) while in practice, their values often can be determined already based on partial assignments alone (i.e., from incomplete input to the external source). This prevents early backtracking in case of conflicts, and hinders more efficient evaluation of HEX-programs. We thus extend the notion of external atoms to allow for three-valued evaluation under partial assignments, while the two-valued semantics of the overall HEX-formalism remains unchanged. This paves the way for three enhancements: first, to evaluate external sources at any point during model search, which can trigger learning knowledge about the source behavior and/or early backtracking in the spirit of theory propagation in SAT modulo theories (SMT). Second, to optimize the knowledge learned in terms of so-called nogoods, which roughly speaking are impossible input-output configurations. Shrinking nogoods to their relevant input part leads to more effective search space pruning. And third, to make a necessary minimality check of candidate answer sets more efficient by exploiting early external evaluation calls. As this check usually accounts for a large share of the total runtime, optimization is here particularly important. We further present an experimental evaluation of an implementation of a novel HEX-algorithm that incorporates these enhancements using a benchmark suite. Our results demonstrate a clear efficiency gain over the state-of-the-art HEX-solver for the benchmarks, and provide insights regarding the most effective combinations of solver configurations.

1. Introduction

Answer Set Programming (ASP) is a well-known declarative programming approach based on the stable-model semantics (Gelfond & Lifschitz, 1991). Thanks to efficient and expressive systems

like CLASP (Gebser, Kaufmann, Kaminski, Ostrowski, Schaub, & Schneider, 2011)[1], SMODELS (Simons, Niemelä, & Soininen, 2002)[2], DLV (Leone, Pfeifer, Faber, Eiter, Gottlob, Perri, & Scarcello, 2006)[3], and WASP (Alviano, Dodaro, Leone, & Ricca, 2015a)[4], it has been successfully applied to a wide range of applications in artificial intelligence and beyond (Brewka, Eiter, & Truszczynski, 2011; Erdem, Gelfond, & Leone, 2016). In a nutshell, a problem at hand is represented by a set of rules (an ASP-program) such that its models, called *answer sets*, encode the solutions to the problem; an answer set solver is used to compute models, from which the solutions are then extracted. The approach is a relative of SAT solving, but in contrast, starts from a relational language where variables range over a (finite) set of constants, which allows for more compact formalization than in propositional logic. Furthermore, the support of negation as failure makes ASP inherently nonmonotonic, which allows one for instance to easily express transitive closure. Finally, a number of language extensions that include, among others, optimization constructs, aggregates, disjunctions, and choice rules, cf. Gebser and Schaub (2016), have turned ASP into a very expressive and powerful problem solving tool.

HEX-programs (Eiter, Kaminski, Redl, Schüller, & Weinzierl, 2017) are an extension of ASP-programs aimed at the integration of heterogeneous external information sources, such as XML/RDF data bases, SAT solvers, route planners etc. So-called *external atoms* can be used in rules, and provide a bidirectional interface between the logic program and the external sources in an API style manner. To this end, an external atom passes information from the program, given by predicates and constants, to an external source, which returns the output values for the respective input; the atom states an input-output relationship, which evaluates to either true or false wrt. each potential output value. For example, an external atom $\&synonym[car](X)$ might find synonyms X of car, e.g. $automobile$, bus, $motocar$ etc., by accessing a thesaurus such as the one of Merriam-Webster (Merriam-Webster Website, 2018); that is, e.g. $\&synonym[car](automobile)$ evaluates to true. As seen from this example, external sources can be of non-logical nature, and without particular assumption about how the external source is evaluated. This is facilitated by an abstract modeling of external atoms that can exhibit nonmonotonic behavior, be used in recursive and cyclic definitions, and introduce new constants which do not appear in the original program (known as *value invention*). This rich expressiveness empowers HEX-programs to subsume many other ASP extensions such as programs with (nonmonotonic) aggregates (Alviano, Faber, & Gebser, 2015b), constraint ASP (Ostrowski & Schaub, 2012), and DL-programs (Eiter, Lukasiewicz, Schindlauer, & Tompits, 2004), to mention a few; furthermore, the versatility and genericity of external atoms has been exploited for different purposes and application domains (cf. Erdem, Gelfond, & Leone, 2016; Eiter, Kaminski, Redl, Schüller, & Weinzierl, 2017).

Efficient evaluation of HEX-programs is a challenging issue, and several methods and techniques have been developed to this end. Current evaluation algorithms for (ground) HEX build on existing ASP solvers (in particular, on CLASP). Roughly speaking, they first compute a complete truth-assignment by guessing the truth values of all external atoms and evaluating an accordingly rewritten program using a solver for ordinary ASP-programs. Only when the assignment is complete, the correctness of the guess can be verified by calls to the external sources; if this succeeds, an additional check is required to establish foundedness (i.e., minimality) of the candidate answer

1. http://potassco.sourceforge.net
2. http://www.tcs.hut.fi/Software/smodels
3. http://www.dlvsystem.com
4. https://github.com/alviano/wasp

set. This basic approach has been enhanced with conflict-driven learning techniques, which learn parts of the external source semantics while the search space is traversed in order to prevent that wrong guesses reoccur (Eiter, Fink, Krennwallner, & Redl, 2012). Moreover, advanced techniques for minimality checking have been developed to boost efficiency (Eiter, Fink, Krennwallner, Redl, & Schüller, 2014a).

Despite these improvements, the evaluation of external atoms over complete assignments is an obstacle to good performance in general. This can be mitigated by launching evaluation when the input to an external atom is complete, rather than the whole assignment to all predicates. However, even then this late evaluation does not yield much information to prune the search space and to guide the search algorithm effectively. Intuitively, evaluating external sources under yet partial assignments (i.e., assignments in which only some input atoms are set to true or false, while others remain unassigned) may in some cases allow to decide the eventual truth value of an external atom, regardless of how the assignment will be completed. For example, suppose an external atom $\&planar[node, edge]()$ interfaces an external source for checking whether a graph whose nodes and edges are captured by the unary predicate $node$ and the binary predicate $edge$, respectively, is a planar graph. If a rule $edge(X, Y) \vee not\_edge(X) \leftarrow connected(X, Y)$ guesses the edges of a graph from a pool of connections, the external checker might detect non-planarity even if the guess is not yet complete (i.e., some edges are missing). Early external evaluation has the potential for significant performance gains, as wrong guesses may be detected early on or avoided entirely. In this way, the external sources can guide the answer set search proactively.

This idea is in the spirit of *theory propagation* in *SAT modulo theories (SMT)* (Barrett, Sebastiani, Seshia, & Tinelli, 2009). However, adopting evaluations under partial assignments for HEX-programs is non-trivial, because – unlike in SMT, which considers only fixed theories – external sources are largely black boxes, without much information about their structure (as in case of privacy and data hiding, or of a wrapped web service) let alone a propagation machinery available. Moreover, their heterogeneity and (possibly) nonmonotonic nature, e.g. if the external source access is to ASP engines or argumentation solvers, adds further conceptual and computational complexity.

In this paper, we address the issue of partial evaluation by extending external source access via external atoms from a Boolean semantics, which is defined only under complete assignments, to a three-valued evaluation semantics that is defined under partial assignments. This extension is furnished with novel evaluation techniques to achieve the main goal of efficiency improvements. In particular, learning about the behavior of external sources during evaluation under partial assignments allows us to acquire additional knowledge that aids in guiding the search, similar as theory propagation in SMT; we will encode such knowledge as *nogoods* (Gebser, Kaufmann, & Schaub, 2012), i.e., sets of literals that must not be true at the same time. Moreover, the possibility of such early evaluation further paves the way for identifying the part of the input that is relevant for the final value of an external atom. This allows for minimizing the learned nogoods in order to obtain a larger cover of the external source semantics. Importantly, the semantics of the overall formalism remains unchanged, i.e., the three-valued semantics of external sources is only exploited for performance improvements during the search, while the final answer sets are still two-valued.

1.1 Contributions

The main contributions of our work are briefly summarized as follows.

- We extend the notion of external atoms in two dimensions: first, that it can be evaluated under partial assignments, which set each atom to either true, false, or unassigned. Second, that the output

of the evaluation can be either true, false or unknown; this is because the truth value of the external atom might be definitely known (true or false), or it is yet unknown under the current partial input. However, the overall semantics of HEX-programs remains unchanged, i.e., answer sets still remain two-valued. Thanks to backwards compatibility, we further show how existing (two-valued) external atoms can be seamlessly integrated in our extended framework for three-valued evaluation.

- Based on this extension, we then present a novel evaluation algorithm which exploits three-valued evaluation in external source access for early search space pruning in answer set search. At the heart of this algorithm is learning additional knowledge about external sources, similar as done by theory propagation in SMT solving. Owing to the generic nature of external atoms, this knowledge is acquired by abstractly defined learning functions and represented by nogoods. The latter consist of sets of literals over the input predicates of an external atom plus a designated literal that excludes an evaluation result for matching inputs.

- As well-known, learning can be much more effective if structural properties of the underlying domain are known (cf. Valiant, 1984). We thus present also concrete learning functions for external sources with certain properties, among them monotonicity (relative to the input assignment). Along with that, we devote particular attention to minimizing the nogoods learned, given the interface for evaluation under partial assignments; as already mentioned, small (non-redundant) nogoods are instrumental for pruning the search space of candidate answer sets effectively. However, each minimization step causes computational costs, and external evaluation comes at a price. We mitigate the minimization cost using several techniques: first, besides sequential (literal by literal) minimization, we exploit the divide-and-conquer strategy that was presented by Junker (2004) for conflict set minimization in constraint programming. Second, we develop *simultaneous minimization* of multiple nogoods: an external source may return for a given input multiple output values (e.g., $\&gas\_station[route](X)$ may return for a tour given in $route$ all gas stations close by), and the structural relationships of according nogoods can be exploited. Notably, simultaneous minimization has not been considered in SMT. Third, we consider heuristics towards a good trade-off between the efforts and benefits of evaluation calls and minimization steps.

- In further pushing the computation effort down, we also exploit the possibility for evaluation under partial assignments for minimality checking of candidate answer sets. This is particularly important as the minimality check accounts for a major share of the overall runtime, and usually involves significantly more external evaluation calls than the search for candidates itself. In particular, we discuss how three-valued external evaluation can be interleaved with the search for an unfounded set, which is a semantics-based characterization of minimality for answer sets (Leone, Rullo, & Scarcello, 1997) that has been lifted to HEX-programs (Eiter et al., 2014a). Again, learning from external source calls is used for guiding the search; notably, the nogoods learned can be pooled with those in the main search, and thus speed up the latter.

- We present a prototype implementation of our approach in the DLVHEX system, based on the grounder GRINGO and the solver CLASP as backends. Furthermore, we perform an experimental evaluation of the new techniques on a rich benchmark suite that comprises problems of different characteristics. It appears that each of them can yield significant performance gains, yet the picture of their combination is (as expected) more complex and does not lend for a canonical way of how to apply them; in particular, heuristics may lead to diverging (though explainable) behavior. In any case, our experimental results show a speedup of up to two orders of magnitude in performance (in theory, even exponential gains are possible).

Thus in conclusion, our results significantly advance the evaluation of ASP-programs with external source access, as formally available in HEX-programs, drawing from and extending similar ideas in SMT solving. In turn, the idea of simultaneous minimization might be of interest for SMT and related approaches as well. Our approach is further related to techniques that are used in constraint ASP solvers to minimize learned knowledge, such as those implemented in the CLINGCON system (Ostrowski & Schaub, 2012). However, these techniques usually rely on a tailored integration of theory solvers crafted by experts, while the DLVHEX system branches out for a broad range of heterogeneous source access and users, where the latter may have little or no prior knowledge on solver construction. The interface-based HEX-approach makes it easier for them to harness performance gains by the new learning techniques. In fact, as we show, nogood minimization and theory-specific learning are closely related and thus, user-crafted optimizations in learning, in particular finding small nogoods, can be shifted to nogood minimization by the system for three-valued external evaluation under partial assignments. Full backward compatibility with existing two-valued source descriptions makes exploiting the new features an option but not a requirement for the use of DLVHEX on legacy and new applications.

1.2 Organisation

The rest of this article is structured as follows. After necessary preliminaries in Section 2, we proceed to present in Section 3 the extension of external evaluations under partial assignments, and an algorithm that exploits this in answer set search. The subsequent Section 4 considers nogood learning functions, while Section 5 is devoted to nogood minimization and its relationship to external learning. In Section 6, the usage of evaluation under partial assignments for minimality checking is explored. Section 7 contains a description of our implementation and the experimental evaluation. The final Section 8 considers related work and concludes the article with a discussion of further issues and future work.

A preliminary version of this work was presented at IJCAI 2016 (Eiter, Kaminski, Redl, & Weinzierl, 2016). The additions in this paper comprise the extension of the techniques to the unfounded set check (Section 6), the introduction of an additional nogood minimization algorithm, a more exhaustive discussion of the theory, additional experiments, and formal proofs of the results.

2. Preliminaries

In this section, we start by introducing the necessary background regarding ASP and HEX-programs, which subsequent sections will build on. Our vocabulary consists of a set $\mathcal{P}$ of *predicates*, where each predicate has a fixed arity, a set $\mathcal{C}$ of *constants*, and a set $\mathcal{X}$ of *variables*, whereby the sets are mutually disjoint.

An *atom* is of the form $p(t_1, \ldots, t_\ell)$, abbreviated as $p(\vec{t})$, with predicate $p \in \mathcal{P}$ of arity ℓ and *terms* $t_1, \ldots, t_\ell \in \mathcal{C} \cup \mathcal{X}$. For a vector $\vec{t} = t_1, \ldots, t_\ell$ we write $t \in \vec{t}$ if $t = t_i$ for some $1 \leq i \leq \ell$.

An *answer set program* P is a finite set of (disjunctive) rules r of the form

$$a_1 \vee \cdots \vee a_k \leftarrow b_1, \ldots, b_m, \text{not } b_{m+1}, \ldots, \text{not } b_n., \tag{1}$$

where all a_i, $1 \leq i \leq k$, and b_j, $1 \leq j \leq n$, are atoms. The *head* of a rule r is $H(r) = \{a_1, \ldots, a_k\}$, its *body* is $B(r) = \{b_1, \ldots, b_m, \text{not } b_{m+1}, \ldots, \text{not } b_n\}$, and its *positive* resp. *negative body* is $B^+(r) = \{b_1, \ldots, b_m\}$ resp. $B^-(r) = \{b_{m+1}, \ldots, b_n\}$. A rule r is called a (*disjunctive*) *fact* if

$B(r) = \emptyset$, and a *constraint* if $H(r) = \emptyset$. As usual, an atom (rule, program etc) is *ground*, if no variable occurs in it.

For a program P we let $X(P) = \bigcup_{r \in P} X(r)$ for each $X \in \{H, B, B^+, B^-\}$ to denote the sets of literals that occur in rule heads (H) and rule bodies (B), and the sets of atoms that occur in the positive (B^+) and negative rule bodies (B^-), respectively. The intuitive meaning of the rule (1) is that if all assertions in the rule body hold, then at least one atom in the head has to hold as well.

In the context of ASP, interpretations are usually *Herbrand interpretations*. In this setting, programs with variables can be reduced to programs without variables, by instantiating the variables in rules in all possible ways with constants from $\mathcal{C}$. This process, which is known as *grounding*, is also adopted commonly by solvers in practice, where in addition optimization steps are made in order to avoid useless rules. Suitable syntactic and/or semantic safety conditions guarantee that a finite number of rule instances suffices for answer set computation. After grounding, in a second *solving phase* the answer sets of the program are then computed. As we focus in this paper only on the solving phase, we can assume in the sequel that the vocabulary (and in particular the set of constant symbols) is finite, and that the programs to consider are ground; in the examples, rules with variables stand for all ground instances with respect to this set of constants.

Herbrand interpretations are usually represented by the sets of ground atoms that are true in them. However, when discussing the evaluation of answer set programs, it is often more convenient to explicitly represent which atoms are assigned to *true* resp. *false*. Following Drescher, Gebser, Grote, Kaufmann, König, Ostrowski, and Schaub (2008), a (*signed*) *literal* is either a positive or a negated ground atom $\mathbf{T}a$ or $\mathbf{F}a$, where a is a ground atom. For $\sigma \in \{\mathbf{T}, \mathbf{F}\}$, we let $\overline{\sigma} = \mathbf{T}$ if $\sigma = \mathbf{F}$ and $\overline{\sigma} = \mathbf{F}$ if $\sigma = \mathbf{T}$, and for a literal $L = \sigma a$, we let $\overline{L} = \overline{\sigma} a$. A *complete assignment* over a (finite) set A of atoms is a set $\mathbf{A}$ of literals such that for all $a \in A$, $\mathbf{T}a \in \mathbf{A}$ iff $\mathbf{F}a \notin \mathbf{A}$; here $\mathbf{T}a \in \mathbf{A}$ expresses that a is true and $\mathbf{F}a \in \mathbf{A}$ that a is false.[5]

Let $\mathbf{A}$ be a complete assignment. Then $\mathbf{A}$ satisfies a ground atom a, denoted $\mathbf{A} \models a$, if $\mathbf{T}a \in \mathbf{A}$, and it does not satisfy it, denoted $\mathbf{A} \not\models a$, if $\mathbf{F}a \in \mathbf{A}$. Furthermore, $\mathbf{A}$ satisfies a default-negated atom $\mathrm{not}\, a$, denoted $\mathbf{A} \models \mathrm{not}\, a$, if $\mathbf{A} \not\models a$, and it does not satisfy it, denoted $\mathbf{A} \not\models \mathrm{not}\, a$, if $\mathbf{A} \models a$. A ground rule r is satisfied by $\mathbf{A}$, denoted $\mathbf{A} \models r$, if either $\mathbf{A} \models a$ for some $a \in H(r)$ or $\mathbf{A} \not\models a$ for some $a \in B(r)$. A ground answer set program P is satisfied by $\mathbf{A}$, denoted $\mathbf{A} \models P$, if $\mathbf{A} \models r$ for all $r \in P$.

Answer set programs are interpreted under the *answer set semantics* based on the well-known *GL-reduct* by Gelfond and Lifschitz (1991). Given a ground answer set program P and a complete assignment $\mathbf{A}$, the GL-reduct of P wrt. $\mathbf{A}$ is the program

$$P^{\mathbf{A}} = \{H(r) \leftarrow B^+(r) \mid r \in P, \mathbf{A} \not\models b \text{ for all } b \in B^-(r)\}.$$

For complete assignments $\mathbf{A}_1$ and $\mathbf{A}_2$, let $\mathbf{A}_1 \leq \mathbf{A}_2$ denote that $\{\mathbf{T}a \mid \mathbf{T}a \in \mathbf{A}_1\} \subseteq \{\mathbf{T}a \mid \mathbf{T}a \in \mathbf{A}_2\}$ holds. A complete assignment $\mathbf{A}$ is an *answer set* of an answer set program P if $\mathbf{A}$ is a $\leq$-minimal model of $P^{\mathbf{A}}$. The general idea of ASP is to encode a problem by means of an answer set program and to extract corresponding solutions from the respective answer sets.

Example 1. *Consider the answer set program* $P = \{\, a \leftarrow \mathrm{not}\, b.;\ b \leftarrow \mathrm{not}\, a.;\ \leftarrow a.\,\}$ *and the complete assignment* $\mathbf{A} = \{\mathbf{F}a, \mathbf{F}b\}$. *It is easy to see that* $\mathbf{A}$ *is not a* $\leq$*-minimal model of* $P^{\mathbf{A}} =$

5. Here, *complete* refers to the fact that the complete assignment defines for each atom $a \in A$ whether it is true or false. We explicitly say *complete* in this section in order to distinguish it from the more general concept of partial assignments we introduce later.

$\{ a \leftarrow .;\ b \leftarrow .;\ \leftarrow a. \}$ and thus, not an answer set of P. On the other hand, $\mathbf{A}' = \{\mathbf{F}a, \mathbf{T}b\}$ is a $\leq$-minimal model of $P^{\mathbf{A}'} = \{ b \leftarrow .;\ \leftarrow a. \}$, and it is the only answer set of P as $\{\mathbf{T}a, \mathbf{F}b\}$ is not an answer set due to the constraint "$\leftarrow a.$".

Sets of signed literals are also utilized to formulate constraints wrt. assignments, i.e. to specify combinations of signed literals that are not permitted to be part of a complete assignment. A *nogood* is a set $\{L_1, \ldots, L_n\}$ of literals; and a complete assignment $\mathbf{A}$ is a *solution* to a nogood δ resp. a set of nogoods Δ, if $\delta \not\subseteq \mathbf{A}$ resp. $\delta \not\subseteq \mathbf{A}$ holds for all $\delta \in \Delta$.

Example 2. *The complete assignment $\mathbf{A} = \{\mathbf{T}p(a), \mathbf{T}p(b), \mathbf{T}p(c)\}$ is a solution to the nogood $\{\mathbf{T}p(a), \mathbf{F}p(b)\}$, but not to the nogood $\{\mathbf{T}p(a), \mathbf{T}p(b)\}$.*

Nogoods correspond to clauses as known from SAT solving, and are utilized by ASP-solvers that are based on *Conflict-Driven Nogood Learning* (*CDNL*) (Drescher et al., 2008) for representing the input program and for guiding the search by learning additional nogoods from conflicts.

2.1 HEX-Programs

We briefly recall HEX-programs, which generalize (disjunctive) logic programs under the answer set semantics by integrating external sources of computation; for more details and background, cf. Eiter et al. (2005b, 2014a).

As in the case of answer set programs, we can restrict our theoretical investigation of HEX-programs to ground programs because safety conditions allow for applying an advanced grounding algorithm to compute finite groundings (cf. Eiter, Fink, Krennwallner, & Redl, 2016). However, our examples will also use variables as shortcuts for instantiations with all possible values.

2.1.1 SYNTAX

HEX-programs extend ordinary ASP-programs by *external atoms*, which enable a bidirectional interaction between a program and external sources of computation. In addition to the sets $\mathcal{C}$, $\mathcal{P}$, and $\mathcal{X}$ introduced above, we assume a further finite set $\mathcal{G}$ of external predicate symbols in our vocabulary, which is disjoint from $\mathcal{C}$, $\mathcal{P}$ and $\mathcal{X}$. External predicates in $\mathcal{G}$ are prefixed with '&' to distinguish them from ordinary predicate symbols. Again, due to our restriction of the formal discussion to ground programs it is sufficient to consider only a finite vocabulary.

Informally, external atoms are associated with input and output values, where constants and the extensions of predicates in the input are provided to an external source which computes whether the respective output values are correct. More formally, a *ground external atom* is of the form $\&g[\vec{p}](\vec{c})$, where $\vec{p} = p_1, \ldots, p_k$ is a list of input parameters (predicate names or object constants), called *input list*, and $\vec{c} = c_1, \ldots, c_l$ are constant output terms. More generally, a *non-ground external atom* is of the form $\&g[\vec{Y}](\vec{X})$, where $\vec{Y} = Y_1, \ldots, Y_k$ is a list of input terms (variables, predicate names or object constants), and $\vec{X} = X_1, \ldots, X_l$ are output terms, i.e, variables or object constants.

In contrast to answer set programs, in HEX-programs external atoms can be used in the bodies of rules to specify dependencies on external sources. Formally:

Definition 1. *A HEX-program Π consists of rules r of the form (1), where each a_i, $1 \leq i \leq k$, is an ordinary atom and each b_j, $1 \leq i \leq n$, is either an ordinary atom or an external atom.*

In the following, we call a program *ordinary* if it does not contain external atoms, i.e., if it is a standard ASP-program; as usual, an (ordinary or external) atom, rule, program etc. is *ground*, if

it is variable-free. The head $H(r)$, the body $B(r)$, the positive body $B^+(r)$ and the negative body $B^-(r)$ of a rule r in a HEX-program are defined as before for ordinary programs. We let $B_o^+(r)$ resp. $B_o^-(r)$ be the set of ordinary atoms in $B^+(r)$ resp. $B^-(r)$. Moreover, we denote by $A(\Pi)$ the set of ordinary atoms that occur in a HEX-program Π.

Like ordinary ASP programs, HEX-programs can be reduced by grounding to variable-free programs, where in a non-ground external atom $\&g[\vec{Y}](\vec{X})$ each variable Y_i in $\vec{Y} = Y_1, \ldots, Y_k$ is instantiated with a predicate name or an object constant, and each variable X_i in $\vec{X} = Y_1, \ldots, X_l$ with an object constant. Solvers for HEX-programs do this in a *grounding phase*, which is followed by a solving phase. As we focus here on solving, we assume in the sequel that the HEX-program for evaluation are ground and over a finite alphabet. [6]

According to Definition 1, the usage of external atoms is restricted to the rule bodies in a HEX-program as they can only be queried for information. A common use case of external atoms consists in eliminating answer sets based on external constraints as illustrated by the following example.

Example 3. *Consider the* HEX*-program* Π *that consists of the following facts and rules:*

$$\begin{aligned} node(a).\ \ node(b). & \\ edge(X,Y) \vee n\_edge(X,Y) \leftarrow\ & node(X), node(Y), X \neq Y. \\ \leftarrow\ & \&geq[edge, 2](). \end{aligned}$$

Informally, the rule on the second line guesses edges (arcs) of a self-loop-free directed graph whose vertices are given as facts on the first line. The constraint on the third line uses an external atom $\&geq[edge, 2]()$ *to check whether the number of edges is at most one, by eliminating the guess if at least two edges exist.*

2.1.2 SEMANTICS

Next, we discuss the semantics of ground HEX-programs Π, which generalizes the answer set semantics of Gelfond and Lifschitz (1991). In the following, if not stated otherwise, assignments are over the set $A = A(\Pi)$ of ordinary atoms that occur in the ground HEX-program Π at hand. Furthermore, we let $\mathbb{A}_{\mathcal{P},\mathcal{C}}$ be the set of all possible complete assignments over predicates $\mathcal{P}$ and constants $\mathcal{C}$; in the following we will drop $\mathcal{P}, \mathcal{C}$ from the index and denote this set just as $\mathbb{A}$ since the vocabulary is assumed to be fixed.

The semantics of a ground external atom $\&g[\vec{p}](\vec{c})$ wrt. a complete assignment $\mathbf{A}$ is given by the value of a $1{+}k{+}l$-ary decidable *two-valued (Boolean) oracle function*

$$f_{\&g} \colon \mathbb{A} \times (\mathcal{P} \cup \mathcal{C})^k \times \mathcal{C}^l \to \{\mathbf{T}, \mathbf{F}\}$$

that is defined for all possible complete assignments $\mathbf{A} \in \mathbb{A}$, and tuples $\vec{p}$ and $\vec{c}$, where k and l are the lengths of $\vec{p}$ and $\vec{c}$, respectively. Thus, $\&g[\vec{p}](\vec{c})$ is true relative to $\mathbf{A}$, denoted $\mathbf{A} \models \&g[\vec{p}](\vec{c})$, if $f_{\&g}(\mathbf{A}, \vec{p}, \vec{c}) = \mathbf{T}$ and false, denoted $\mathbf{A} \not\models \&g[\vec{p}](\vec{c})$, otherwise. Importantly, external oracles support *value invention* such that they can be true for output values that do not occur in a respective (non-ground) program. However, all relevant constants are imported by available grounding algorithms (Eiter, Fink, Krennwallner, & Redl, 2016) invoked during the grounding phase of HEX-program evaluation. In practice, oracle functions are realized as solver-plugins, which are usually implemented in *C++* or *Python*.

6. More sophisticated evaluation interleaves grounding and solving, which we omit here for simplicity as it is irrelevant for this work.

While in general, the value of an external atom $f_{\&g}(\mathbf{A}, \vec{p}, \vec{c})$ may depend on any literal in $\mathbf{A}$, we assume in the following that its value depends only on literals over predicates that appear in $\vec{p}$; formally: $f_{\&g}(\mathbf{A}, \vec{p}, \vec{c}) = f_{\&g}(\mathbf{A}', \vec{p}, \vec{c})$ for all complete assignments $\mathbf{A}$ and $\mathbf{A}'$ that assign the same truth values to all atoms of the form $p(\vec{c}')$ where $p \in \vec{p}$.

Satisfaction of ordinary rules and ASP-programs (Gelfond & Lifschitz, 1991) is then extended to HEX-rules and HEX-programs in the obvious way by also taking the satisfaction of external atoms wrt. a complete assignment $\mathbf{A}$ into account. The answer sets of a ground HEX-program Π are then defined as follows. Let the *FLP-reduct* (Faber, Pfeifer, & Leone, 2011) of Π wrt. a complete assignment $\mathbf{A}$ be the set $f\Pi^{\mathbf{A}} = \{r \in \Pi \mid \mathbf{A} \models b, \text{ for all } b \in B(r)\}$ of all rules whose body is satisfied by $\mathbf{A}$. Then:

Definition 2. *A complete assignment $\mathbf{A}$ is an answer set of a ground* HEX*-program Π, if $\mathbf{A}$ is a $\leq$-minimal model of $f\Pi^{\mathbf{A}}$.*

For a given ground HEX-program Π we let $\mathcal{AS}(\Pi)$ denote the set of all answer sets of Π.

For ordinary ASP-programs (i.e., HEX-programs without external atoms), the above definition of answer sets based on the FLP-reduct $f\Pi^{\mathbf{A}}$ is equivalent to the original definition of answer sets by Gelfond and Lifschitz (1991) based on the *GL-reduct*. However, for HEX-programs, the FLP-reduct is more attractive than the GL-reduct as it prevents unintuitive answer sets involving cyclic justifications. Furthermore, the stronger notion of well-justified answer set by Shen, Wang, Eiter, Fink, Redl, Krennwallner, and Deng (2014) excludes any cyclic justification whatsoever, as the whole answer must be obtained in a constructive process that involves classical provability.

We illustrate the notion of answer set in the case of HEX-programs on two simple examples.

Example 4 (Ex. 3 cont'd)**.** *The external atom in Example 3 has an associated oracle function $f_{\&geq}(\mathbf{A}, p, n)$ defined as follows:*

$$f_{\&geq}(\mathbf{A}, p, n) = \begin{cases} \mathbf{T} & \text{if } |\{p(x,y) |\ \mathbf{T}p(x,y) \in \mathbf{A}\}| \geq n, \\ \mathbf{F} & \text{otherwise}. \end{cases}$$

The complete assignment $\mathbf{A}_1 = \{\mathbf{T}node(a), \mathbf{T}node(b), \mathbf{F}edge(a,a), \mathbf{F}edge(a,b), \mathbf{F}edge(b,a), \mathbf{F}edge(b,b)\}$ is an answer set of Π if we add $\mathbf{T}n\_edge(c,c')$ if $\mathbf{F}edge(c,c') \in \mathbf{A}_1$, and we add $\mathbf{F}n\_edge(c,c')$ if $\mathbf{T}edge(c,c') \in \mathbf{A}_1$, where $c, c' \in \{a,b\}$. On the other hand, the assignment $\mathbf{A}_2 = \{\mathbf{T}node(a), \mathbf{T}node(b), \mathbf{F}edge(a,a), \mathbf{T}edge(a,b), \mathbf{T}edge(b,a), \mathbf{F}edge(b,b)\}$ is not an answer set of Π for an analogous addition. As easily seen, Π has three answer sets that correspond to the self-loop-free directed graphs on a,b with less than two edges.

Example 5. *Consider as another example the program $\Pi = \{p \leftarrow \&id[p]().\}$, where $\&id[p]()$ is true iff p is true. Then Π has the answer set $\mathbf{A}_1 = \{\mathbf{F}p\}$; indeed $\mathbf{A}_1$ is a $\leq$-minimal model of the reduct $f\Pi^{\mathbf{A}_1} = \emptyset$. We remark that using the traditional Gelfond-Lifschitz reduct, which had been devised for ordinary ASP-programs by Gelfond and Lifschitz (1991), adapted to* HEX*-programs instead of the FLP-reduct, would admit another answer set $\mathbf{A}_2 = \{\mathbf{T}p\}$; constructing the latter would however involve cyclic justification, which is intuitively not acceptable.*

2.1.3 EVALUATION

The basic evaluation procedure for ground HEX-programs uses a guess-and-check rewriting to ordinary ASP (Eiter, Fink, Ianni, Krennwallner, Redl, & Schüller, 2016) and leverages available solvers

such as CLASP (Gebser et al., 2011) for HEX-evaluation. At this, HEX-programs Π are transformed to ordinary programs by replacing each external atom $\&g[\vec{p}](\vec{c})$ in Π by an ordinary *replacement atom* $e_{\&g[\vec{p}]}(\vec{c})$, and by adding a rule $e_{\&g[\vec{p}]}(\vec{c}) \vee ne_{\&g[\vec{p}]}(\vec{c}) \leftarrow .$ that represents a guess for the truth value of the respective external atom. The answer sets of the resulting *guessing program* $\hat{\Pi}$ are then computed by an ASP solver. The assignment encoded by such an answer set may not satisfy Π, as $f_{\&g}$ may yield for $\&g[\vec{p}](\vec{c})$ a value that is different from the guess for $e_{\&g[\vec{p}]}(\vec{c})$. Thus, the answer set is merely a *model candidate*; if a check against the external sources finds no discrepancy, it is a *compatible set*. Formally:

Definition 3. *A* compatible set *of a program* Π *is an answer set* $\hat{\mathbf{A}}$ *of the guessing program* $\hat{\Pi}$ *such that* $f_{\&g}(\hat{\mathbf{A}}, \vec{p}, \vec{c}) = \mathbf{T}$ *iff* $\mathbf{T}e_{\&g[\vec{p}]}(\vec{c}) \in \hat{\mathbf{A}}$ *for all external atoms* $\&g[\vec{p}](\vec{c})$ *in* Π.

Each answer set of Π is the projection $\mathbf{A}$ of a compatible set $\hat{\mathbf{A}}$ to the atoms $A(\Pi)$ in Π, but not vice versa. To discard the non-answer sets, the evaluation algorithm calls an FLP check which uses unfounded sets to check minimality wrt. $f\Pi^{\mathbf{A}}$ (Eiter et al., 2014a). This check constitutes a second minimality check which intuitively is needed in addition to the usual check that ensures minimality wrt. $\hat{\Pi}^{\hat{\mathbf{A}}}$ to prevent cyclic justifications involving external atoms.

Example 6 (Ex. 3 cont'd)**.** *For the program* Π *in Example 3, the guessing program* $\hat{\Pi}$ *is as follows:*

$$
\begin{aligned}
node(a).\ \ node(b).& \\
edge(X,Y) \vee n\_edge(X,Y) &\leftarrow node(X), node(Y), X \neq Y. \\
&\leftarrow e_{\&geq[edge,2]}().\\
e_{\&geq[edge,2]}() \vee ne_{\&geq[edge,2]}() &\leftarrow .
\end{aligned}
$$

The answer sets of $\hat{\Pi}$ *comprise the sets* $\hat{\mathbf{A}}_1 = \mathbf{A}_1 \cup \{\mathbf{F}e_{\&geq[edge,2]}()\}$ *where* $\mathbf{A}_1 = \{\mathbf{T}node(a), \mathbf{T}node(b), \mathbf{F}edge(a,a), \mathbf{F}edge(a,b), \mathbf{F}edge(b,a), \mathbf{F}edge(b,b)\}$*, and* $\hat{\mathbf{A}}_2 = \mathbf{A}_2 \cup \{\mathbf{F}e_{\&geq[edge,2]}()\}$ *where* $\mathbf{A}_2 = \{\mathbf{T}node(a), \mathbf{T}node(b), \mathbf{F}edge(a,a), \mathbf{T}edge(a,b), \mathbf{T}edge(b,a), \mathbf{F}edge(b,b)\}$*. While* $\hat{\mathbf{A}}_1$ *is a compatible set of* $\hat{\Pi}$*,* $\hat{\mathbf{A}}_2$ *is not. Thus, the latter cannot give rise to some answer set of* Π*. Regarding* $\hat{\mathbf{A}}_1$*, it is easy to see that* $\mathbf{A}_1$ *is a minimal model of the FLP-reduct* $f\Pi^{\mathbf{A}_1} = \{node(a).; node(b).; edge(a,b) \vee n\_edge(a,b) \leftarrow node(a), node(b), a \neq b.; edge(b,a) \vee n\_edge(b,a) \leftarrow node(b), node(a), b \neq a.\}$*. Hence,* $\mathbf{A}_1$ *is an answer set of* Π.

Example 7 (Ex. 5 cont'd)**.** *Reconsider* $\Pi = \{\, p \leftarrow \&id[p]().\,\}$ *from Example 5. Then the guessing program* $\hat{\Pi} = \{\, p \leftarrow e_{\&id[p]}().;\ e_{\&id[p]} \vee ne_{\&id[p]} \leftarrow .\,\}$ *has the answer sets* $\hat{\mathbf{A}}_1 = \{\mathbf{F}p, \mathbf{F}e_{\&id[p]}\}$ *and* $\hat{\mathbf{A}}_2 = \{\mathbf{T}p, \mathbf{T}e_{\&id[p]}\}$*; as easily seen, both are compatible sets of* $\hat{\Pi}$*. Here the projection* $\mathbf{A}_1$ *is a* $\leq$*-minimal model of* $f\Pi^{\mathbf{A}_1} = \emptyset$*, and thus* $\mathbf{A}_1$ *is an answer set of* $\hat{\Pi}$*; on the other hand,* $\mathbf{A}_2$ *not a minimal model of* $f\Pi^{\mathbf{A}_2} = \Pi$*, and thus* $\mathbf{A}_2$ *is not an answer set of* $\hat{\Pi}$.

3. Extension to Partial Assignments

In this section, we start by generalizing complete assignments and oracle functions to partial assignments, which provide a means for explicitly representing that some atom is yet unassigned. To this end, we introduce signed literals $\mathbf{U}a$ to represent that an atom a is yet unassigned in an assignment. Also oracle functions may be extended to deal with unassigned input atoms and in turn, may also evaluate to $\mathbf{U}$ to represent that the value of the corresponding external atom is yet unknown under

the given input. This allows us, in the next step, to enhance the existing evaluation algorithm in such a way that external sources are already evaluated early during search, which potentially allows for earlier backtracking. We note that the extended concepts are only used by the algorithm during solving, while the semantics of the formalism remains unchanged. That is, answer sets define the truth values of all atoms, and are thus still two-valued.

We start with a formal definition of the required concepts:

Definition 4. *A* partial assignment *over a set A of atoms is a set $\mathbf{A}$ of signed literals of the form $\mathbf{T}a$, $\mathbf{F}a$ and $\mathbf{U}a$ such that for every $a \in A$ it holds that $|\mathbf{A} \cap \{\mathbf{T}a, \mathbf{F}a, \mathbf{U}a\}| = 1$.*

Then, a complete assignment as defined in Section 2 corresponds to the special case of a partial assignment which contains no signed literal $\mathbf{U}a$. Since the rest of the paper will use the more general concept of partial assignment only (with complete assignments as special case thereof), we will drop 'partial' in the following and say only *assignment*.

To avoid the introduction of further symbols and heavy notation, we let $\mathbb{A}_{\mathcal{P},\mathcal{C}}$ denote the set of all three-valued assignments over the given vocabulary from now on; as before we drop $\mathcal{P}, \mathcal{C}$ from the index since the vocabulary is fixed. Since we use only three-valued assignments in the remaining part of the paper, this is unambiguous.

For assignments $\mathbf{A}, \mathbf{A}'$ we call $\mathbf{A}'$ an *extension* of $\mathbf{A}$, denoted $\mathbf{A}' \succeq \mathbf{A}$, if it holds that $\mathbf{A} \setminus \{\mathbf{U}a \in \mathbf{A} \mid a \in A\} \subseteq \mathbf{A}'$ (i.e., some unassigned atoms in $\mathbf{A}$ may be flipped to true resp. false). Oracle functions are then extended as follows in order to define the semantics of an external atom $\&g[\vec{p}](\vec{c})$ wrt. partial assignments.

Definition 5. *A* three-valued oracle function *$f_{\&g}$ for a ground external atom $\&g[\vec{p}](\vec{c})$ with k input and l output parameters is a $1{+}k{+}l$-ary function*

$$f_{\&g} : \mathbb{A} \times (\mathcal{P} \cup \mathcal{C})^k \times \mathcal{C}^l \to \{\mathbf{T}, \mathbf{F}, \mathbf{U}\},$$

where $\mathbb{A}$ is the set of all possible assignments $\mathbf{A}$, such that $f_{\&g}(\mathbf{A}, \vec{p}, \vec{c}) \neq \mathbf{U}$ whenever $\mathbf{A}$ is a complete assignment.

Thus, $\&g[\vec{p}](\vec{c})$ is true, false or unassigned relative to $\mathbf{A}$, if the value of $f_{\&g}(\mathbf{A}, \vec{p}, \vec{c})$ is $\mathbf{T}$, $\mathbf{F}$ or $\mathbf{U}$, respectively. As in the case of two-valued oracle functions, we assume that $f_{\&g}(\mathbf{A}, \vec{p}, \vec{c}) = f_{\&g}(\mathbf{A}', \vec{p}, \vec{c})$ for all partial assignments $\mathbf{A}$ and $\mathbf{A}'$ that assign the same truth values to all atoms of the form $p(\vec{c}')$ where $p \in \vec{p}$.

We require that once the output of $f_{\&g}$ is assigned to true or false for some $\mathbf{A}$, the value stays the same for all extensions.

Definition 6. *A three-valued oracle function $f_{\&g}$ is* assignment-monotonic *if $f_{\&g}(\mathbf{A}, \vec{p}, \vec{c}) = X$, $X \in \{\mathbf{T}, \mathbf{F}\}$, implies $f_{\&g}(\mathbf{A}', \vec{p}, \vec{c}) = X$ for all assignments $\mathbf{A}' \succeq \mathbf{A}$.*

Assignment-monotonicity guarantees that no compatible set is lost when querying external sources on partial assignments.

Example 8 (Ex. 3 cont'd)**.** *Reconsider the program Π in Example 3 and recall that the (two-valued) oracle function $f_{\&geq}(\mathbf{A}, edge, 2)$ for a complete assignment $\mathbf{A}$ evaluates to true if $\mathbf{A}$ contains at least two distinct literals $\mathbf{T}edge(x, y)$, and to false otherwise.*

We can extend the oracle to partial assignments $\mathbf{A}'$ *by defining an assignment-monotonic three-valued oracle function* $f'_{\&geq}(\mathbf{A}', p, n)$ *as follows:*

$$f'_{\&geq}(\mathbf{A}', p, n) = \begin{cases} \mathbf{T} & \text{if } |\{p(x,y) \mid \mathbf{T}p(x,y) \in \mathbf{A}'\}| \geq n, \\ \mathbf{U} & \text{if } |\{p(x,y) \mid \mathbf{T}p(x,y) \in \mathbf{A}'\}| < n \text{ and} \\ & \quad |\{p(x,y) \mid \mathbf{T}p(x,y) \in \mathbf{A}' \text{ or } \mathbf{U}p(x,y) \in \mathbf{A}'\}| \geq n, \\ \mathbf{F} & \text{otherwise}. \end{cases}$$

s.t. $\&geq[edge, 2]()$ *can also be evaluated under partial assignments, where* $f'_{\&geq}(\mathbf{A}', edge, 2)$ *yields true if* $|\{edge(x,y) \mid \mathbf{T}edge(x,y) \in \mathbf{A}'\}| \geq 2$*, unassigned if* $|\{edge(x,y) \mid \mathbf{T}edge(x,y) \in \mathbf{A}'\}| < 2$ *and* $|\{edge(x,y) \mid \mathbf{T}edge(x,y) \in \mathbf{A}' \text{ or } \mathbf{U}edge(x,y) \in \mathbf{A}'\}| \geq 2$*, and false otherwise.*

Note that the definition of answer sets carries immediately over to programs with external atoms that use three-valued oracle functions. This is because answer sets are complete assignments and thus, the oracle function call for an external atom $\&g[\vec{p}](\vec{c})$ evaluates to either $\mathbf{T}$ or $\mathbf{F}$. It is therefore not necessary to extend the definitions of satisfaction of ordinary atoms, rules, and programs, and the definition of answer sets, to partial assignments.

A two-valued oracle function, however, cannot handle partial assignments and is thus *not* a special case of a three-valued oracle function that can be passed to an algorithm expecting the latter. However, we can always obtain a three-valued from a two-valued oracle function such that answer sets remain invariant.

Proposition 1. *For every* HEX*-program* Π *and external predicate* $\&g$ *defined by a two-valued oracle function, one can redefine* $\&g$ *by an assignment-monotonic three-valued oracle function without changing the answer sets of* Π*.*

Intuitively, we construct a three-valued oracle function which coincides with the two-valued one for complete assignments, and returns $\mathbf{U}$ otherwise. Hence, Proposition 1 allows us to "wrap" two-valued oracle functions for use by our algorithms below; in the implementation this is the basis for backwards compatibility with existing external sources.

We exploit partial assignments by extending previous evaluation algorithms. In the spirit of *theory propagation* in SMT solvers (Barrett et al., 2009), we use *external theory learning (ETL)*. It is related to *external behavior learning (EBL)*, which encodes observed output of external sources as nogoods (Eiter et al., 2012), but our extension works over partial assignments such that external sources may drive early propagation of truth values implied by the current partial assignment.

As for EBL, we can associate with each external source a *learning-function* Λ that yields a set of nogoods $\Lambda(\&g[\vec{p}], \mathbf{A})$ learned from the evaluation of $\&g[\vec{p}]$ under an assignment $\mathbf{A}$. Learned nogoods have to be correct, i.e., they must not eliminate compatible sets. Formally, a nogood δ is *correct wrt. a program* Π, if all compatible sets of Π are solutions to δ.

We extend learning functions for partial assignments as follows. Let $\mathcal{E}(\Pi)$ contain all expressions $\&g[\vec{p}]$ (called *external predicate instances*) that occur in Π, and let $\mathcal{L}(\hat{\Pi}) = \{\mathbf{T}a, \mathbf{F}a, \mathbf{U}a \mid a \in A(\hat{\Pi})\}$ denote the set of all signed literals on atoms that occur in $\hat{\Pi}$.

Definition 7. *A* (three-valued) learning function *for a* HEX*-program* Π *is a mapping* $\Lambda\colon \mathcal{E}(\Pi) \times \mathbb{A} \to 2^{2^{\mathcal{L}(\hat{\Pi})}}$ *that assigns each external predicate instance* $\&g[\vec{p}]$ *and partial assignment* $\mathbf{A}$ *a set* $\Lambda(\&g[\vec{p}], \mathbf{A})$ *of nogoods. We call* Λ correct *for* Π*, if for all arguments* $\&g[\vec{p}] \in \mathcal{E}$ *and* $\mathbf{A} \in \mathbb{A}$*, every nogood* $\delta \in \Lambda(\&g[\vec{p}], \mathbf{A})$ *is correct for* Π*.*

Algorithm 1: HEX-CDNL

Input: A HEX-program Π
Output: An answer set of Π if one exists, and $\bot$ otherwise

Let $\hat{\Pi}$ be the guessing program of Π
$\hat{\mathbf{A}} \leftarrow \{\mathbf{U}a \mid a \in A(\Pi)\}$ `// all atoms unassigned`
$\nabla \leftarrow \emptyset$ `// set of dynamic nogoods`
$dl \leftarrow 0$ `// decision level`
while *true* **do**
  $(\hat{\mathbf{A}}, \nabla) \leftarrow$ Propagation$(\hat{\Pi}, \nabla, \hat{\mathbf{A}})$ **(a)**
  if *some nogood* δ *violated by* $\hat{\mathbf{A}}$ **then** **(b)**
    if $dl = 0$ **then return** $\bot$
    analyze conflict, add learned nogood to ∇, set dl to backjump level
  else if $\hat{\mathbf{A}}$ *is complete* **then** **(c)**
    $\mathbf{A} \leftarrow \hat{\mathbf{A}} \cap \{\mathbf{T}a, \mathbf{F}a \mid a \in A(\hat{\Pi})\}$
    if *there is an unfounded set* U *of* $\hat{\Pi}$ *wrt.* $\hat{\mathbf{A}}$ *s.t.* $U \cap \{\mathbf{T}a \mid \mathbf{T}a \in \hat{\mathbf{A}}\} \neq \emptyset$ **then** **(d)**
      construct violated nogood for U and add it to ∇
      analyze conflict, add learned nogood to ∇, set dl to backjump level
    else if $\hat{\mathbf{A}}$ *is not compatible for* $\hat{\Pi}$ *or* $\mathbf{A}$ *is not a minimal model of* $f\Pi^{\mathbf{A}}$ **then**
      $\nabla \leftarrow \nabla \cup \{\hat{\mathbf{A}}\}$
    else
      return $\mathbf{A}$
  else if *heuristics evaluates* $\&g[\vec{y}]$ *and* $\Lambda(\&g[\vec{y}], \hat{\mathbf{A}}) \not\subseteq \nabla$ **then** **(e)**
    $\nabla \leftarrow \nabla \cup \Lambda(\&g[\vec{y}], \hat{\mathbf{A}})$
  else **(f)**
    Guess $\sigma a \in \{\mathbf{T}a, \mathbf{F}a\}$ for some atom a with $\mathbf{U}a \in \hat{\mathbf{A}}$
    $dl \leftarrow dl + 1$
    $\hat{\mathbf{A}} \leftarrow (\hat{\mathbf{A}} \setminus \{\mathbf{U}a\}) \cup \{\sigma a\}$

Throughout the rest, we assume that learning functions are always correct for the programs at hand.

We now present a procedure for computing an answer set of a HEX-program, shown in Algorithm 1 and illustrated by Figure 1. To compute multiple answer sets, we can naively add previous answer sets as constraints and call the algorithm again (cf. Gebser, Kaufmann, Neumann, & Schaub, 2007) for more elaborated techniques. The basic structure of Algorithm 1 resembles an ordinary ASP solver, but has additional checks in Part (c) and external calls to learn further nogoods in Part (e), which is based on partial assignments. Without the extensions, it computes an answer set $\hat{\mathbf{A}}$ of the guessing program $\hat{\Pi}$ and returns the projection of $\hat{\mathbf{A}}$ to the atoms in Π (cf. Drescher et al., 2008). To this end, it starts from a void assignment and performs unit propagation in Part (a) to derive further truth values. Part (b) backtracks and learns nogoods from conflicts. Part (c) checks compatibility and minimality of the model candidate. To this end, the (more efficient) check in the **if**-block checks minimality from the perspective of an ordinary ASP-solver without respecting the semantics of external sources (i.e., minimality of $\hat{\mathbf{A}}$ wrt. $\hat{\Pi}$); the minimality check is realized using so-called *unfounded sets* (Eiter et al., 2014a), which are atoms that support each other only cyclically. We will introduce them formally and discuss them in detail in Section 6. If this check fails,

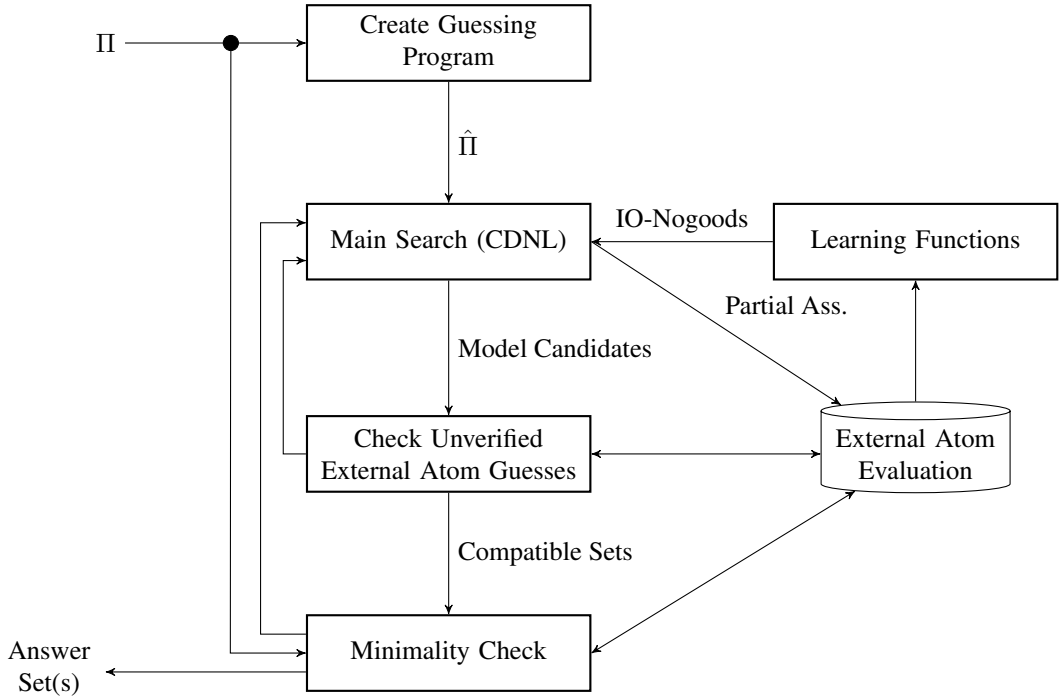

Figure 1: Illustration of the workflow of Algorithm 1.

the algorithm learns a nogood and backtracks. Only if this check is passed, the **elsif**-block in Part (d) checks compatibility and minimality under consideration of external sources (i.e., minimality of $\mathbf{A}$ wrt. Π), cf. Definition 2; if this check is also passed, an answer set has been found. Finally, without Part (e), the algorithm makes a guess in Part (f) if no further truth values can be derived and the assignment is incomplete.

The additional calls to external sources and nogood learning in Part (e) are not mandatory but prune the search space; they may eliminate assignments violating known behavior of external sources already early in the search, while the correctness of the learning function Λ guarantees that no compatible set of $\hat{\Pi}$ (and hence no answer set of Π) is eliminated. Notably and in contrast to previous algorithms (Eiter et al., 2012), external atoms are evaluated under partial assignments and use a three-valued oracle function.

We can show that this algorithm is sound and complete:

Theorem 1 (Soundness and Completeness of Algorithm 1)**.** *If Algorithm 1 returns for an input program* Π *(i) an assignment* $\mathbf{A}$*, then* $\mathbf{A}$ *is an answer set of* Π*; (ii) the symbol* $\perp$*, then* Π *is inconsistent.*

Algorithm HEX-CDNL describes the schematic backbone of concrete incarnations that are obtained by choosing particular learning functions and heuristics for driving the learning process under partial assignment evaluation. Furthermore, different procedures for the unfounded set check might be used; we shall deal with these aspects in the next sections.

4. Nogood Learning with Partial Assignments

In this section, we discuss the generation of nogoods which partially encode the semantics of external atoms. In contrast to previous work on external behavior learning (EBL), this generation however will work for partial assignments in general, and not only for complete assignments. When certain ground instances of an external atom can already be decided, nogoods can be learned early on, and incompatible assignments identified; thus, they can guide the solver. Intuitively, nogoods learned based on partial assignments are preferable as they are usually smaller and cut incomplete assignments.

4.1 Three-valued Learning Functions

Let us first assume that we have no further knowledge about external sources and can only observe their (partial) output under a given (possibly partial) input. We introduce a three-valued learning function for the general case, which is a lifting of the respective two-valued learning function defined by Eiter et al. (2012).

Definition 8. *An* input-output *(*io-*)*nogood *is any nogood of the form*

$$N = \{\sigma_1 a_1, \ldots, \sigma_n a_n\} \cup \{\sigma_{n+1} e_{\&g[\vec{p}]}(\vec{c})\} \text{ where } \sigma_1, \ldots, \sigma_{n+1} \in \{\mathbf{T}, \mathbf{F}\};$$

we let $N_I = \{\sigma_1 a_1, \ldots, \sigma_n a_n\}$ *be the literals over ordinary atoms (called the* input part*),* $N_O = \{\sigma_{n+1} e_{\&g[\vec{p}]}(\vec{c})\}$ *be the replacement atom (called the* output part*) of* N*, and* $\sigma(N_O) = \sigma_{n+1}$*. We call* N faithful*, if* $f_{\&e}(\mathbf{A}, \vec{p}, \vec{c}) = \overline{\sigma(N_O)}$ *for all partial assignments* $\mathbf{A} \supseteq N_I$*, i.e., it resembles the semantics of the external source.*

We note the following property.

Proposition 2. *If* N *is a faithful io-nogood such that* $N_O = \{\sigma_{n+1} e_{\&g[\vec{p}]}(\vec{c})\}$*, then* N *is correct wrt. all programs* Π *that use* $e_{\&g[\vec{p}]}(\vec{c})$*.*

As for the converse, correct nogoods wrt. a given program Π may be io-nogoods that are not faithful, or simply even no io-nogoods. In particular, for inconsistent ordinary ASP-programs any io-nogood is trivially correct as there are no compatible sets which could be wrongly eliminated, but e.g. the empty nogood is not an io-nogood.

When the oracle of an external atom is evaluated, the solver can create a new nogood for the observed input-output relationship. That is, evaluating $\&g[\vec{p}]$ for a partial assignment $\mathbf{A}$, the solver learns, given all true and false literals of input predicates, whether the output contains $\vec{c}$, where $f_{\&g}(\mathbf{A}, \vec{p}, \vec{c}) \neq \mathbf{U}$. Note that, since we only consider ground HEX-programs Π, for any partial assignment $\mathbf{A}$ and input list $\vec{p}$ there can only be finitely many tuples $\vec{c}$ where $f_{\&g}(\mathbf{A}, \vec{p}, \vec{c}) = \mathbf{F}$ such that $\vec{c}$ occurs in a given program Π. Hence, in general we only need to consider a fixed number of potential output tuples, which we call a *scope* $\mathcal{S}$ of output tuples. In general, the scope may contain all output tuples over the given finite vocabulary.

Definition 9. *The learning function for an external predicate with input parameters* $\&g[\vec{p}]$ *under partial assignment* $\mathbf{A}$ *and scope* $\mathcal{S}$ *is*

$$\Lambda_u(\&g[\vec{p}], \mathbf{A}) = \left\{ \mathbf{A}' \cup \{\overline{\sigma e_{\&g[\vec{p}]}(\vec{c})}\} \mid f_{\&g}(\mathbf{A}, \vec{p}, \vec{c}) = \sigma \neq \mathbf{U}, \vec{c} \in \mathcal{S} \right\},$$

where $\mathbf{A}' = \{\sigma' p(\vec{c}') \in \mathbf{A} \mid p \in \vec{p}, \sigma' \neq \mathbf{U}\}$ *is the relevant part of the external atom input.*

Each respective nogood is an io-nogood by construction and as we have that the oracle is assignment-monotonic, also faithful. Hence:

Proposition 3. *Let $\&g[\vec{p}](\cdot)$ be an external atom in a* HEX*-program* Π*. Then for all assignments* $\mathbf{A}$*, the nogoods* $\Lambda_u(\&g[\vec{p}], \mathbf{A})$ *in Definition 9 are correct wrt.* Π*.*

Example 9 (Ex. 3 cont'd)**.** *Assume we are given for the graph guessing program* Π *in Example 3 the partial assignment* $\mathbf{A} = \{\mathbf{T}node(a), \mathbf{T}node(b), \mathbf{F}edge(a,b), \mathbf{U}edge(b,a), \mathbf{T}n\_edge(a,b), \mathbf{U}n\_edge(b,a)\}$*. Then the learning function* $\Lambda_u(\&geq[edge,2], \mathbf{A})$ *yields the single io-nogood* $\{\mathbf{F}edge(a,b), \mathbf{T}e_{\&geq[edge,2]}()\}$*, which is indeed faithful. On the other side, for* $\mathbf{A}' = \{\mathbf{T}node(a), \mathbf{T}node(b), \mathbf{T}edge(a,b), \mathbf{U}edge(b,a), \mathbf{F}n\_edge(a,b), \mathbf{U}n\_edge(b,a)\}$*, we find that no io-nogood can be learned and* $\Lambda_u(\&geq[edge,2], \mathbf{A}')$ *returns* $\emptyset$ *as* $f'_{\&geq}(\mathbf{A}', edge, 2) = \mathbf{U}$*, where* $f'_{\&geq}$ *is the three-valued assignment-monotonic oracle functions as in Example 8.*

According to Eiter et al. (2012), given an external predicate with input parameters $\&g[\vec{p}]$ and a complete assignment $\mathbf{A}$, an input parameter $p_i \in \vec{p}$ is *monotonic*, if $f_{\&g}(\mathbf{A}, \vec{p}, \vec{c}) = \mathbf{T}$ implies that $f_{\&g}(\mathbf{A}', \vec{p}, \vec{c}) = \mathbf{T}$ for every $\mathbf{A}' \geq \mathbf{A}$ that augments $\mathbf{A}$ only by atoms with predicate p_i. We can refine a three-valued function Λ_u similar to two-valued learning functions (cf. Eiter et al., 2012), and tailor it to external sources with specific properties. We show this for external atoms which are monotonic in an input predicate p_i, i.e., the value of the external atom cannot switch from true to false if more atoms over p_i become true and, conversely, it cannot switch from false to true if more atoms over p_i become false. Then, literals of the form $\mathbf{F}p_i(\vec{c}')$ may be dropped from io-nogoods containing $\mathbf{F}e_{\&g[\vec{p}]}(\vec{c})$, and literals of the form $\mathbf{T}p_i(\vec{c}')$ may be dropped from io-nogoods containing $\mathbf{T}e_{\&g[\vec{p}]}(\vec{c})$. Accordingly, by exploiting monotonic behavior of oracle functions we are able to obtain smaller, i.e. more general, io-nogoods than by using the general learning function Λ_u.

Definition 10. *The learning function for an external predicate with input parameters $\&g[\vec{p}]$ that is monotonic in $\vec{p_m} \subseteq \vec{p}$, under a partial assignment $\mathbf{A}$ and a scope $\mathcal{S}$, yields*

$$\Lambda_{mu}(\&g[\vec{p}], \mathbf{A}) = \left\{ \mathbf{A}'_\sigma \cup \{\overline{\sigma e_{\&g[\vec{p}]}(\vec{c})} \mid f_{\&g}(\mathbf{A}, \vec{p}, \vec{c}) = \sigma \neq \mathbf{U}, \vec{c} \in \mathcal{S} \right\},$$

where $\mathbf{A}'_\sigma = \{\sigma' p(\vec{c}') \in \mathbf{A} \mid p \in \vec{p}, p \notin \vec{p_m}, \sigma' \neq \mathbf{U}\} \cup \{\sigma p(\vec{c}') \in \mathbf{A} \mid p \in \vec{p_m}\}$.

As before, we can also show that nogoods learned by means of the learning function Λ_{mu} are not violated by any compatible set:

Proposition 4. *Let $\&g[\vec{p}](\cdot)$ be an external atom in a* HEX*-program* Π*. Then for all assignments* $\mathbf{A}$*, the nogoods* $\Lambda_{mu}(\&g[\vec{p}], \mathbf{A})$ *in Definition 10 are correct wrt.* Π*.*

Example 10 (Ex. 3 cont'd)**.** *Consider again program* Π *from Example 3 and the partial assignment* $\mathbf{A} = \{\mathbf{T}node(a), \mathbf{T}node(b), \mathbf{F}edge(a,b), \mathbf{T}edge(b,a), \mathbf{T}n\_edge(a,b), \mathbf{U}n\_edge(b,a)\}$*. When employing the learning function* Λ_u*, we obtain* $\Lambda_u(\&geq[edge,2], \mathbf{A}) = \left\{\{\mathbf{F}edge(a,b), \mathbf{T}edge(b,a), \mathbf{T}e_{\&geq[edge,2]}()\}\right\}$*. However, we obtain* $\Lambda_{mu}(\&geq[edge,2], \mathbf{A}) = \left\{\{\mathbf{F}edge(a,b), \mathbf{T}e_{\&geq[edge,2]}()\}\right\}$ *by exploiting monotonicity of the input parameter* $edge$*.*

The learning functions Λ_u and Λ_{mu} generate nogoods depending on the oracle function given a certain input. However, an external source provider usually knows the source semantics better and can thus provide better nogoods. The latter might include only the necessary atoms in the input; they are thus smaller and prune more of the search space. In such cases, it makes sense to provide custom learning functions $\Lambda_l(\&g[\vec{p}], \mathbf{A})$ which generate for $\&g[\vec{p}]$ and a (possibly partial) assignment $\mathbf{A}$ a set of nogoods.

5. Nogood Minimization

In this section, we discuss a second way to exploit partial assignments for more effective learning of io-nogoods based on three-valued oracle functions. Instead of calling three-valued oracle functions with partial input assignments that are generated during solving, a new partial assignment $\mathbf{A}'$ can be obtained from a given assignment $\mathbf{A}$, where $\mathbf{A} \succeq \mathbf{A}'$, by changing part of the truth values of literals in $\mathbf{A}$ from $\mathbf{T}$ or $\mathbf{F}$ to $\mathbf{U}$ afterwards. Then, a three-valued oracle function can be called with the resulting assignment $\mathbf{A}'$ in order to detect truth assignments in $\mathbf{A}$ which are irrelevant for the evaluation of the respective external source.

By employing this strategy, we eliminate redundant (input) literals from the nogoods in Λ_u and Λ_{mu}, while faithfulness of io-nogoods is retained (relying on assignment-monotonicity of three-valued oracle functions). Recall that io-nogoods do not contain any literals which are unassigned. Hence, we can obtain smaller and thus, more general io-nogoods, which potentially prune larger parts of the search space. For this purpose, we introduce two new algorithms for computing minimal io-nogoods, i.e, nogoods from which no literal in the input part can be removed without changing the output value of the respective oracle function to unassigned. Moreover, we show that minimization and theory-specific learning are in fact closely related.

Definition 11. *Given a faithful io-nogood N with $N_O = \{\sigma e_{\&g[\vec{p}]}(\vec{c})\}$, the set of minimized nogoods of N is*

$$minimize(N) = \{N' \subseteq N \mid N' \text{ is a faithful io-nogood}, f_{\&g}(N'', \vec{p}, \vec{c}) = \mathbf{U} \text{ for all } N'' \subsetneq N'_I\}.$$

This extends to sets S of nogoods by $minimize(S) = \bigcup_{N \in S} minimize(N)$. Note that exponentially many minimal nogoods in the size of N are possible.

Example 11 (Ex. 10 cont'd)**.** *Consider the assignment $\mathbf{A} = \{\mathbf{T}node(a), \mathbf{T}node(b), \mathbf{F}edge(a,b), \mathbf{F}edge(b,a), \mathbf{T}n\_edge(a,b), \mathbf{T}n\_edge(b,a)\}$ together with the learned faithful io-nogood $N = \{\mathbf{F}edge(a,b), \mathbf{F}edge(b,a), \mathbf{T}e_{\&geq[edge,2]}()\} \in \Lambda_u(\&geq[edge,2], \mathbf{A})$. According to Definition 11, we obtain $minimize(N) = \{\{\mathbf{F}edge(a,b), \mathbf{T}e_{\&geq[edge,2]}()\}, \{\mathbf{F}edge(b,a), \mathbf{T}e_{\&geq[edge,2]}()\}\}$.*

The minimized nogoods subsume all faithful io-nogoods.

Proposition 5. *Let $\mathbf{A}$ be a partial assignment and N be a faithful io-nogood for $\&g[\vec{p}]$ over the atoms in $\mathbf{A}$. Then some $N' \in minimize(\Lambda_u(\&g[\vec{p}], \mathbf{A}))$ exists such that $N' \subseteq N$.*

As a subset of each faithful io-nogood occurs among all minimized nogoods, no further faithful io-nogoods prune the search space more effectively. Still, there might be further correct nogoods (non-io ones and/or depending on the program).

In the following, we call a theory-specific learning function $\Lambda_l(\cdot,\cdot)$ *io-complete* for an external source $\&g$, if for every partial assignment $\mathbf{A}' \subseteq \mathbf{A}$ and input list $\vec{p}$, it holds that $\Lambda_l(\&g[\vec{p}], \mathbf{A})$ is the least set that contains $\mathbf{A}' \cup \{\overline{\sigma e_{\&g[\vec{p}]}(\vec{c})}\}$ for every output list $\vec{c}$ such that $f_{\&g}(\mathbf{A}', \vec{p}, \vec{c}) = \sigma \in \{\mathbf{T}, \mathbf{F}\}$; otherwise, we call $\Lambda_l(\cdot,\cdot)$ *partial*. That is, Λ_l learns *all and only* io-nogoods with a premise over the current partial assignment which resemble the semantics of $\&g$.

As it turns out, learning using io-complete theory-specific learning functions and nogood minimization are closely related. Let $min_{\subseteq}(S) = \{N \in S \mid \nexists N' \in S \text{ s.t. } N' \subsetneq N\}$ be the restriction of S to subset-minimal nogoods.[7] Then:

7. Despite similar names, $minimize$ differs from $min_{\subseteq}$ as it minimizes wrt. oracle results while $min_{\subseteq}$ just selects the minimal sets.

Algorithm 2: Simultaneous Nogood Minimization

Input: A set S of faithful io-nogoods N with $N_I = \{\sigma_1 a_1, \ldots, \sigma_n a_n\}$
Output: A set of minimal faithful io-nogoods

$ch \leftarrow \emptyset$ `// cache for oracle calls`
for *each signed literal* $\sigma_i a_i \in N_I$ **do**
  for *each io-nogood* $N' \in S$ *with* $N'_O = \{\sigma_{n+1} e_{\&g[\vec{p}]}(\vec{c})\}$ **do** (a)
    $N^s \leftarrow N'_I \setminus \{\sigma_i a_i\}$ `// smaller oracle input`
    if $\langle N^s, \cdot \rangle \notin ch$ **then** (b)
      $ch \leftarrow ch \cup \{\langle N^s, \{\sigma e_{\&g[\vec{p}]}(\vec{c'}) \mid f_{\&g}(N^s, \vec{p}, \vec{c'}) = \sigma \neq \mathbf{U}\}\rangle\}$
    if $\overline{\sigma_{n+1}} e_{\&g[\vec{p}]}(\vec{c}) \in$ *output for* $\langle N^s, output\rangle \in ch$ **then** (c)
      Replace N' by $N^s \cup \{\sigma_{n+1} e_{\&g[\vec{p}]}(\vec{c})\}$ in S
return S

Proposition 6. *Let Λ_l be an io-complete theory-specific learning function for an external source &g. Then, for all partial assignments $\mathbf{A}$ and input lists $\vec{p}$ we have $minimize(\Lambda_u(\&g[\vec{p}], \mathbf{A})) = min_{\subseteq}(\Lambda_l(\&g[\vec{p}], \mathbf{A}))$.*

This proposition implies that we have alternative techniques to learn all nogoods that prune the search space in an optimal (cf. Proposition 5) way. As above, it considers only *faithful io-nogoods* while further correct nogoods may exist. Notably, the equality holds only under the premises of *exhaustive* minimization in the first case and an *io-complete* theory-specific learning function in the second; otherwise, different sets of nogoods may be produced. As both operations are expensive and impractical, it makes sense to support both (incomplete) minimization and (incomplete) theory-specific learning functions.

5.1 Sequential Nogood Minimization

In practice, we use Algorithm 2 to compute only one minimal io-nogood for each learned io-nogood. Instead of minimizing each nogood separately and to avoid redundant queries, we proceed in parallel and use a cache for the external atom output of a set S of io-nogoods with identical input but different outputs. The algorithm works by sequentially removing the same literal simultaneously from the premises of all N in S in Part (a), and checking whether the output for the resulting premises is already in the cache, in Part (b). If not, all outputs $\vec{c'}$ s.t. $f_{\&g}(\mathbf{A}, \vec{p}, \vec{c'}) \neq \mathbf{U}$ are computed (this is a single call in the implementation) and stored in the cache. Otherwise, no external source call is needed. It is then checked if the resulting nogood is still faithful in Part (c), and N is replaced by its reduced equivalent in S in this case. Formally:

Proposition 7. *For a set S of faithful io-nogoods with equal input parts and distinct output parts, Algorithm 2 yields exactly one faithful io-nogood $N' \in minimize(N)$ for each $N \in S$.*

5.2 Divide-and-Conquer Strategy for Nogood Minimization

Even when io-nogoods with the same input parts are minimized simultaneously, removing each literal from the respective input separately and checking the output of the corresponding oracle function may result in a large number of external calls, which directly depends on the length of the

Algorithm 3: QuickXplain Nogood Minimization

Input: A faithful io-nogood $N = \{\sigma_1 a_1, \ldots, \sigma_n a_n, \sigma_{n+1} e_{\&g[\vec{p}]}(\vec{c})\}$
Output: A minimal faithful io-nogood $N' \in minimize(N)$

if $f_{\&g}(\emptyset, \vec{p}, \vec{c}) = \overline{\sigma_{n+1}}$ **then return** $\{\sigma_{n+1} e_{\&g[\vec{p}]}(\vec{c})\}$ (a)
return $quickXplain(\emptyset, \emptyset, N_I) \cup \{\sigma_{n+1}\, e_{\&g[\vec{p}]}(\vec{c})\}$ (b)

function $quickXplain(B, D, N')$
  if $D \neq \emptyset$ *and* $f_{\&g}(B, \vec{p}, \vec{c}) = \overline{\sigma_{n+1}}$ **then return** $\emptyset$ (c)
  if $|N'| = 1$ **then return** N' (d)
  Partition N' into two non-empty sets N_1 and N_2 (e)
  $D_1 \leftarrow quickXplain(B \cup N_2, N_2, N_1)$
  $D_2 \leftarrow quickXplain(B \cup D_1, D_1, N_2)$
  return $D_1 \cup D_2$

input part. In cases where io-nogoods are large or the external evaluation requires a lot of time, the additional computational effort required for nogood minimization may outweigh the positive effect of obtaining smaller nogoods, or even make minimization infeasible. While in the worst case, i.e. when an io-nogood is already minimal, this situation cannot be improved, it can be more efficient to remove several literals from a nogood at once before evaluating the external source when the input part contains many irrelevant literals.

The QUICKXPLAIN algorithm, which has been introduced by Junker (2004) for efficiently computing minimal conflict sets in the context of constraint programming, can be used for this purpose as an alternative algorithm for minimizing io-nogoods. It implements a divide-and-conquer strategy producing a binary search tree, and can be employed for computing a minimal nogood from a given io-nogood more efficiently if the nogood contains many irrelevant literals, especially when the input part of the io-nogood is large. In this way, given an io-nogood with input part of size n, instead of n calls to the oracle function, only $O(log_2 n)$ external calls are required. However, in the worst case, i.e. when no literal can be removed from a given nogood, $O(n)$ calls are necessary. Consequently, the algorithm can behave either better or worse than a sequential algorithm, depending on the properties of the io-nogoods that are minimized.

Algorithm 3 is a variant of the QUICKXPLAIN algorithm as presented by Shchekotykhin, Jannach, and Schmitz (2015), adapted to our specific setting of io-nogood minimization. The algorithm receives a faithful io-nogood $N = \{\sigma_1 a_1, \ldots, \sigma_n a_n, \sigma_{n+1} e_{\&g[\vec{p}]}(\vec{c})\}$ and first checks whether the literal in the output part N_O depends on a non-empty input part N_I, in Part (a). Subsequently, a recursive function is called in Part (b), which during its execution checks if different subsets of N_I imply the same external replacement literal in N_O as N_I.

The first argument B of the function $quickXplain(B, D, N')$ contains the current subset of the input part N_I wrt. which the oracle function $f_{\&g}(B, \vec{p}, \vec{c})$ is evaluated in Part (c). The second argument D indicates if the oracle function needs to be evaluated for a given B, which is only the case if D is non-empty as only then B has changed since the last external evaluation. If B is determined to imply the same truth value for $e_{\&g[\vec{p}]}(\vec{c})$ as N_I in Part (c), no further literals from N' need to be added to B and thus, the empty set is returned. In case the subset N' of the input part N_I of literals that can still be added to B to obtain the correct value for $f_{\&g}(B, \vec{p}, \vec{c})$ is a singleton, it is returned in Part (d). Finally, in Part (e), the provided subset N' of the input part N_I is partitioned into two nonempty sets N_1 and N_2, and the function is called recursively, once for each partition

N_1 and N_2 of N'. The result D_1 of the first call, where N_1 is provided as new subset of the input part N_I, contains all literals from N_1 that need to be added to $B \cup N_2$ such that the oracle function still evaluates to σ_{n+1}. Similarly, all literals from N_I that need to be added to $B \cup D_1$ such that the oracle function still evaluates to σ_{n+1} are stored in D_2. As a result, no literal can be removed from D_1 or D_2 such that $f_{\&g}(B \cup D_1 \cup D_2, \vec{p}, \vec{c}) = \sigma$ still holds, and the input part $D_1 \cup D_2$, which together with $\{\sigma_{n+1} e_{\&g[\vec{p}]}(\vec{c})\}$ yields a minimal io-nogood, is returned.

The computation of a minimal io-nogood by Algorithm 3 is illustrated by the following example.

Example 12 (Ex. 8 cont'd)**.** *Reconsider $f'_{\&geq}$ from Example 8 and the faithful io-nogood $N = \{\mathbf{F}edge(a,a), \mathbf{T}edge(b,b), \mathbf{T}edge(a,b), \mathbf{T}edge(b,a), \mathbf{F}e_{\&geq[edge,2]}()\}$. When Algorithm 3 is executed with input N, the function call $quickXplain(\emptyset, \emptyset, N_I)$ is performed with $N_I = \{\mathbf{F}edge(a,a), \mathbf{T}edge(b,b), \mathbf{T}edge(a,b), \mathbf{T}edge(b,a)\}$. Since $D = \emptyset$ and $|N| \neq 1$ hold wrt. the first call, $N' = N_I$ is partitioned into two sets, for example $N_1 = \{\mathbf{F}edge(a,a), \mathbf{T}edge(b,b)\}$ and $N_2 = \{\mathbf{T}edge(a,b), \mathbf{T}edge(b,a)\}$.*

Subsequently, the first recursive call of the function $quickXplain$ returns $\emptyset$, which is assigned to D_1, as $f'_{\&geq}(N_2, edge, 2) = \mathbf{T}$, i.e. $\{\mathbf{T}edge(a,b), \mathbf{T}edge(b,a), \mathbf{F}e_{\&geq[edge,2]}()\} \subset N$ is still a faithful io-nogood. Accordingly, the second recursive call in Part (e) corresponds to the call $quickXplain(\emptyset, \emptyset, N_2)$, in which N_2 is partitioned again into $\{\mathbf{T}edge(a,b)\}$ and $\{\mathbf{T}edge(b,a)\}$. Because each of the sets has cardinality 1 but none of them suffices to derive $\&geq[edge,2]()$, N_1 and N_2 are returned from the two recursive calls in Part (e), respectively. Thus, $N_1 \cup N_2$ is returned by the second recursive call in the outer function call, which is assigned to D_2. Consequently, $\emptyset \cup \{\mathbf{T}edge(a,b), \mathbf{T}edge(b,a)\}$ is returned by the function called in Part (b). Finally, the minimal io-nogood $\{\mathbf{T}edge(a,b), \mathbf{T}edge(b,a), \mathbf{F}e_{\&geq[edge,2]}()\}$ is returned by Algorithm 3.

Algorithm 3 always finds a minimal faithful io-nogood, which follows directly from Proposition 6 and Theorem 1 by Junker (2004):

Proposition 8. *Given a faithful io-nogood N, Algorithm 3 terminates and returns exactly one faithful io-nogood $N' \in minimize(N)$.*

Like Algorithm 2, Algorithm 3 returns exactly one minimal io-nogood for a given input. A straightforward way to obtain multiple minimal io-nogoods consists in re-running Algorithm 3 with different partition heuristics in Part (e); and every minimal io-nogood can be obtained in this way.

6. Interleaving External Evaluation and Unfounded Set Search

So far, we have only considered external evaluations based on partial assignments which are performed during the search for compatible sets. As described in Section 2, not every compatible set is also an answer set, due to the possibility of cyclic support involving external atoms, and an additional minimality check (cf. Part (d) in Algorithm 1) is required for finding answer sets of HEX-programs. The basic approach for ensuring minimality of answer sets wrt. the FLP-reduct (called *explicit FLP check* by Eiter et al., 2014a) consists in explicitly constructing the FLP-reduct for a given compatible set $\mathbf{A}$ and searching for a model $\mathbf{A}'$ of the reduct for which $\mathbf{A}' \subseteq \mathbf{A}$ holds. In general, it is also necessary to evaluate external atoms again for finding smaller models of the FLP-reduct because their truth value might change when the truth value of some ordinary atoms is switched from true to false. In the previous approach, similar as in the case of the main search, this evaluation could only be performed after the complete input to an external atom in a potential smaller model had been decided since input atoms were not allowed to be unassigned.

In this section, we discuss how, based on three-valued assignments, the evaluation of external atoms can be interleaved with the search in the minimality check. As before, the goal is to increase the efficiency by evaluating external atoms as early as possible and thus, to potentially avoid many wrong guesses. Improving the efficiency of the minimality check for HEX-programs is of special interest as the check accounts for many programs to a large share of the total runtime. In addition, the models of the FLP-reduct (built for a complete assignment) often outnumber the compatible sets of the program (cf. Eiter et al., 2014a), and for each such set the guesses for the truth values of external atoms need to be verified.

Here, we consider a more sophisticated variant of the FLP check that utilizes the concept of *unfounded sets* in order to ensure minimality of answer sets in HEX, i.e., that they amount to minimal models of the FLP-reduct with the complete interpretation encoded by a compatible set. An unfounded set of a HEX-program Π wrt. an assignment $\mathbf{A}$ is a set of atoms that can be jointly set to false without violating any rule in Π because they only circularly support each other wrt. $\mathbf{A}$. Formally:

Definition 12 (adapted from Eiter et al., 2014a)**.** *Let Π be a HEX-program and let $\mathbf{A}$ and $\mathbf{U}$ be complete assignments over $A(\Pi)$. Then, $\mathbf{U}$ is an* unfounded set *for Π wrt. $\mathbf{A}$ if, for each rule r with $H(r) \cap \{a \mid \mathbf{T}a \in \mathbf{U}\} \neq \emptyset$, at least one of the following holds, where $\mathbf{A} \dot\cup \neg.\mathbf{U} = (\mathbf{A} \setminus \{\mathbf{T}a \mid \mathbf{T}a \in \mathbf{U}\}) \cup \{\mathbf{F}a \mid \mathbf{T}a \in \mathbf{U}\}$:*

(1) some literal of $B(r)$ is false wrt. $\mathbf{A}$,

(2) some literal of $B(r)$ is false wrt. $\mathbf{A} \dot\cup \neg.\mathbf{U}$, or

(3) some atom of $H(r) \setminus \{a \mid \mathbf{T}a \in \mathbf{U}\}$ is true wrt. $\mathbf{A}$.

Note that unlike previous literature, we define unfounded sets as complete assignments rather than sets of atoms. This is in order to make the operator $\dot\cup$ reusable for additional results below. However, conceptually an unfounded set $\mathbf{U}$ still represents a set of atoms, given by the true atoms $\mathbf{T}a \in \mathbf{U}$.

Answer sets of a HEX-program Π correspond exactly to those models M of Π where the true part of M does not intersect with any unfounded set for Π wrt. M, i.e. $\{\mathbf{T}a \mid \mathbf{T}a \in M \cap \mathbf{U}\} = \emptyset$ for every unfounded set $\mathbf{U}$ for Π wrt. M (Faber, 2005; Eiter et al., 2014a). Eiter et al. (2014a) showed that ensuring the absence of unfounded sets is a more efficient strategy for verifying minimality than applying the explicit FLP check, due to the fact that smaller models of the FLP-reduct do not have to be generated explicitly in the former case. However, truth values of external atoms still need to be checked as described above to verify that candidate unfounded sets that have been detected actually constitute unfounded sets.

Example 13 (Ex. 5 cont'd)**.** *Reconsider the program $\Pi = \{p \leftarrow \&id[p]().\}$ from Example 5. As observed, $\mathbf{A} = \{\mathbf{T}p\}$ is not an answer set of the program since it is not a subset-minimal model of $f\Pi^{\mathbf{A}} = \Pi$. This is because there is an unfounded set $\mathbf{U} = \{\mathbf{T}p\}$, which intersects with the true atoms in $\mathbf{A}$: the only rule whose head intersects with $\{a \mid \mathbf{T}a \in \mathbf{U}\}$ is $p \leftarrow \&id[p]()$, for which condition (2) is satisfied.*

To enable external checks at any point during the search for unfounded sets, even before a candidate unfounded set has been detected, we introduce a novel algorithm for unfounded set checking that exploits external evaluations based on partial assignments. Subsequently, we show the correctness and completeness of the new algorithm. At this, interleaving unfounded set search with

Algorithm 4: HEX-UFSCheck

Input: A HEX-program Π, a complete assignment $\mathbf{A}$, a set of nogoods ∇ of Π
Output: *true* if the true part of $\mathbf{A}$ intersects with an unfounded set for Π wrt. $\mathbf{A}$ and *false* otherwise, learned nogoods added to ∇

$\Omega'_\Pi \leftarrow \Omega_\Pi \cup \mathcal{A}_\mathbf{A} \cup \{\mathcal{T}_\Omega(N) \mid N \text{ is an io-nogood in } \nabla\}$ `// SAT instance with assumptions and io-nogoods from main search`
$\mathbf{S} \leftarrow \{\mathbf{U}a \mid a \in A(\Omega'_\Pi)\}$ `// all atoms unassigned`
$dl \leftarrow 0$ `// decision level`
while *true* **do**
  $\mathbf{S} \leftarrow$ Propagation$(\Omega'_\Pi, \mathbf{S})$ **(a)**
  if *some nogood in* Ω'_Π *violated by* $\mathbf{S}$ **then** **(b)**
    if $dl = 0$ **then return** *false*
    Analyze conflict, add learned nogood to Ω'_Π, set dl to backjump level
  else if $\mathbf{S}$ *is complete* **then** **(c)**
    $isUFS \leftarrow true$
    for *all external atoms* $\&g[\vec{p}](\vec{c})$ *in* Π **do**
      $\nabla \leftarrow \nabla \cup \Lambda(\&g[\vec{y}], \mathbf{A} \dot\cup \neg.\mathbf{S})$
      $\Omega'_\Pi \leftarrow \Omega'_\Pi \cup \{\mathcal{T}_\Omega(N) \mid N \in \Lambda(\&g[\vec{y}], \mathbf{A} \dot\cup \neg.\mathbf{S})\}$
      if $\mathbf{T}e_{\&g[\vec{p}]}(\vec{c}) \in \mathbf{S}$, $\mathbf{A} \not\models \&g[\vec{p}](\vec{c})$ *and* $\mathbf{A} \dot\cup \neg.\mathbf{S} \not\models \&g[\vec{p}](\vec{c})$ **then**
        $isUFS \leftarrow false$
      if $\mathbf{F}e_{\&g[\vec{p}]}(\vec{c}) \in \mathbf{S}$, $\mathbf{A} \models \&g[\vec{p}](\vec{c})$ *and* $\mathbf{A} \dot\cup \neg.\mathbf{S} \models \&g[\vec{p}](\vec{c})$ **then**
        $isUFS \leftarrow false$
    if $isUFS$ **then**
      Let N be a nogood learned from the UFS
      $\nabla \leftarrow \nabla \cup \{N\}$
      if $\{\mathbf{T}a \mid \mathbf{T}a \in \mathbf{A} \cap \mathbf{S}\} \neq \emptyset$ **then return** *true*
    else
      $\Omega'_\Pi \leftarrow \Omega'_\Pi \cup \{\mathbf{S}\}$
  else if *Heuristics evaluates* $\&g[\vec{y}]$ *and* $\Lambda(\&g[\vec{y}], \mathbf{A} \dot\cup \neg.\mathbf{S}) \not\subseteq \nabla$ **then** **(d)**
    $\nabla \leftarrow \nabla \cup \Lambda(\&g[\vec{y}], \mathbf{A} \dot\cup \neg.\mathbf{S})$
    $\Omega'_\Pi \leftarrow \Omega'_\Pi \cup \{\mathcal{T}_\Omega(N) \mid N \in \Lambda(\&g[\vec{y}], \mathbf{A} \dot\cup \neg.\mathbf{S})\}$
  else **(e)**
    Guess σa with $\sigma \in \{\mathbf{T}, \mathbf{F}\}$ for some variable a with $\mathbf{U}a \in \mathbf{S}$
    $dl \leftarrow dl + 1$
    $\mathbf{S} \leftarrow (\mathbf{S} \setminus \{\mathbf{U}a\}) \cup \{\sigma a\}$

external evaluations can initiate backjumping as soon as it can be determined that guesses for external atoms violate the conditions for unfounded sets. So far, this could only be detected by means of a post-check. As before, input-output relations learned from oracle calls wrt. partial assignments can also be exploited to avoid wrong guesses in the further unfounded set search.

We start by providing background on the previous unfounded set check for HEX-programs (Eiter et al., 2014a), which we extend to partial evaluations in the following.

6.1 Background on Unfounded Set Search

For detecting unfounded sets of a HEX-program Π wrt. a complete assignment $\mathbf{A}$, Eiter et al. (2014a) introduced an encoding Ω_Π that is represented by a set of nogoods such that solutions to the encoding that are compatible with the semantics of external sources used in the program Π correspond exactly to the unfounded sets of Π wrt. $\mathbf{A}$. The encoding is uniform wrt. all executions of the unfounded set check, i.e. it does not depend on the current assignment and thus, only needs to be generated once. Accordingly, a compatible set $\mathbf{A}$ for which the check is performed needs to be injected by adding a set of so-called *assumptions* $\mathcal{A}_\mathbf{A}$, represented by a consistent set of signed literals, that fix the truth values of dedicated atoms. In this way, the encoding does not have to be regenerated for each compatible set from scratch, and in an implementation, assumptions can be treated in a special way such that part of the solver state can be maintained when assumptions are changed. As a result, a SAT solver can be utilized to detect unfounded set candidates by searching for a solution to Ω_Π with assumptions $\mathcal{A}_\mathbf{A}$.

Because external replacement atoms in Ω_Π do not encode the truth values of external atoms wrt. a solution $\mathbf{S}$ of the SAT encoding, but relative to a compatible set modified by $\mathbf{S}$, faithful io-nogoods learned wrt. $\mathbf{S}$ cannot be added directly to the encoding. For this reason, Eiter et al. (2014a) defined a nogood transformation $\mathcal{T}_\Omega$ that ranges over io-nogoods and yields corresponding nogoods that imply the correct truth value for external replacement atoms in Ω_Π. We do not go into the details of the particular encoding and the nogood transformation here, as they are not relevant for our purposes; we refer to the work of Eiter et al. (2014a) for more information.

6.2 Extension to Partial Assignments

As in the search for compatible sets, we can also add the input-output relations that are learned from external evaluations based on partial assignments for the SAT encoding in form of nogoods to the SAT solver. However, here we have to take into account that external replacement atoms do not encode the truth values of external atoms under a partial assignment in the solver, but represent their evaluation relative to the current compatible set modified by the respective partial assignment for Ω_Π with assumptions $\mathcal{A}_\mathbf{A}$. For this reason, we generalize the definition of $\mathbf{A} \dot\cup \neg.\mathbf{X}$ as follows, considering also partial assignments for the SAT encoding.

Definition 13. *Given a complete assignment* $\mathbf{A}$ *and a partial assignment* $\mathbf{X}$*, let* $\mathbf{A} \dot\cup \neg.\mathbf{X} = (\mathbf{A} \setminus \{\mathbf{T}a \mid \mathbf{T}a \in \mathbf{X} \text{ or } \mathbf{U}a \in \mathbf{X}\}) \cup \{\mathbf{F}a \mid \mathbf{T}a \in \mathbf{X}\} \cup \{\mathbf{U}a \mid \mathbf{U}a \in \mathbf{X} \text{ and } \mathbf{T}a \in \mathbf{A}\}$.

In contrast to Definition 12, where $\mathbf{U}$ is considered to be a complete assignment, atoms which are unassigned in $\mathbf{X}$ and true in $\mathbf{A}$ remain unassigned in $\mathbf{A} \dot\cup \neg.\mathbf{X}$; those atoms can potentially be set to false in $\mathbf{A} \dot\cup \neg.\mathbf{X}'$ wrt. some assignment $\mathbf{X}' \succeq \mathbf{X}$. Atoms that are true in $\mathbf{X}$ and $\mathbf{A}$ are false under $\mathbf{A} \dot\cup \neg.\mathbf{X}$ as before.

Example 14. *Consider the complete assignment* $\mathbf{A} = \{\mathbf{T}p, \mathbf{T}q, \mathbf{T}r\}$ *and the assignment* $\mathbf{X} = \{\mathbf{T}p, \mathbf{F}q, \mathbf{U}r\}$*. We then obtain* $\mathbf{A} \dot\cup \neg.\mathbf{X} = \{\mathbf{F}p, \mathbf{T}q, \mathbf{U}r\}$.

We note that due to assignment monotonicity of three-valued oracle functions, extending in a partial assignment $\mathbf{A} \dot\cup \neg.\mathbf{X}$ the set $\mathbf{X}$ does not change the value of an oracle function call that is determined (i.e., yields true or false). Formally:

Proposition 9. *Let* $\mathbf{A}$ *be a complete assignment,* $\mathbf{X}$ *be a partial assignment, and* $f_{\&g}$ *be an assignment monotonic three-valued oracle function. Then,* $f_{\&g}(\mathbf{A} \dot\cup \neg.\mathbf{X}, \vec{p}, \vec{c}) = X$*,* $X \in \{\mathbf{T}, \mathbf{F}\}$*, implies for every assignment* $\mathbf{X}' \succeq \mathbf{X}$ *that* $f_{\&g}(\mathbf{A} \dot\cup \neg.\mathbf{X}', \vec{p}, \vec{c}) = X$.

The proposition implies that early external evaluations during the unfounded set search wrt. an assignment $\mathbf{A} \dot{\cup} \neg.\mathbf{X}$ yield nogoods N s.t. $f_{\&e}(\mathbf{A} \dot{\cup} \neg.\mathbf{X}', \vec{p}, \vec{c}) = \overline{\sigma(N_O)}$ for all extensions $\mathbf{X}'$ of $\mathbf{X}$. The fact that faithful io-nogoods added via the transformation $\mathcal{T}_\Omega$ to the encoding Ω_Π do not remove unfounded sets, as stated in Proposition 15 by Eiter et al. (2014a), is based on the latter property.

We are now ready to present our new algorithm for detecting unfounded sets which also exploits learning wrt. partial assignments.

6.3 New Algorithm for Unfounded Set Detection

Our new procedure for detecting unfounded sets, which is formalized by Algorithm 4, extends the unfounded set check procedure described by Eiter et al. (2014a).[8] It is used in Part (d) of Algorithm 1 in order to check whether a compatible set $\mathbf{A}$ for a HEX-program Π is an answer set, i.e. its true part does not intersect with an unfounded set for Π wrt. $\mathbf{A}$.

Algorithm 4 receives as input a HEX-program Π, a complete assignment $\mathbf{A}$ representing a compatible set of Π, and a set ∇ of nogoods that have been generated by Algorithm 1. It returns *true* if Π has an unfounded set wrt. $\mathbf{A}$ that intersects with the true part of $\mathbf{A}$, and false otherwise, i.e. when $\mathbf{A}$ is an answer set of Π. At first, the assumptions $\mathcal{A}_\mathbf{A}$ and the transformations of io-nogoods already learned in the main search are added to the encoding Ω_Π. In our implementation, elements in $\mathcal{A}_\mathbf{A}$ are marked as assumptions and hence, they can be removed from the encoding without the need to reinitialize the SAT solver completely.

Similar to Algorithm 1, Algorithm 4 explores the search space in one loop based on the well-known *CDCL procedure* (Marques-Silva, Lynce, & Malik, 2009), where unit propagation is performed in Part (a), conflict learning and backjumping in Part (b), and guessing in Part (e). However, to take the semantics of external atoms into account, there are two additional parts integrated into the CDCL procedure, where the first is necessary to ensure correctness of the algorithm, while the second potentially increases its efficiency.

On the one hand, in Part (c), after a solution $\mathbf{S}$ to Ω'_Π has been found, it is checked for each $\&g[\vec{p}](\vec{c})$ in Π whether the truth value assigned to the replacement atom $e_{\&g[\vec{p}](\vec{c})}$ is compatible with the evaluation of the corresponding oracle function under $\mathbf{A} \dot{\cup} \neg.\mathbf{S}$. It has been shown that when the truth value of an external atom $\&g[\vec{p}](\vec{c})$ under $\mathbf{A}$ coincides with the one for $e_{\&g[\vec{p}](\vec{c})}$ assigned by $\mathbf{S}$, the check for ensuring that $\mathbf{S}$ represents an unfounded set can be skipped (cf. Eiter et al., 2014a).

If a solution $\mathbf{S}$ passes the external checks, an unfounded set for Π wrt. $\mathbf{A}$ has been detected and the algorithm returns *true* in case $\mathbf{S}$ intersects with the true part of the complete assignment $\mathbf{A}$; otherwise, $\mathbf{S}$ is added to Ω'_Π and the search continues. The io-nogoods learned from the external evaluations are added to the nogood store ∇ for use in the search for compatible sets and to Ω'_Π via the nogood transformation $\mathcal{T}_\Omega$, in order to avoid wrong guesses for replacement atoms in the further unfounded set search.

On the other hand, external evaluations can also be performed based on partial assignments for Ω'_Π, which are triggered by a heuristics in Part (d). Accordingly, the respective oracle function is evaluated in Part (c) under an assignment $\mathbf{A} \dot{\cup} \neg.\mathbf{S}$ as in Definition 13. As before, learned nogoods are added to ∇ and (via the nogood transformation) to Ω'_Π, respectively.

8. The algorithm by Eiter et al. (2014a) further implements a decision criterion that allows to skip the UFS check entirely for some syntactic HEX-program classes where answer set existence is known to be in NP. The criterion is independent of the techniques presented in this paper and can thus be readily used also for the extended algorithm with no further change; we thus omit this criterion here for simplicity.

While the details of how the learned nogood N is constructed are not relevant in the following, we stress that using the unfounded set itself as learned nogood is in general *not* correct.

Example 15. *Consider the program* $\Pi = \{a \vee b \leftarrow; c \leftarrow b; b \leftarrow c\}$*, which has two answer sets* $\mathbf{A}_1 = \{\mathbf{T}a, \mathbf{F}b, \mathbf{F}c\}$ *and* $\mathbf{A}_2 = \{\mathbf{F}a, \mathbf{T}b, \mathbf{T}c\}$*. Note that* $U = \{b, c\}$ *is an unfounded set of* Π *wrt.* $\mathbf{A} = \{\mathbf{T}a, \mathbf{T}b, \mathbf{T}c\}$*. Using* $\{\mathbf{T}b, \mathbf{T}c\}$ *as learned nogood (constructed from the atoms in* U*) would eliminate the answer set* $\mathbf{A}_2$*. Informally, this is the case because* U *is unfouneded* only wrt. the current assignment*, which must be respected in nogood learning. In this case, the nogood learned from* U *is* $N = \{\mathbf{T}a, \mathbf{T}b\}$ *(or, alternatively,* $N' = \{\mathbf{T}a, \mathbf{T}c\}$*).*

For details on the construction of N we refer to the work of Eiter et al. (2014a). Note that Algorithm 4 is parametric on the learning function Λ used in Parts (c) and (d). Because the nogood transformation $\mathcal{T}_\Omega$ by Eiter et al. (2014a) can only be applied to faithful io-nogoods, we assume that Λ only returns faithful io-nogoods in Algorithm 4. That is, all other nogoods returned by the learning function Λ are simply ignored. In practice, we employ the learning functions Λ_u and Λ_{mu}.

6.4 Properties

The following proposition, adapted from Theorem 10 by Eiter et al. (2014a), states that by the checks in Part (c) of Algorithm 4, we can determine whether a *complete* solution $\mathbf{S}$ corresponds to an unfounded set for Π wrt. $\mathbf{A}$:

Proposition 10. *Let* Π *be a* HEX*-program and let* $\mathbf{A}$ *be a complete assignment over* $A(\Pi)$*. If there is a solution* $\mathbf{S}$ *for* Ω_Π *with assumptions* $\mathcal{A}_\mathbf{A}$ *such that for all external atoms* $\&g[\vec{p}](\vec{c})$ *in* Π *it holds that*

(1) $\mathbf{T}e_{\&g[\vec{p}](\vec{c})} \in \mathbf{S}$ *and* $\mathbf{A} \not\models \&g[\vec{p}](\vec{c})$ *implies* $\mathbf{A} \dot\cup \neg.\mathbf{S} \not\models \&g[\vec{p}](\vec{c})$*, and*

(2) $\mathbf{F}e_{\&g[\vec{p}](\vec{c})} \in \mathbf{S}$ *and* $\mathbf{A} \models \&g[\vec{p}](\vec{c})$ *implies* $\mathbf{A} \dot\cup \neg.\mathbf{S} \models \&g[\vec{p}](\vec{c})$*,*

then $\mathbf{U} = \{Xa \mid a \in A(\Pi), Xa \in \mathbf{S}, X \in \{\mathbf{T}, \mathbf{F}\}\}$ *is an unfounded set for* Π *wrt.* $\mathbf{A}$*.*

Moreover, we can show that for every unfounded set $\mathbf{U}$ for Π wrt. $\mathbf{A}$ where $\mathbf{U}$ intersects with the true part of $\mathbf{A}$, a solution to Ω_Π with assumptions $\mathcal{A}_\mathbf{A}$ can be generated that passes the checks in Part (c) of Algorithm 4, and that the nogoods added in Part (d) of Algorithm 4 do not eliminate the solution:

Proposition 11. *Let* Π *be a* HEX*-program, let* $\mathbf{A}$ *be a complete assignment over* $A(\Pi)$ *and suppose Algorithm 4 is executed with* Π *and* $\mathbf{A}$ *as inputs. If there is an unfounded set* $\mathbf{U}$ *for* Π *wrt.* $\mathbf{A}$ *s.t.* $\{\mathbf{T}a \mid \mathbf{T}a \in \mathbf{A} \cap \mathbf{U}\} \neq \emptyset$*, then there is a solution* $\mathbf{S}$ *for* Ω_Π *with assumptions* $\mathcal{A}_\mathbf{A}$*, s.t.* $\{\mathbf{T}a \mid \mathbf{T}a \in \mathbf{A} \cap \mathbf{S}\} \neq \emptyset$*, that satisfies conditions (1) and (2) of Proposition 10 and all transformed nogoods* $\mathcal{T}_\Omega(N)$ *added to* Ω'_Π *in Part (d) of Algorithm 4.*

We remark that in case the learning function Λ_u is used in Part (d), backjumping is triggered by the added nogoods as soon as it can be determined that a partial assignment cannot be extended to a solution satisfying conditions (1) and (2). However, we refrain from a formal statement and proof of this behavior in the special case, as it would require to delve into the very details of the uniform encoding and the particular nogood transformation (the respective conflict involves a transformed nogood).

Example 16. *Consider the* HEX*-program* $\Pi = \{r \leftarrow \&id[q](). \ ; \ q \leftarrow . \ ; \ p \leftarrow \&id[p]().\}$*, the complete assignment* $\mathbf{A} = \{\mathbf{T}p, \mathbf{T}q, \mathbf{T}r\}$ *and a partial assignment* $\mathbf{S}$ *s.t.* $\mathbf{S} \supseteq \{\mathbf{F}e_{\&id[\vec{q}]}(), \mathbf{F}e_{\&id[\vec{r}]}(), \mathbf{T}r, \mathbf{F}q, \mathbf{U}p\}$*. Note that* $\mathbf{S}$ *cannot be extended s.t. it corresponds to an unfounded set of* Π *wrt.* $\mathbf{A}$*. Accordingly, by performing external evaluations wrt.* $\mathbf{S}$*, we find that it violates condition (2) of Proposition 10 as* $\mathbf{F}e_{\&id[q]}() \in \mathbf{S}$*,* $\mathbf{A} \models \&id[q]()$ *and* $f_{\&id}(\mathbf{A} \dot{\cup} \neg.\mathbf{S}, q) = \mathbf{T}$*. This demonstrates that we can detect that* $\mathbf{S}$ *cannot be extended to a solution corresponding to an unfounded set without constructing a complete solution to* Ω'_Π*.*

Correctness and completeness of Algorithm 4 can be derived from the facts that it returns only solutions for Ω_Π with assumptions $\mathcal{A}_\mathbf{A}$ that satisfy conditions (1) and (2) of Proposition 10, and that no such solution is removed due to the nogoods learned in Part (d).

Theorem 2 (Soundness and Completeness of Algorithm 4)**.** *Given a* HEX*-program* Π *and a complete assignment* $\mathbf{A}$ *over* $A(\Pi)$ *as inputs, Algorithm 4 returns* $true$ *if there is an unfounded set* $\mathbf{U}$ *for* Π *wrt.* $\mathbf{A}$ *s.t.* $\{\mathbf{T}a \mid \mathbf{T}a \in \mathbf{A} \cap \mathbf{U}\} \neq \emptyset$*, and false otherwise.*

Thus, by employing Algorithm 4, we are now also able to exploit partial assignments for evaluating external sources at any point during the unfounded set search, and for learning corresponding io-nogoods that can decrease the number of unfounded set candidates which need to be generated.

7. Implementation and Evaluation

In this section, we present the results of an experimental evaluation of our techniques. To this end, we integrated them into DLVHEX 2.5.0 with GRINGO 4.4.0 and CLASP 3.1.1 as backends. The system's website is http://www.kr.tuwien.ac.at/research/systems/dlvhex; its source code is available from https://github.com/hexhex. External sources are realized as plugins to the DLVHEX reasoner, which are lazily called by the reasoner only when the value of an external atom under an assignment needs to be known.

We remark that although CLINGO 5 is known for its theory solving capabilities (Gebser, Kaminski, Kaufmann, Ostrowski, Schaub, & Wanko, 2016), also previous versions of GRINGO resp. CLASP had similar features, which are exploited by DLVHEX; CLINGO 5 makes these features more accessible. While we plan to upgrade our backend to CLINGO 5 as part of future work, this will mainly simplify the interfaces, but will not allow for algorithmic improvements, hence there is no interference with the techniques presented in this paper. For a more detailed discussion of the differences to CLINGO 5 we refer to Section 8. We remark that DLVHEX can also handle ordinary ASP programs with some well-known extensions such as aggregates, weak constraints, choice rules, etc. The overhead when evaluating an ASP program with DLVHEX compared to ordinary ASP solvers is typically negligible since after the initialization, the program is basically directly handed to the GRINGO and CLASP backends.

7.1 Experimental Setup

In the following we first describe the platform used for carrying out our benchmarks and the configurations we are going to compare. We then describe the benchmark suite used for the evaluation.

7.1.1 EVALUATION PLATFORM

All benchmarks were run on a Linux machine with two 12-core AMD Opteron 6238 SE CPUs and 512 GB RAM; the timeout was 300 seconds and the memout was 8 GB per instance. We used the *HTCondor* load distribution system (HTCondor Website, 2018) to ensure robust runtimes (i.e., deviations of runs on the same instance are negligible). The average runtime of 50 instances per problem size is reported (in seconds) for computing all answer sets respectively the first answer set; the number of timeouts is shown in parentheses and furthermore, the average number of solutions, where '$\geq$' respects timeouts. Line chart representations of the benchmark results can be found in Appendix B. All instances and details on the experiments can be found at http://www.kr.tuwien.ac.at/research/projects/inthex/partialeval.

7.1.2 BENCHMARK CONFIGURATIONS

Naturally, there is a tradeoff between the information that can be gained from additional external evaluations under partial assignments during solving, and the runtime that has to be invested for the respective external calls. For this reason, we used different heuristics for controlling the number of external evaluations, and investigated the impact of 10 different solver configurations.

Initially, we tested 3 heuristics for additional external source calls during the main search for compatible sets (cf. Algorithm 1, Part (e)) without nogood minimization, namely

- **never**: no additional calls, i.e. DLVHEX without the new techniques;
- **periodic**: partial evaluation at each 10^{th} heuristics call; and
- **always**: partial evaluation at every heuristics call.

Moreover, we tested two heuristics for interleaving external evaluations with the search for unfounded sets (cf. Algorithm 4, Part (d)), namely

- **ufs-p**: partial evaluation at each 10^{th} heuristics call during unfounded set search, and
- **ufs-a**: partial evaluation at every heuristics call during unfounded set search.

In addition, we investigated the effect of enabling external evaluations based on partial assignments both during the main search and the unfounded set search, i.e. combining configurations **always** and **ufs-a**. We then tested nogood minimization instead of additional calls (i.e., only for complete assignments), where we used the algorithm for simultaneous nogood minimization (cf. Algorithm 2) and the QUICKXPLAIN algorithm (cf. Algorithm 3), respectively, for minimizing either

- *all* nogoods in conditions **ngm** and **qxp**, or
- the currently *conflicting* ones, i.e. those which violate the current solver assignment and trigger backjumping, in conditions **ngm-c** and **qxp-c**.

For benchmarks where external atoms have output values, we also compared simultaneous minimization with sequential minimization (**ngm-sq**), i.e. minimizing each io-nogood separately. We omit results for minimization combined with **periodic**, **always**, **ufs-p** or **ufs-a**, as this was always significantly slower than some other configuration (due to many more external calls with little gain).

7.1.3 BENCHMARK PROBLEMS

We have considered encodings of three problems in the evaluation:

- *Pseudo-boolean* (*PB-*)problems, also known as *0-1 integer linear programs*, representing linear constraints over boolean variables, which are among Karp's famous 21 NP-complete problems (Karp, 1972).
- Assignment of taxi drivers to customers under constraints, where queries to an external ontology, expressed in the lightweight *description logic* (*DL*) *DL-Lite*, are made via external atoms to find out locations and classify customers and drivers (*Taxi Assignment with Ontology Access*) (Eiter, Fink, Redl, & Stepanova, 2014b; Eiter, Fink, & Stepanova, 2016). Note that despite a similar scenario, our benchmark is different from the one used by Eiter et al. (2014b), as it admits multiple solutions due to nondeterministic guessing of customer assignments.
- Different variants of the well-known *Strategic Companies* problem (Cadoli, Eiter, & Gottlob, 1997), which is popular with ASP competitions, extended with externally stored conflicts among companies (*Conflicting Strategic Companies*) and externally computed control among companies based on shares (*Strategic Companies (with Nonmonotonic) External Controls Relation*).

The problems have different characteristics with regard to the computational complexity and the external atoms and their usage. While query answering wrt. the DL-Lite ontology used in our taxi assignment benchmark is tractable (Calvanese, De Giacomo, Lembo, Lenzerini, & Rosati, 2007), and solving PB-problems is NP-complete, computing strategic companies is located at the second level of the polynomial hierarchy. Moreover, the general learning function Λ_u is used for the PB-problems benchmark and the variant of the strategic companies benchmark where a nonmonotonic external control relation is added. Due to monotonicity of external sources, the learning function Λ_{mu} can be utilized in all other benchmarks. A further difference consists in the fact that external atoms are used to formulate integrity constraints in the PB-problems and the conflicting strategic companies benchmark, and output values are only derived in the other benchmarks.

7.2 Investigating the Effect of Partial Evaluation in the Main Search

First, we used the three different benchmark problems to investigate the effect of partial evaluations during the main search using different heuristics. In addition, we compared the results to the runtimes achieved by employing our new algorithms for nogood minimization.

7.2.1 HYPOTHESES

We started our investigation with the following hypotheses regarding the employment of partial evaluation in the main search:

(H1) The heuristics **periodic** and **always** decrease the runtime over **never** if useful information is obtainable by early evaluation with little runtime overhead, and increase it otherwise, whereby the effect is stronger for **always**.

(H2) The heuristics **periodic** performs better than **always** if more runtime needs to be invested for each external call, mitigating the tradeoff between information gain and runtime invested in additional calls.

$$trueAt(X) \vee falseAt(X) \leftarrow atom(X).$$
$$\leftarrow \&pbCheck[trueAt, PBInst]().$$

Figure 2: Pseudo-Boolean Problems Rules.

(H3) The tradeoff between information gain and runtime invested in additional calls can be mitigated even more effectively by just minimizing io-nogoods on complete assignments using **ngm** or **ngm-c** instead of evaluating early.

(H4) Using **qxp** or **qxp-c** instead of **ngm** or **ngm-c** decreases the runtime when io-nogoods contain many irrelevant literals, but does not increase it significantly otherwise.

7.2.2 PSEUDO-BOOLEAN PROBLEMS

Pseudo-boolean (*PB-*)problems constitute sets of *pseudo-boolean constraints* of the form $C_0p_0 + ... + C_{n-1}p_{n-1} \geq C_n$, where all p_i are literals and all C_i are integers (Roussel & Manquinho, 2009). A solution to a PB-problem P is a truth assignment to the boolean variables occurring in P such that all inequalities in P are satisfied, where a true literal is interpreted as the value 1 and a false literal as the value 0. Several dedicated PB-solvers have been developed (cf. Manquinho & Silva, 2005), and CLASP can also be employed for efficient PB-problem solving.

Here, however, our goal is not to implement a reasoner for solving PB-problems that can compete with tailored solvers, but to specify external constraints of a HEX-program in the form of PB-problems such that answer sets are restricted to those assignments that also represent solutions to the respective PB-problem. This strict separation of the guess and the check part results in benchmark instances that are well-suited for investigating the effect of a tighter integration of the solving algorithm and the evaluation of external constraints.[9] Moreover, applying an analogous pattern for outsourcing constraints in HEX-programs is a common strategy to avoid the explicit generation of all forbidden combinations of atoms during grounding (Eiter, Redl, & Schüller, 2016).

In our benchmark implementation, we search for solutions to a PB-problem P by guessing an interpretation of the atoms occurring in P utilizing a disjunctive rule, and we restrict the answer sets of the program to solutions of P by employing the external atom $\&pbCheck[trueAt, PBInst]()$ in a program constraint. This results in a simple encoding shown in Figure 2, where a fact $atom(a)$ is added for each atom a occurring in P. At this, the variable $PBInst$ is instantiated by a string containing the path to a file encoding the instance P, and the true extension of the predicate $trueAt$ wrt. an assignment $\mathbf{A}$ represents those atoms occurring in P that are mapped to true by $\mathbf{A}$. The external atom $\&pbCheck[trueAt, PBInst]()$ evaluates to true wrt. a complete assignment $\mathbf{A}$ iff the interpretation of the atoms occurring in P represented by $\mathbf{A}$ constitutes a solution for P. We extend the semantics of the associated evaluation function to partial assignments $\mathbf{A}$ as follows:

$$f_{\&pbCheck'}(\mathbf{A}, trueAt, PBInst) = \begin{cases} \mathbf{T} & \text{if every PB-constraint } C_0p_0 + ... + C_{n-1}p_{n-1} \geq C_n \\ & \text{in } P \text{ fulfills } \sum_{\{c \mid \mathbf{T}trueAt(c) \in \mathbf{A}\} \models p_i} C_i \geq C_n; \\ \mathbf{F} & \text{if some PB-constraint } C_0p_0 + ... + C_{n-1}p_{n-1} \geq C_n \\ & \text{in } P \text{ fulfills } \sum_{\{c \mid \mathbf{F}trueAt(c) \notin \mathbf{A}\} \models p_i} C_i < C_n; \\ \mathbf{U} & \text{otherwise.} \end{cases}$$

9. Note that for the purpose of solving PB-problems as part of a HEX-program (possibly in combination with other external sources), the external source could directly interface a dedicated PB solver.

#	All Answer Sets							solutions
	never	**periodic**	**always**	**ngm**	**ngm-c**	**qxp**	**qxp-c**	
4	**0.13** (0)	**0.13** (0)	0.14 (0)	0.14 (0)	**0.13** (0)	0.14 (0)	0.14 (0)	2.06
8	0.34 (0)	0.33 (0)	**0.22** (0)	0.24 (0)	**0.22** (0)	0.26 (0)	0.23 (0)	4.82
12	4.82 (0)	3.95 (0)	0.80 (0)	0.59 (0)	0.50 (0)	0.60 (0)	**0.46** (0)	10.96
16	280.02 (1)	71.99 (0)	3.28 (0)	1.29 (0)	1.11 (0)	1.23 (0)	**0.94** (0)	12.66
20	300.00 (50)	300.00 (50)	18.87 (0)	2.63 (0)	2.16 (0)	2.41 (0)	**1.62** (0)	24.56
24	300.00 (50)	300.00 (50)	76.75 (1)	4.28 (0)	3.58 (0)	3.69 (0)	**2.42** (0)	32.00
28	300.00 (50)	300.00 (50)	247.06 (32)	9.92 (0)	7.25 (0)	9.48 (0)	**4.61** (0)	82.28
32	300.00 (50)	300.00 (50)	294.05 (47)	20.49 (0)	11.18 (0)	22.03 (1)	**6.94** (0)	269.24
36	300.00 (50)	300.00 (50)	300.00 (50)	36.44 (1)	17.31 (0)	39.27 (3)	**10.28** (0)	519.20
38	300.00 (50)	300.00 (50)	298.99 (49)	38.66 (1)	19.48 (0)	40.86 (2)	**10.90** (0)	451.78
40	300.00 (50)	300.00 (50)	300.00 (50)	37.13 (0)	23.89 (0)	36.04 (0)	**12.70** (0)	233.50

Table 1: Results for random PB-problems with 4 to 40 variables (all answer sets).

#	First Answer Set						
	never	**periodic**	**always**	**ngm**	**ngm-c**	**qxp**	**qxp-c**
4	**0.12** (0)	0.13 (0)	0.13 (0)	0.13 (0)	0.13 (0)	0.13 (0)	0.13 (0)
8	0.23 (0)	0.22 (0)	**0.16** (0)	0.18 (0)	0.18 (0)	0.18 (0)	0.18 (0)
12	2.23 (0)	1.82 (0)	0.35 (0)	0.34 (0)	0.33 (0)	0.31 (0)	**0.30** (0)
16	123.32 (0)	38.42 (0)	1.32 (0)	0.83 (0)	0.81 (0)	0.70 (0)	**0.67** (0)
20	259.72 (42)	237.47 (32)	7.92 (0)	1.59 (0)	1.58 (0)	1.22 (0)	**1.17** (0)
24	294.30 (49)	286.30 (46)	31.13 (0)	2.90 (0)	2.84 (0)	1.96 (0)	**1.89** (0)
28	300.00 (50)	300.00 (50)	96.86 (7)	5.30 (0)	5.26 (0)	3.32 (0)	**3.20** (0)
32	300.00 (50)	300.00 (50)	179.38 (21)	8.05 (0)	8.00 (0)	4.71 (0)	**4.60** (0)
36	300.00 (50)	300.00 (50)	272.92 (42)	12.29 (0)	12.30 (0)	6.65 (0)	**6.58** (0)
38	300.00 (50)	300.00 (50)	264.80 (40)	13.66 (0)	13.76 (0)	7.23 (0)	**7.10** (0)
40	300.00 (50)	300.00 (50)	289.35 (46)	17.26 (0)	17.11 (0)	8.74 (0)	**8.68** (0)

Table 2: Results for random PB-problems with 4 to 40 variables (first answer set).

By exploiting this three-valued semantics, inconsistent partial assignments to the atoms occurring in P can be detected earlier. As a result, potentially large parts of the search space can be pruned and the inconsistent partial assignments can be learned in form of io-nogoods to avoid revisiting the same partial assignments subsequently.

First, we tested randomly generated problems with $N \in [4, 40]$ variables and $4 \times N$ PB-constraints with $n = 6$ and $C_i \in [1, 5]$ for $0 \leq i \leq n$. The results are shown in Tables 1 and 2.

The specific ratio between the number of variables and the number of constraints ensures that only a small fraction of all assignments are answer sets. A clear improvement over **never** is observed whenever external atoms are evaluated early. The configuration **always** shows the best performance, with **periodic** falling in-between **always** and **never**; hence learning the io-behavior of the external source as early as possible outweighs the runtime overhead for querying it additionally. When minimizing io-nogoods only after a complete assignment has been generated in condition **ngm**, the overhead of many external calls can be reduced, while similar information can be obtained from them, resulting in much lower runtimes. Nogood minimization is even more effective when only conflicting nogoods are minimized in condition **ngm-c**. The reason is that in this benchmark, the external atom is only used in a program constraint such that it must evaluate to false wrt. any answer set of the program. Accordingly, the truth value of the external atom is never guessed to be true and non-conflicting io-nogoods, i.e. those which imply a false evaluation of the external atom, cannot prune the search space. The conditions **qxp** and **qxp-c** perform better than **ngm** and **ngm-c**, respectively, which is explained by the fact that in this benchmark io-nogoods typically

#	All Answer Sets							solutions
	never	**periodic**	**always**	**ngm**	**ngm-c**	**qxp**	**qxp-c**	
2	51.59 (0)	23.31 (0)	**0.13** (0)	0.14 (0)	0.14 (0)	**0.13** (0)	**0.13** (0)	0.00
4	61.01 (0)	29.80 (0)	0.42 (0)	0.32 (0)	0.31 (0)	**0.24** (0)	**0.24** (0)	0.04
6	70.69 (0)	40.36 (0)	9.87 (0)	5.32 (0)	2.21 (0)	7.20 (0)	**2.18** (0)	286.58
8	75.15 (0)	58.03 (0)	66.40 (0)	72.73 (0)	**14.78** (0)	98.86 (0)	15.04 (0)	6178.00
10	78.48 (0)	78.48 (0)	150.24 (0)	191.57 (0)	**43.80** (0)	242.41 (0)	44.13 (0)	18297.76
12	87.54 (0)	98.84 (0)	222.71 (0)	258.52 (1)	**72.83** (0)	282.77 (16)	73.49 (0)	26785.20
14	95.24 (0)	111.57 (0)	267.78 (0)	275.09 (3)	**90.38** (0)	269.99 (7)	91.07 (0)	30629.80
16	103.85 (0)	123.68 (0)	299.38 (37)	281.36 (6)	**103.35** (0)	245.74 (2)	103.38 (0)	32141.44
18	**113.89** (0)	135.02 (0)	300.00 (50)	285.74 (5)	114.60 (0)	221.97 (0)	114.31 (0)	32538.18
20	**122.84** (0)	146.10 (0)	300.00 (50)	294.55 (12)	123.51 (0)	205.68 (0)	123.65 (0)	32685.16

Table 3: Results for random PB-problems with PB-constraint length of 2 to 20 (all answer sets).

#	First Answer Set						
	never	**periodic**	**always**	**ngm**	**ngm-c**	**qxp**	**qxp-c**
2	51.91 (0)	23.21 (0)	**0.12** (0)	0.14 (0)	0.14 (0)	0.13 (0)	0.13 (0)
4	60.33 (0)	29.36 (0)	0.42 (0)	0.32 (0)	0.31 (0)	**0.24** (0)	**0.24** (0)
6	2.48 (0)	1.87 (0)	**0.31** (0)	0.50 (0)	0.48 (0)	0.48 (0)	0.45 (0)
8	0.20 (0)	**0.18** (0)	**0.18** (0)	0.25 (0)	0.22 (0)	0.28 (0)	0.23 (0)
10	**0.14** (0)	**0.14** (0)	0.16 (0)	0.19 (0)	0.16 (0)	0.22 (0)	0.17 (0)
12	**0.13** (0)	**0.13** (0)	0.17 (0)	0.18 (0)	0.14 (0)	0.20 (0)	0.14 (0)
14	0.13 (0)	0.13 (0)	0.18 (0)	0.17 (0)	0.13 (0)	0.19 (0)	**0.12** (0)
16	**0.13** (0)	**0.13** (0)	0.19 (0)	0.18 (0)	**0.13** (0)	0.19 (0)	**0.13** (0)
18	**0.13** (0)	**0.13** (0)	0.19 (0)	0.18 (0)	**0.13** (0)	0.19 (0)	**0.13** (0)
20	**0.12** (0)	0.14 (0)	0.20 (0)	0.18 (0)	0.13 (0)	0.19 (0)	0.13 (0)

Table 4: Results for random PB-problems with PB-constraint length of 2 to 20 (first answer set).

contain many irrelevant literals. Overall, **qxp-c** shows the best performance wrt. all instance sizes. Regarding computing the first answer set we observe a similar pattern.

Second, to investigate the behavior when large parts of the search space contain solutions, i.e. when there is less room for pruning it, we fixed the number of variables and PB-constraints to 15 and 60, respectively, and tested different lengths $N \in [2, 20]$. The results are shown in Tables 3 and 4.

The solution count increases with length, and for $N > 14$ nearly all assignments are answer sets. As expected, **periodic** and **always** are slower than **never** if many (more than about half of) the candidates are solutions. Frequent evaluation is detrimental here, as runtime investment has no pay-off in information gain or early search termination. Likewise, minimizing all io-nogoods in conditions **ngm** and **qxp** performs worse than **never** as identical nogoods are computed for many complete assignments. However, the configuration **ngm-c** is very efficient and finds valuable io-nogoods without investing much runtime because it focuses on valuable (i.e. conflicting) io-nogoods. Hence, the overhead of **ngm-c** compared to **never** is also small for instance sizes 18 and 20, where hardly useful information for pruning the search space is available. In contrast to Table 1, **qxp-c** performs slightly worse than **ngm-c** because conflicting io-nogoods now contain mostly relevant literals. As the search space contains a large number of solutions for instances with $N > 6$, the first answer set is always found very fast for such instances.

7.2.3 TAXI ASSIGNMENT WITH ONTOLOGY ACCESS

To facilitate query access for logic programs to external *description logics knowledge bases (DL-KBs)* was one of the early motivating applications of the HEX-formalism, which has been syn-

$$drives(X,Y) \leftarrow driver(X), customer(Y), \&DL[n,n,n,n,isIn](X,A), \quad \text{(r1)}$$
$$\&DL[n,n,n,n,isIn](Y,A), region(A), \text{not } ndrives(X,Y).$$
$$ndrives(X,Y) \leftarrow driver(X), customer(Y), \text{not } drives(X,Y). \quad \text{(r2)}$$
$$driven(Y) \leftarrow drives(\_,Y). \quad \text{(r3)}$$
$$\leftarrow \text{not } driven(Y), customer(Y). \quad \text{(r4)}$$
$$\leftarrow drives(X,Y), drives(X1,Y), X \neq X1. \quad \text{(r5)}$$
$$r^+(\text{“}drivesECust\text{”},X,Y) \leftarrow drives(X,Y), \&DL[n,n,r^+,n,ECust](Y). \quad \text{(r6)}$$
$$\leftarrow \#count\{Y : drives(X,Y)\} > 4, driver(X). \quad \text{(r7)}$$
$$\leftarrow drives(X,Y), \text{not } \&DL[n,n,r^+,n,ECust](Y), \&DL[n,n,r^+,n,EDrv](X). \quad \text{(r8)}$$
$$\leftarrow drives(X,Y), \&DL[n,n,r^+,n,ECust](Y), \text{not } \&DL[n,n,r^+,n,EDrv](X). \quad \text{(r9)}$$

Figure 3: Taxi Assignment Rules.

tactically framed by so-called *DL-programs* (Eiter, Ianni, Lukasiewicz, Schindlauer, & Tompits, 2008). Common reasoning tasks wrt. DL-ontologies are concept and role retrieval, i.e. deriving all individuals respectively pairs of individuals that are instances of a given concept respectively role relationship. For integrating concept and role queries into ASP, *DL-programs* provide so-called *DL-atoms*, which can be represented by external atoms of the form $\&DL[c^+, c^-, r^+, r^-, q](\vec{X})$. Here the inputs c^+ and c^- are binary predicates that declare positive and negative assertions of ontology concept instances, respectively. More specifically an atom $c^+(\text{“}C\text{”}, a)$ (resp. $c^-(\text{“}C\text{”}, a)$) encodes that $C(a)$ (resp. $\neg C(a)$) should be asserted in the DL-KB. Similarly, r^+ and r^- are ternary predicates where $r^+(\text{“}R\text{”}, a, b)$ (resp. $r^-(\text{“}R\text{”}, a, b)$) encodes that $R(a, b)$ (resp. $\neg R(a, b)$) should be asserted in the DL-KB. Evaluating the DL-atom retrieves all instances of the query q, which is either a concept or a role name, relative to the modified ontology. In this way, a bidirectional interaction between the rules of a logic program and the DL-KB is enabled. Accordingly, DL-programs constitute a special type of HEX-programs; using the *DL-Lite* plug-in for DLVHEX (Eiter et al., 2014b), one can evaluate DL-programs with a DL-KB formulated in the DL-Lite language.

For our experiments, we employ the DL-program shown in Figure 3, which assigns taxi drivers to customers under constraints. Our encoding is similar to the one by Eiter, Fink, and Stepanova (2016), but guesses assignments of drivers to customers such that different combinations are possible, whereby non-permissible ones can possibly be detected early by partial evaluation. An external DL-KB formulated in DL-Lite holds part of the information, e.g. about locations of individuals, about e-customers (customers demanding electric cars), and about e-drivers (drivers of electric cars). At this, exactly one driver is assigned to each customer by the rules (r1)-(r5), where the respective driver must be located in the same region as the customer. The latter condition is enforced by using information regarding the regions in which drivers and customers are located that is imported via the DL-atom $\&DL[n, n, n, n, isIn](X, A)$ from the external DL-KB. Customers may share the driver, where a taxi fits at most four customers according to rule (r7). Based on information about which customers are e-customers and which drivers are e-drivers, which is imported via the DL-atoms $\&DL[n, n, r^+, n, ECust](Y)$ and $\&DL[n, n, r^+, n, EDrv](X)$, e-customers must be assigned to e-drivers and normal customers to normal drivers according to rules (r8) and (r9), respectively. Mo-

#	All Answer Sets								solutions
	never	**periodic**	**always**	**ngm-sq**	**ngm**	**ngm-c**	**qxp**	**qxp-c**	
4	0.20 (0)	0.18 (0)	0.22 (0)	0.19 (0)	0.18 (0)	**0.17** (0)	0.18 (0)	**0.17** (0)	7.88
6	0.33 (0)	0.26 (0)	0.32 (0)	0.29 (0)	0.26 (0)	**0.22** (0)	0.26 (0)	**0.22** (0)	17.44
8	1.13 (0)	0.41 (0)	0.60 (0)	0.57 (0)	0.47 (0)	**0.33** (0)	0.41 (0)	**0.33** (0)	36.08
10	7.61 (0)	0.89 (0)	1.42 (0)	1.60 (0)	1.15 (0)	0.66 (0)	0.88 (0)	**0.65** (0)	93.76
12	228.44 (18)	2.19 (0)	3.98 (0)	5.50 (0)	2.98 (0)	**1.49** (0)	2.65 (0)	1.75 (0)	329.92
14	300.00 (50)	9.25 (0)	12.71 (0)	24.52 (1)	15.44 (1)	**5.15** (0)	6.78 (0)	6.45 (0)	651.52
16	300.00 (50)	15.34 (1)	24.22 (1)	55.81 (2)	33.73 (1)	**12.31** (1)	16.89 (1)	13.27 (1)	≥964.68
18	300.00 (50)	67.38 (5)	79.43 (4)	131.03 (12)	87.58 (9)	**47.30** (3)	51.88 (4)	58.43 (3)	≥2767.34
20	300.00 (50)	79.94 (6)	108.65 (7)	186.26 (21)	139.49 (14)	**50.26** (3)	76.36 (5)	67.77 (6)	≥3783.20
22	300.00 (50)	**146.88** (15)	201.91 (23)	265.82 (42)	209.16 (27)	160.66 (17)	167.53 (17)	178.43 (18)	≥5665.76
24	300.00 (50)	**194.62** (25)	243.07 (32)	286.70 (46)	249.06 (34)	216.56 (28)	212.44 (25)	209.65 (26)	≥5840.56
26	300.00 (50)	265.54 (41)	284.26 (45)	294.01 (49)	290.69 (47)	**261.73** (39)	275.54 (43)	265.89 (40)	≥5743.16
28	300.00 (50)	248.42 (39)	253.66 (42)	258.08 (43)	254.46 (42)	**243.08** (39)	252.00 (41)	247.76 (40)	≥5057.28
30	300.00 (50)	293.90 (48)	294.02 (49)	294.01 (49)	294.01 (49)	**292.78** (48)	294.02 (49)	294.01 (49)	≥5322.62

Table 5: Results for taxi assignment with ontology access (all answer sets).

reover, drivers of e-customers are positively asserted for the concept *drivesECust* by rule (r6), which affects subsequent inferences in the DL-KB.

The answer sets of the program with the rules in Figure 3 and further facts *driver(d)*, *customer(c)* and *region(r)* for drivers d, customers c and regions r encode legal assignments. For a complete assignment $\mathbf{A}$, a ground DL-atom $\&DL[c^+, c^-, r^+, r^-, q](\vec{c})$ evaluates as follows:

$$f_{\&DL}(\mathbf{A}, c^+, c^-, r^+, r^-, q) = \begin{cases} \mathbf{T} & \text{if } q(\vec{c}) \text{ is derivable from } KB \cup Assrt(\mathbf{A}), \\ \mathbf{F} & \text{otherwise,} \end{cases}$$

where $Assrt(\mathbf{A})$ consists of all assertions $c(i)$ such that $\mathbf{T}c^+(\text{“}C\text{”}, i) \in \mathbf{A}$, all assertions $\neg c(i)$ such that $\mathbf{T}c^-(\text{“}C\text{”}, i) \in \mathbf{A}$, all assertions $r(i_1, i_2)$ such that $\mathbf{T}r^+(\text{“}r\text{”}, i_1, i_2) \in \mathbf{A}$ and all assertions $\neg r(i_1, i_2)$ such that $\mathbf{T}r^-(\text{“}r\text{”}, i_1, i_2) \in \mathbf{A}$. Exploiting monotonicity of DLs, the evaluation of the associated three-valued oracle function is as follows:

$$f_{\&DL'}(\mathbf{A}, c^+, c^-, r^+, r^-, q) = \begin{cases} \mathbf{T} & \text{if } q(\vec{c}) \text{ is derivable from } KB \cup Assrt(\mathbf{A}), \\ \mathbf{F} & \text{if } q(\vec{c}) \text{ is not derivable from } KB \cup Assrt(\mathbf{A}_{\max}), \\ \mathbf{U} & \text{otherwise,} \end{cases}$$

where $\mathbf{A}_{\max} \supseteq \mathbf{A}$ is the (unique) assignment leading to the largest addition set of assertions.

For instance, the atom $\&DL[n, n, r^+, n, EDrv](d)$ is true wrt. partial assignment $\mathbf{A}$ if $KB \cup \{r(i_1, i_2) \mid \mathbf{T}r^+(\text{“}r\text{”}, i_1, i_2) \in \mathbf{A}\} \models EDrv(d)$; it is false if $KB \cup \{r(i_1, i_2) \mid \mathbf{F}r^+(\text{“}r\text{”}, i_1, i_2) \notin \mathbf{A}\} \not\models EDrv(d)$; and it is unassigned otherwise. The input parameters n in Figure 3 are dummies that, as they do not occur in rule heads or in facts added, have empty extent in every answer set.

In our tests, we increased the number N of drivers and customers gradually from 4 to 30, which were put in $N/2$ regions randomly, where the drivers were balanced among regions. Furthermore, half of the customers were e-customers. The results are shown in Tables 5 and 6.

As DL-atoms have output constants, simultaneous minimization (**ngm**) and sequential minimization (**ngm-sq**) yield different results in this benchmark and we tested both configurations. All configurations that exploit partial evaluations are significantly faster than **never**. The configuration **periodic** now shows better results than **always** because the external DL calls are costly, and waiting a bit until issuing the next one can pay off. Since the premise of an io-nogood can be large but the output often depends only on a small part, minimization can drastically shrink io-nogoods.

#	First Answer Set							
	never	**periodic**	**always**	**ngm-sq**	**ngm**	**ngm-c**	**qxp**	**qxp-c**
4	**0.15** (0)	**0.15** (0)	0.16 (0)	**0.15** (0)	0.16 (0)	**0.15** (0)	0.16 (0)	**0.15** (0)
6	0.18 (0)	**0.16** (0)	0.18 (0)	0.18 (0)	0.18 (0)	0.17 (0)	0.18 (0)	0.18 (0)
8	0.36 (0)	**0.17** (0)	0.20 (0)	0.22 (0)	0.21 (0)	0.22 (0)	0.21 (0)	0.22 (0)
10	1.24 (0)	**0.18** (0)	0.23 (0)	0.25 (0)	0.25 (0)	0.33 (0)	0.25 (0)	0.32 (0)
12	23.56 (0)	**0.22** (0)	0.27 (0)	0.33 (0)	0.33 (0)	0.44 (0)	0.30 (0)	0.43 (0)
14	139.70 (16)	**0.25** (0)	0.32 (0)	0.41 (0)	0.44 (0)	1.10 (0)	0.36 (0)	1.08 (0)
16	273.78 (40)	**0.29** (0)	0.37 (0)	0.49 (0)	0.63 (0)	7.39 (1)	0.42 (0)	7.50 (1)
18	300.00 (50)	**0.40** (0)	0.42 (0)	0.59 (0)	0.73 (0)	23.39 (2)	0.50 (0)	26.53 (3)
20	300.00 (50)	**0.34** (0)	0.46 (0)	0.66 (0)	2.00 (0)	2.12 (0)	0.56 (0)	9.62 (1)
22	300.00 (50)	**0.43** (0)	0.69 (0)	0.64 (0)	0.53 (0)	61.87 (4)	0.67 (0)	53.99 (3)
24	300.00 (50)	**0.46** (0)	0.76 (0)	0.72 (0)	0.60 (0)	113.78 (12)	0.77 (0)	88.03 (9)
26	300.00 (50)	**0.52** (0)	0.86 (0)	0.85 (0)	0.68 (0)	59.02 (6)	0.85 (0)	84.77 (10)
28	300.00 (50)	**0.56** (0)	1.00 (0)	0.90 (0)	0.74 (0)	76.59 (5)	0.88 (0)	95.74 (14)
30	300.00 (50)	**0.63** (0)	1.10 (0)	1.04 (0)	0.83 (0)	112.02 (11)	1.05 (0)	103.03 (11)

Table 6: Results for taxi assignment with ontology access (first answer set).

However, this comes at the price of many external calls due to the large size of the io-nogoods, such that **ngm-sq** is slower than **periodic** and **always**. The costs of minimization can be reduced by minimizing nogoods with the same premise simultaneously, or applying binary search in form of the QUICKXPLAIN algorithm. Accordingly, both **ngm** and **qxp** perform better than **ngm-sq**. Moreover, we observe that **qxp** is slightly faster than **ngm**, even though io-nogoods with identical input parts are not minimized simultaneously by the QUICKXPLAIN algorithm. As for the previous benchmark, minimizing only conflicting nogoods in conditions **ngm-c** and **qxp-c** yields the best results.

Notably, by employing partial evaluations, the first solution can be found rapidly and much faster than in condition **never**, except for the configurations **ngm-c** and **qxp-c**. In contrast to the PB-problems benchmark, here the use of external atoms is not limited to constraints such that minimal nogoods obtained from non-conflicting io-nogoods may contain valuable information. As a result, the missing information leads to timeouts for certain instances even before the first answer set is found, while for other instances the set of all answer sets can be computed faster by **ngm-c** and **qxp-c** than by other configurations.

7.2.4 CONFLICTING STRATEGIC COMPANIES

Strategic Companies is a business problem that is a popular benchmark for ASP competitions, located at the second level of the polynomial hierarchy (Cadoli et al., 1997; Leone et al., 2006). The scenario is that a set $C = \{c_1, ..., c_m\}$ of companies and a set $G = \{g_1, ..., g_n\}$ of goods are given, where each company $c_i \in C$ produces some goods $G_i \subseteq G$ and is possibly controlled by a consortium of owner companies $O_i \subseteq C$. A set of companies $C' \subseteq C$ constitutes a *strategic set* if (1) the companies in C' produce all the goods in G, (2) if $O_i \subseteq C'$ for some $1 \leq i \leq m$, then c_i is in C' as well, and (3) C' is subset-minimal wrt. conditions (1) and (2) (Leone et al., 2006). The knowledge about which companies belong to a strategic set can be crucial for a holding owning the companies in C, e.g. if it has to sell some of its companies and does not want to suffer a loss in economic power. The problem can be encoded concisely in ASP by exploiting the minimality of answer sets, so that each answer set corresponds to one strategic set.

In our benchmark setting, we assume that each product is produced and each company is controlled by at most four companies in C. We further assume an additional conflict relation $R \subseteq C \times C$,

#	All Answer Sets							solutions
	never	**periodic**	**always**	**ngm**	**ngm-c**	**qxp**	**qxp-c**	
5	0.15 (0)	**0.14** (0)	**0.14** (0)	**0.14** (0)	0.15 (0)	**0.14** (0)	**0.14** (0)	1.72
10	**0.13** (0)	**0.13** (0)	0.14 (0)	**0.13** (0)	**0.13** (0)	**0.13** (0)	**0.13** (0)	3.22
15	0.20 (0)	0.19 (0)	0.21 (0)	0.16 (0)	0.16 (0)	**0.15** (0)	**0.15** (0)	8.08
20	0.65 (0)	0.49 (0)	0.51 (0)	0.25 (0)	0.21 (0)	**0.20** (0)	**0.20** (0)	23.42
25	3.49 (0)	1.60 (0)	1.35 (0)	0.43 (0)	0.31 (0)	0.29 (0)	**0.27** (0)	53.50
30	26.79 (0)	7.61 (0)	5.30 (0)	1.09 (0)	0.55 (0)	0.50 (0)	**0.44** (0)	105.32
35	193.38 (0)	33.34 (0)	17.46 (0)	5.72 (0)	1.05 (0)	1.07 (0)	**0.87** (0)	282.26
40	300.00 (50)	135.66 (0)	46.70 (0)	45.95 (3)	1.82 (0)	2.03 (0)	**1.53** (0)	507.08
45	300.00 (50)	297.18 (49)	139.51 (5)	131.44 (15)	5.46 (0)	13.53 (1)	**4.97** (0)	1794.60
50	300.00 (50)	300.00 (50)	267.27 (33)	227.98 (31)	10.25 (0)	27.46 (1)	**9.59** (0)	3453.50
55	300.00 (50)	300.00 (50)	295.25 (46)	262.83 (42)	35.64 (0)	108.24 (14)	**35.15** (0)	5419.98
60	300.00 (50)	300.00 (50)	300.00 (50)	297.71 (49)	56.68 (1)	147.00 (16)	**55.26** (1)	≥6300.94
65	300.00 (50)	300.00 (50)	300.00 (50)	295.89 (49)	151.06 (11)	242.18 (35)	**150.19** (11)	≥7531.40
70	300.00 (50)	300.00 (50)	300.00 (50)	293.64 (48)	194.48 (21)	267.46 (41)	**192.18** (22)	≥7005.06

Table 7: Results for conflicting strategic companies (all answer sets).

$$s(C1) \vee s(C1) \vee s(C3) \vee s(C4) \leftarrow \mathit{producedBy}(\_, C1, C2, C3, C4).$$
$$s(C) \leftarrow \mathit{controlledBy}(C, C1, C2, C3, C4), s(C1), s(C2), s(C3), s(C4).$$
$$\leftarrow \&\mathit{stratConflict}[s]().$$

Figure 4: Conflicting Strategic Companies Rules.

such that companies which are related by R cannot occur together in a strategic set. This constraint makes sense when certain companies may not be kept simultaneously, e.g. due to legislation. The program in Figure 4 encodes the strategic sets that satisfy the conflict relation in its answer sets. In this program, we check the conflict constraint on strategic sets via the external atom $\&\mathit{strategicConflict}[\mathit{strategic}]()$, where $\mathit{strategic}$ contains all companies in the strategic set; on complete assignments, it evaluates to true if some companies c_i, c_j in $\mathit{strategic}$ are in conflict, i.e., $(c_i, c_j) \in R$ holds (where R is externally stored).

Since finding a strategic set is computationally hard, excluding candidate strategic sets with a conflict early in the search by partial evaluations should noticeably decrease the runtime. We use for such evaluations a three-valued oracle function $f_{\&\mathit{strategicConflict}'}(\mathbf{A}, s)$, defined as follows:

$$f_{\&\mathit{strategicConflict}'}(\mathbf{A}, s) = \begin{cases} \mathbf{T} & \text{if } \mathbf{T}s(c_i), \mathbf{T}s(c_j) \in \mathbf{A} \text{ holds for some } (c_i, c_j) \in R, \\ \mathbf{F} & \text{if } \mathbf{F}s(c_i) \in \mathbf{A} \text{ or } \mathbf{F}s(c_j) \in \mathbf{A} \text{ for every } (c_i, c_j) \in R, \\ \mathbf{U} & \text{otherwise.} \end{cases}$$

We ran tests on instances with $N \in [5, 70]$ companies, at most N randomly assigned control relations, $5 \times N$ products with randomly assigned producers, and $N/2$ randomly created conflicts. The results are shown in Tables 7 and 8.

The external conflict constraint cuts more than 90% of the strategic sets (i.e., solution candidates). Thus, like in the first PB-problems benchmark, only a small part of the search space contains solutions. Accordingly, we observe a similar pattern as in Tables 1 and 2, where partial evaluation significantly decreases the runtime in all conditions. The configuration **qxp-c** again exhibits the best results. Since strategic sets are minimal, io-nogoods learned on complete assignments do not provide any valuable information, such that we did not observe a difference when the learning function Λ_u is used instead of Λ_{mu} in this case. Notably, for computing strategic sets containing a specific company (which is $\mathbf{\Sigma_2^P}$-hard in general) we obtain similar results.

#	First Answer Set						
	never	**periodic**	**always**	**ngm**	**ngm-c**	**qxp**	**qxp-c**
5	**0.14** (0)	**0.14** (0)	0.15 (0)	**0.14** (0)	**0.14** (0)	0.15 (0)	0.15 (0)
10	**0.12** (0)	**0.12** (0)	0.13 (0)	**0.12** (0)	**0.12** (0)	**0.12** (0)	**0.12** (0)
15	0.14 (0)	0.14 (0)	0.14 (0)	0.14 (0)	0.14 (0)	0.14 (0)	**0.13** (0)
20	0.24 (0)	0.20 (0)	0.17 (0)	0.16 (0)	0.16 (0)	**0.15** (0)	**0.15** (0)
25	0.76 (0)	0.36 (0)	0.21 (0)	0.20 (0)	0.20 (0)	**0.18** (0)	**0.18** (0)
30	4.60 (0)	1.03 (0)	0.41 (0)	0.29 (0)	0.29 (0)	**0.23** (0)	**0.23** (0)
35	26.05 (0)	2.71 (0)	0.54 (0)	0.39 (0)	0.37 (0)	**0.27** (0)	**0.27** (0)
40	118.70 (12)	10.67 (0)	1.47 (0)	0.54 (0)	0.54 (0)	0.34 (0)	**0.33** (0)
45	158.24 (22)	21.35 (0)	1.38 (0)	0.56 (0)	0.55 (0)	**0.34** (0)	**0.34** (0)
50	230.00 (32)	43.29 (3)	6.19 (0)	0.76 (0)	0.75 (0)	**0.40** (0)	**0.40** (0)
55	287.99 (47)	131.69 (15)	5.67 (0)	1.05 (0)	1.04 (0)	**0.51** (0)	**0.51** (0)
60	291.70 (47)	187.90 (26)	10.10 (0)	1.47 (0)	1.45 (0)	0.64 (0)	**0.63** (0)
65	294.49 (49)	229.99 (35)	15.06 (0)	1.85 (0)	1.86 (0)	**0.76** (0)	0.77 (0)
70	300.00 (50)	244.27 (39)	52.26 (1)	2.32 (0)	2.26 (0)	0.85 (0)	**0.83** (0)

Table 8: Results for conflicting strategic companies (first answer set).

Regarding the results for finding the first answer set, the larger difference between **never** and **always** in comparison to Table 2 is due to the higher computational effort required for finding compatible sets, where learning based on partial assignments is able to guide the search towards a compatible set which is also an answer set.

7.2.5 FINDINGS REGARDING PARTIAL EVALUATION IN THE MAIN SEARCH

In our experiments, we found that early evaluation in conditions **always** and **periodic** increased the performance for all benchmarks, except for the case where nearly all candidate solutions correspond to answer sets such that no useful information is obtainable from additional oracle calls. This finding is in line with hypothesis (H1). Moreover, in the case of the taxi assignment benchmark, where external calls require more runtime than in the other benchmark implementations, **periodic** performed better than **always** due to less runtime overhead, which supports our hypothesis (H2). Similar improvements could be achieved by minimizing all io-nogoods that are learned based on complete assignments with configuration **ngm**. However, regarding hypothesis (H3), the results are mixed because minimization performs worse when io-nogoods are large or contain many relevant literals (cf. Tables 3 and 5). This overhead is avoided by only minimizing nogoods that directly trigger backjumping in condition **ngm-c**. The configuration **ngm-c** performs well for all benchmark problems and has only little overhead when no useful information is available, as can be observed in Table 3. Finally, the fact that **qxp** performed better than **ngm** in the taxi assignment benchmark, where io-nogoods typically contain many irrelevant literals, and did not increase the runtime much when nearly all literals are relevant (as it is the case for the second experiment using PB-problems) provides supporting evidence for hypothesis (H4). This effect results from the number of external calls that have to be performed in the best resp. the worst case by the QUICKXPLAIN algorithm compared to sequential minimization.

Overall, minimization of conflicting nogoods by configuration **ngm-c** or **qxp-c** in most cases yielded the best results with only small differences between the two conditions. Hence they are suggested as the default configurations.

#	All Answer Sets					
	never	**always**	**ufs-p**	**ufs-a**	**always + ufs-a**	**qxp-c**
4	0.19 (0)	0.22 (0)	0.20 (0)	0.48 (0)	0.53 (0)	**0.17** (0)
6	0.35 (0)	0.35 (0)	0.38 (0)	1.64 (0)	1.78 (0)	**0.22** (0)
8	1.27 (0)	0.70 (0)	1.02 (0)	8.70 (0)	8.31 (0)	**0.33** (0)
10	7.86 (0)	1.41 (0)	3.67 (0)	33.22 (0)	30.28 (0)	**0.65** (0)
12	228.11 (17)	4.16 (0)	37.50 (1)	139.01 (10)	93.41 (3)	**1.75** (0)
14	300.00 (50)	12.25 (0)	186.79 (20)	281.62 (42)	190.64 (21)	**6.45** (0)
16	300.00 (50)	24.42 (1)	292.14 (47)	300.00 (50)	273.96 (40)	**13.27** (1)
18	300.00 (50)	81.46 (4)	300.00 (50)	300.00 (50)	295.59 (49)	**58.43** (3)
20	300.00 (50)	110.71 (8)	300.00 (50)	300.00 (50)	300.00 (50)	**67.77** (6)
22	300.00 (50)	203.06 (24)	300.00 (50)	300.00 (50)	288.02 (48)	**178.43** (18)
24	300.00 (50)	243.39 (32)	300.00 (50)	300.00 (50)	294.02 (49)	**209.65** (26)
26	300.00 (50)	284.93 (45)	300.00 (50)	300.00 (50)	294.01 (49)	**265.89** (40)
28	300.00 (50)	253.71 (42)	300.00 (50)	300.00 (50)	258.15 (43)	**247.76** (40)
30	300.00 (50)	294.02 (49)	300.00 (50)	300.00 (50)	294.02 (49)	**294.01** (49)

Table 9: Results for taxi assignment with ontology access wrt. partial evaluation during unfounded set search (all answer sets).

7.3 Investigating the Effect of Partial Evaluation in the Unfounded Set Search

In our benchmark programs, the one for the taxi assignment problem contains cyclic dependencies through external atoms, while this is not the case for the pseudo-boolean and conflicting strategic companies programs. The absence of such cyclic dependencies means that compatible sets of a HEX-program already correspond to its answer sets and a minimality check can be skipped (Eiter et al., 2014a). Consequently, partial evaluation during the search for unfounded sets did not have an impact on the performance results for the pseudo-boolean and conflicting strategic companies benchmarks.

For this reason, in addition to the taxi assignment benchmark, we have considered two variants of the conflicting strategic companies problem in order to test partial evaluation during the unfounded set check with different heuristics.

7.3.1 HYPOTHESES

Our hypotheses concerning the use of partial evaluation in the unfounded set search were the following:

(H5) The heuristics **ufs-a** and **ufs-p** decrease the runtime over **never** if useful information is obtainable by early evaluation during the unfounded set search with little runtime overhead, and increase it otherwise, whereby the effect is stronger for **ufs-a**.

(H6) The heuristics **ufs-p** performs better than **ufs-a** if external calls need more time or less information can be gained from them, mitigating the tradeoff between information gain and runtime invested in additional calls.

(H7) If the heuristics **ufs-a** or **ufs-p** are combined with the heuristics **always**, there is a further speedup in case many io-nogoods learned during the unfounded set search are not already learned during the main search. Thus, the combination is expected to be more effective when for learning the function Λ_u is used instead of Λ_{mu}.

#	First Answer Set					
	never	**always**	**ufs-p**	**ufs-a**	**always** + **ufs-a**	**qxp-c**
4	0.16 (0)	0.17 (0)	0.16 (0)	0.20 (0)	0.21 (0)	**0.15** (0)
6	0.20 (0)	0.19 (0)	0.20 (0)	0.28 (0)	0.28 (0)	**0.18** (0)
8	0.39 (0)	**0.22** (0)	0.37 (0)	0.55 (0)	0.38 (0)	**0.22** (0)
10	1.27 (0)	**0.22** (0)	1.16 (0)	1.65 (0)	0.46 (0)	0.32 (0)
12	22.60 (0)	**0.27** (0)	21.88 (0)	25.55 (1)	0.59 (0)	0.43 (0)
14	137.84 (15)	**0.32** (0)	134.38 (14)	139.96 (15)	0.79 (0)	1.08 (0)
16	273.70 (40)	**0.37** (0)	267.38 (39)	270.47 (40)	1.04 (0)	7.50 (1)
18	300.00 (50)	**0.43** (0)	300.00 (50)	300.00 (50)	1.32 (0)	26.53 (3)
20	300.00 (50)	**0.47** (0)	300.00 (50)	300.00 (50)	1.61 (0)	9.62 (1)
22	300.00 (50)	**0.69** (0)	300.00 (50)	300.00 (50)	2.92 (0)	53.99 (3)
24	300.00 (50)	**0.78** (0)	300.00 (50)	300.00 (50)	3.78 (0)	88.03 (9)
26	300.00 (50)	**0.89** (0)	300.00 (50)	300.00 (50)	4.56 (0)	84.77 (10)
28	300.00 (50)	**1.04** (0)	300.00 (50)	300.00 (50)	5.05 (0)	95.74 (14)
30	300.00 (50)	**1.29** (0)	300.00 (50)	300.00 (50)	8.12 (0)	103.03 (11)

Table 10: Results for taxi assignment with ontology access wrt. partial evaluation during unfounded set search (first answer set).

$$s(C1) \vee s(C1) \vee s(C3) \vee s(C4) \leftarrow \mathit{producedBy}(\_, C1, C2, C3, C4).$$
$$s(C) \leftarrow \&\mathit{majority}[s](C), \mathit{company}(C).$$

Figure 5: Strategic Companies with External Controls Relation Rules.

7.3.2 TAXI ASSIGNMENT

For testing the effect of partial evaluation during the unfounded set search wrt. the taxi assignment benchmark, we used the same set of problem instances as before (cf. Table 5). The results are shown in Tables 9 and 10, where we also report the running times of the conditions **always** and **qxp-c** for comparison with partial evaluation during the main search.

For this benchmark, the compatible sets are identical to the answer sets, such that no unfounded sets are detected during unfounded set search. However, due to cyclic dependencies through external atoms, minimality wrt. the FLP-reduct still needs to be verified for each instance by means of the unfounded set check. We observe that configurations **ufs-p** and **ufs-a** slightly improve the efficiency over **never**, where **ufs-p** yields better results since calls to the external oracle are costly in this benchmark. However, performing early evaluations during the main search in condition **always** is much faster resulting in less timeouts; and combining early evaluation in the main and the unfounded set search does not result in an additional speedup. This is expected: as reasoning in a DL ontology is monotonic and for DL-Lite ontologies the io-nogoods are small, the information that is obtained by early evaluation in the respective parts is largely overlapping.

7.3.3 STRATEGIC COMPANIES WITH EXTERNAL CONTROLS RELATION

We considered a second variant of the strategic companies problem, where the controls relation is derived by means of an external atom of the form $\&\mathit{majority}[\mathit{strategic}](c)$, based on the company shares that other companies own (cf. Figure 5). A company is then controlled by a suite of other companies if their combined shares exceed 50 %. No conflict relations are added in this benchmark as they only remove compatible sets and do not have a direct influence on the minimality check,

#	All Answer Sets						solutions / compatible sets
	never	**always**	**ufs-p**	**ufs-a**	**always + ufs-a**	**qxp-c**	
2	**0.13** (0)	0.14 (0)	**0.13** (0)	0.14 (0)	0.14 (0)	0.14 (0)	1.18 / 1.68
4	**0.15** (0)	0.17 (0)	0.16 (0)	0.19 (0)	0.21 (0)	0.17 (0)	1.70 / 2.96
6	**0.19** (0)	0.25 (0)	0.21 (0)	0.31 (0)	0.36 (0)	0.26 (0)	2.56 / 4.46
8	**0.29** (0)	0.41 (0)	0.33 (0)	0.58 (0)	0.71 (0)	0.50 (0)	4.20 / 6.42
10	**0.63** (0)	0.81 (0)	0.76 (0)	1.27 (0)	1.49 (0)	0.95 (0)	7.44 / 10.10
12	1.80 (0)	**1.56** (0)	1.91 (0)	2.86 (0)	2.95 (0)	1.89 (0)	9.30 / 13.62
14	4.97 (0)	**3.14** (0)	5.09 (0)	6.96 (0)	6.31 (0)	3.49 (0)	18.08 / 24.46
16	15.88 (0)	**5.98** (0)	15.03 (0)	16.97 (0)	12.81 (0)	6.65 (0)	26.96 / 35.04
18	59.10 (0)	**13.52** (0)	50.48 (0)	47.16 (0)	26.65 (0)	13.86 (0)	36.52 / 47.56
20	169.77 (1)	**30.46** (0)	132.65 (0)	124.17 (0)	55.99 (0)	31.44 (0)	64.30 / 79.58
22	297.81 (46)	**59.32** (0)	285.57 (40)	275.11 (32)	108.75 (0)	61.88 (0)	97.02 / 116.22
24	300.00 (50)	128.94 (0)	300.00 (50)	300.00 (50)	210.98 (11)	**127.59** (4)	≥154.66 / 195.84
26	300.00 (50)	255.72 (20)	300.00 (50)	300.00 (50)	269.39 (36)	**235.22** (17)	≥242.22 / 366.26
28	300.00 (50)	292.88 (43)	300.00 (50)	300.00 (50)	295.10 (46)	**279.55** (41)	≥121.64 / 402.66
30	300.00 (50)	300.00 (50)	300.00 (50)	300.00 (50)	300.00 (50)	**298.91** (49)	≥102.06 / 548.94

Table 11: Results for strategic companies with external controls relation (all answer sets).

while the aim of this experiment is to investigate the effect of partial evaluation on the unfounded set search.

Let $shares(c_1, c_2)$ denote the fraction of shares of company c_2 that company c_1 owns. Given a partial assignment $\mathbf{A}$, a company c, and a predicate $strategic$ representing a set of companies, the corresponding three-valued oracle function $f_{\&majority}(\mathbf{A}, strategic, c)$ is defined as follows:

$$f_{\&majority}(\mathbf{A}, strategic, c) = \begin{cases} \mathbf{T} & \text{if } \sum_{\mathbf{T}strategic(c_i) \in \mathbf{A}} shares(c_i, c) > 50\,\%; \\ \mathbf{F} & \text{if } \sum_{\mathbf{T}strategic(c_i), \mathbf{U}strategic(c_i) \in \mathbf{A}} shares(c_i, c) \leq 50\,\%; \\ \mathbf{U} & \text{otherwise.} \end{cases}$$

As the oracle function behaves monotonically, we can employ the learning function Λ_{mu}.

For testing, we randomly generated instances with $N \in [2, 30]$ companies, randomly distributed 50 % to 100 % of the shares of each company over 1 to 4 other companies, and added $5 \times N$ products with randomly assigned producers. The results are shown in Tables 11 and 12, again with the running times of **always** and **qxp-c**.

In contrast to the taxi assignment benchmark, where all compatible sets were answer sets, now around 20 % of the solution candidates are eliminated by the unfounded set check. While **always** again significantly increases the performance, there is no clear winner among the conditions **never**, **ufs-p** and **ufs-a**. For instance sizes smaller than 16, configuration **never** is faster than **ufs-a**, with **ufs-p** falling in between. However, for instances with more than 16 companies, this pattern is inverted and **ufs-a** exhibits a slightly better performance than the other two configurations. The reason is that for larger instances, the unfounded set search requires a larger fraction of the overall solving time. Thus, triggering backjumping earlier has a higher impact on the overall running time of the unfounded set search wrt. larger instances. Overall, the effect of employing partial evaluations only in the unfounded set search is small since monotonicity of the external source already allows to learn small io-nogoods that are exploited by the unfounded set search. The fact that conditions **always** and **qxp-c** are still very efficient indicates that nogoods learned in the main search help to speed up the unfounded set search as well, but not the other way around. This is also supported by the observation that exploiting early evaluations based on partial assignments both in the main and

#	First Answer Set					
	never	**always**	**ufs-p**	**ufs-a**	**always + ufs-a**	**qxp-c**
2	**0.13** (0)	0.14 (0)	**0.13** (0)	0.14 (0)	0.14 (0)	0.14 (0)
4	**0.14** (0)	0.16 (0)	0.15 (0)	0.16 (0)	0.17 (0)	0.15 (0)
6	**0.16** (0)	0.18 (0)	**0.16** (0)	0.20 (0)	0.21 (0)	0.21 (0)
8	**0.19** (0)	0.22 (0)	0.20 (0)	0.26 (0)	0.29 (0)	0.32 (0)
10	**0.25** (0)	0.29 (0)	0.29 (0)	0.39 (0)	0.43 (0)	0.55 (0)
12	0.51 (0)	**0.46** (0)	0.55 (0)	0.71 (0)	0.67 (0)	1.01 (0)
14	0.72 (0)	**0.55** (0)	0.77 (0)	0.98 (0)	0.79 (0)	1.45 (0)
16	2.03 (0)	**0.85** (0)	2.46 (0)	2.13 (0)	1.25 (0)	2.67 (0)
18	5.01 (0)	**2.00** (0)	7.49 (0)	5.06 (0)	2.05 (0)	5.87 (0)
20	14.15 (0)	**2.31** (0)	15.46 (0)	12.55 (0)	2.41 (0)	11.64 (0)
22	54.07 (2)	5.10 (0)	54.66 (3)	49.00 (3)	**4.19** (0)	21.82 (0)
24	73.45 (3)	10.84 (0)	99.72 (4)	64.73 (2)	**4.92** (0)	48.33 (1)
26	144.56 (12)	23.70 (1)	155.93 (15)	124.87 (10)	**7.13** (0)	73.96 (7)
28	189.26 (21)	34.27 (2)	190.98 (25)	169.98 (22)	**9.29** (0)	153.97 (20)
30	222.31 (31)	86.92 (7)	230.87 (35)	190.72 (27)	**11.35** (0)	172.17 (22)

Table 12: Results for strategic companies with external controls relation (first answer set).

in the unfounded set search in condition **always + ufs-a** decreases the performance compared to **always**.

Notably, configuration **always + ufs-a** significantly outperforms all other conditions for computing the first answer set. This is explained by the fact that, in this case, different io-nogoods are learned in the main and the unfounded set search, respectively, while the overlap increases when more answer sets are computed. Accordingly, nogoods learned in each of the two search procedures are more likely to complement each other, resulting in lower running times.

7.3.4 STRATEGIC COMPANIES WITH NONMONOTONIC EXTERNAL CONTROLS RELATION

We considered a second variant of the strategic companies problem with an external controls relation to test the effect of early external evaluation during the unfounded set check when the external source behaves nonmonotonically. In this case, the general learning function Λ_u has to be used instead of Λ_{mu}. As a result, io-nogoods are less general because they also contain the negative input part (cf. Definition 10) and thus, nogoods learned in the main search are less likely to be also useful in the unfounded set search.

Here, the same problem instances as in the previous monotonic case are used, but the semantics of the oracle function associated with the external atom $\&majority[strategic](c)$ is modified as follows:

$$f_{\&majority'}(\mathbf{A}, strategic, c) = \begin{cases} \mathbf{T} & \text{if } \sum_{\mathbf{T}strategic(c_i)\in\mathbf{A}} shares(c_i, c) > 50\,\%, \text{ and} \\ & \sum_{\mathbf{T}strategic(c_i),\mathbf{U}strategic(c_i)\in\mathbf{A}} shares(c_i, c) < 100\,\%; \\ \mathbf{F} & \text{if } \sum_{\mathbf{T}strategic(c_i),\mathbf{U}strategic(c_i)\in\mathbf{A}} shares(c_i, c) \leq 50\,\%, \text{ or} \\ & \sum_{\mathbf{T}strategic(c_i)\in\mathbf{A}} shares(c_i, c) = 100\,\%; \\ \mathbf{U} & \text{otherwise.} \end{cases}$$

Accordingly, a company is only added to a candidate strategic set via the external atom if its shares owned by other companies in the set exceed 50 %, but are less than 100 %. This is motivated by the fact that selling the full shares, i.e., the entire company, might result in a higher payoff than selling only bits; hence a holding might be inclined to not keep a company that it owns fully. The corresponding results are shown in Tables 13 and 14.

#	All Answer Sets						solutions / compatible sets
	never	**always**	**ufs-p**	**ufs-a**	**always + ufs-a**	**qxp-c**	
2	**0.13** (0)	**0.13** (0)	**0.13** (0)	0.14 (0)	0.14 (0)	**0.13** (0)	1.18 / 1.68
4	**0.14** (0)	0.17 (0)	0.15 (0)	0.18 (0)	0.20 (0)	0.17 (0)	1.70 / 2.96
6	**0.19** (0)	0.24 (0)	0.21 (0)	0.30 (0)	0.34 (0)	0.28 (0)	2.56 / 4.46
8	**0.41** (0)	0.43 (0)	0.44 (0)	0.63 (0)	0.66 (0)	0.56 (0)	4.20 / 6.42
10	1.48 (0)	**1.03** (0)	1.65 (0)	1.91 (0)	1.66 (0)	1.31 (0)	7.44 / 10.10
12	5.48 (0)	**2.31** (0)	5.65 (0)	5.68 (0)	3.33 (0)	3.14 (0)	9.30 / 13.62
14	21.73 (0)	**5.42** (0)	21.90 (0)	18.82 (0)	8.16 (0)	8.85 (0)	18.08 / 24.46
16	82.33 (0)	**10.30** (0)	85.19 (1)	62.76 (0)	16.58 (0)	16.81 (0)	26.96 / 35.04
18	295.92 (38)	**27.77** (0)	276.44 (25)	223.77 (5)	31.65 (0)	51.90 (2)	36.52 / 47.56
20	300.00 (50)	**66.34** (1)	300.00 (50)	300.00 (50)	66.86 (0)	104.89 (7)	64.30 / 79.58
22	300.00 (50)	**128.22** (6)	300.00 (50)	300.00 (50)	130.16 (1)	141.36 (13)	≥96.84 / 116.22
24	300.00 (50)	**216.67** (15)	300.00 (50)	300.00 (50)	237.73 (15)	234.66 (27)	≥143.32 / 195.74
26	300.00 (50)	291.04 (43)	300.00 (50)	300.00 (50)	**277.07** (39)	278.78 (38)	≥131.10 / 311.74
28	300.00 (50)	295.25 (47)	300.00 (50)	300.00 (50)	295.78 (46)	**286.65** (45)	≥120.28 / 280.04
30	300.00 (50)	300.00 (50)	300.00 (50)	300.00 (50)	**299.88** (49)	300.00 (50)	≥108.38 / 267.14

Table 13: Results for strategic companies with nonmonotonic external controls relation (all answer sets).

We do not observe a difference regarding the number of solutions compared to the previous benchmark as 100 % of controlled shares are usually not reached wrt. the used instances. Nevertheless, the external source needs to ensure that this limit cannot be reached, before returning the truth value $\mathbf{T}$ for a particular company in the output. Consequently, the learning function Λ_u has to be utilized such that nogoods are typically larger than in the previous benchmark. Due to the altered semantics of the external source, running times in general increase, while the overall pattern remains similar to the one encountered for the previous benchmark. However, we observe that the running times for configuration **never** increase to a higher degree relative to the other conditions. This indicates that exploiting external evaluations wrt. partial assignments has an additional benefit when the external source behaves nonmonotonically. Now, for instances containing more than 12 companies we always observe an advantage of **ufs-a** and **ufs-p** over **never**, whereby **ufs-a** is faster than **ufs-p**.

Again, condition **always + ufs-a** proved to be very efficient for computing only the first answer set. Even though the running times for computing one solution in all other conditions significantly increase compared to the previous benchmark setting, the running times for configuration **always + ufs-a** are similar to before. This is because, on the one hand, less information about the external source semantics is available to the solver from the preceding search before the first answer set has been computed. On the other hand, io-nogoods learned in conditions **always** and **ufs-a**, respectively, are now less likely to be useful for the main search and the unfounded set search simultaneously, due to nonmonotonicity of the external source. In contrast, configuration **always + ufs-a** enables the learning of io-nogoods tailored to each of the two search procedures.

7.3.5 FINDINGS REGARDING PARTIAL EVALUATION IN THE UNFOUNDED SET SEARCH

We found that configurations **ufs-p** and **ufs-a** improve the performance in all benchmarks considered for testing partial evaluation in the unfounded set search. However, the improvement was not as pronounced as the one we found for partial evaluation in the main search, and depends on how much room there is for decreasing the runtime of the unfounded set search by detecting conflicts earlier. Thus, hypothesis (H5) is partly supported by our experimental results. In the experiments employing strategic companies problems, where external calls are inexpensive, condition **ufs-a** showed lower

#	First Answer Set					
	never	**always**	**ufs-p**	**ufs-a**	**always + ufs-a**	**qxp-c**
2	0.13 (0)	0.13 (0)	**0.12** (0)	0.13 (0)	0.14 (0)	0.13 (0)
4	**0.14** (0)	0.15 (0)	**0.14** (0)	0.15 (0)	0.16 (0)	0.15 (0)
6	**0.15** (0)	0.17 (0)	0.16 (0)	0.19 (0)	0.20 (0)	0.21 (0)
8	**0.22** (0)	**0.22** (0)	0.23 (0)	0.27 (0)	0.27 (0)	0.33 (0)
10	0.46 (0)	**0.35** (0)	0.49 (0)	0.55 (0)	0.43 (0)	0.70 (0)
12	1.51 (0)	**0.63** (0)	1.57 (0)	1.53 (0)	0.73 (0)	1.62 (0)
14	3.79 (0)	1.11 (0)	3.77 (0)	3.49 (0)	**0.98** (0)	4.11 (0)
16	13.33 (0)	2.91 (0)	20.98 (1)	10.42 (0)	**1.51** (0)	10.99 (0)
18	47.45 (0)	10.23 (0)	74.04 (5)	38.08 (0)	**2.49** (0)	41.02 (2)
20	147.18 (8)	27.06 (0)	154.22 (12)	129.44 (6)	**3.18** (0)	79.82 (6)
22	241.06 (31)	58.19 (5)	242.13 (32)	237.55 (29)	**5.53** (0)	104.79 (12)
24	280.97 (45)	90.34 (7)	280.88 (44)	277.49 (42)	**6.14** (0)	179.49 (23)
26	294.46 (48)	125.64 (16)	294.04 (48)	294.23 (48)	**8.90** (0)	153.72 (21)
28	300.00 (50)	154.06 (22)	300.00 (50)	300.00 (50)	**10.48** (0)	210.73 (33)
30	300.00 (50)	194.76 (29)	300.00 (50)	300.00 (50)	**12.76** (0)	198.39 (31)

Table 14: Results for strategic companies with nonmonotonic external controls relation (first answer set).

running times than **ufs-p**, while for the taxi benchmark with more costly external evaluations it was the other way around. This is in line with hypotheses (H5) and (H6). In support of hypothesis (H6), **ufs-p** also performs better than **ufs-a** for small instances of the strategic companies benchmarks, where the information obtained from external calls is less useful due to less time required by the minimality check and low usefulness of learned nogoods wrt. the main search.

Finally, we found that for computing all answer sets the combination **always + ufs-a** does not increase the efficiency compared to only utilizing partial evaluation in the main search. However, the combination is very efficient when only the first answer set is computed, where the sets of io-nogoods learned in the main and the unfounded set search, respectively, are less likely to overlap. Moreover, we observed that the combination has an even higher advantage over other conditions when a nonmonotonic external source is accessed as this as well increases the chance of learning different nogoods in the main search and the unfounded set search, respectively. Accordingly, our experiments confirm hypothesis (H7).

We conclude that even when partial evaluation during the unfounded set search does not increase the efficiency for computing all answer sets compared to other configurations, it can be highly effective in combination with partial evaluation during the main search in case only one solution is required.

8. Discussion and Conclusion

In this section, we first continue and extend our review of related work, and we then conclude with a summary and outlook on ongoing and future issues.

8.1 Related Work

As mentioned in the introduction, our work is most closely related to SMT solving, in particular to theory propagation there (Nieuwenhuis, Oliveras, & Tinelli, 2006), and naturally to constraint ASP solving, as developed by Gebser, Ostrowski, and Schaub (2009), and theory solving in CLINGO 5 (Gebser et al., 2016).

As for the relation to SMT solving, we observe that the latter typically considers fixed types of theories, while HEX is more general and geared towards supporting heterogeneous theories. Using a fixed type of theory is a characteristics of several extensions of ASP with SMT, such as DINGO (Janhunen, Liu, & Niemelä, 2013), which uses difference logic, NLP-DL (Eiter, Ianni, Schindlauer, & Tompits, 2005a), which uses description logics, and ASPMT (Lee & Meng, 2013).

Moreover, the abstract level of semantics in terms of input-output relations accommodates even arbitrary non-logical theories. However, there is closer similarity regarding integration schemas and learning techniques. Typical integration schemas for SMT have been identified (Balduccini & Lierler, 2013b), which apply to ASP modulo theories as well (a comparison is given by Balduccini & Lierler, 2013a):

- In *black-box* integration, the SAT solver blindly generates a model and passes it for checking to the theory solver. If it passes the check, the model is returned, otherwise it is added in constraint form to the instance and the solver restarts. This allows for easy coupling with arbitrary theories but does not enable search space pruning.

- In *grey-box* integration, the theory solver is only called for complete models of the SAT instance, but the SAT solver is merely suspended during checking and can continue its search afterwards; integration is still relatively simple.

- Only in *clear-box* integration, the SAT solver is interleaved with the theory solver, which is called already for partial assignments and in turn may propagate further truth values or detect inconsistencies. However, the integration is much more challenging as the theory solver must identify atoms implied by the given partial assignment, or by inconsistency reasons, respectively.

Examining HEX, the grey-box schema corresponds to the evaluation algorithms in use before external behavior learning was introduced by Eiter et al. (2012); black-box integration, i.e. resorting to complete restarts, has never been used for HEX solving. With incorporation of such learning, the algorithms fit an intermediate schema between grey- and clear-box integration: external sources were still only evaluated under complete assignments, but the learned nogoods possibly pruned the search space.

Compared to constraint ASP solving, the HEX formalism is more general as it supports access to arbitrary external sources which are largely black boxes, and without (implicit) assumptions of their properties. In this respect, constraint ASP can be considered as a special case of HEX with theory-specific knowledge. There are a number of integrations of ASP with constraint programming, as realized e.g. in CLINGCON (Ostrowski & Schaub, 2012), LC2CASP (Cabalar, Kaminski, Ostrowski, & Schaub, 2016), EZCSP (Balduccini, 2009), and EZSMT (Susman & Lierler, 2016); we refer to the work of Lierler, Maratea, and Ricca (2016) for an overview of systems. Here, we focus on the work of Ostrowski and Schaub (2012), who considered nogood minimization as we do, but used different algorithms that avoid expensive resets of the constraint solver. However, this is only possible by exploiting properties of the specific theory at hand (monotonic constraint satisfaction), which in our more general setting do not always apply (e.g., for nonmonotonic external atoms); furthermore, the user-friendly plug-and-play integration of external sources does not provide control over the external algorithms. On the other hand, other possibilities for optimizations arise, e.g., simultaneous io-nogood minimization since external atoms can have multiple output values for the same input.

Unlike HEX-programs, theory solving in CLINGO 5 (and constraint ASP as a special instance thereof) does not support external atoms with dedicated input and output. Instead, certain atoms in the logic program are declared as *theory atoms* whose truth values are set via the external theory. In that, CLINGO 5 follows a global perspective rather than the local one of external atoms in HEX-programs, where the scope is the rule body (as customary in logic programming), whereby theory atoms may be shared by different rules. Another more distinguishing difference concerns the actual semantics of programs, and here in particular foundedness of answer sets. Roughly speaking, CLINGO 5 fixes a valuation of the theory atoms and computes then an answer set of the program relative to this valuation; this amounts to using a GL-style reduct where theory atoms are removed from rules. In contrast, for evaluating HEX-programs all external atoms remain in the rules, according to the FLP reduct. While CLINGO 5 makes no further minimality check, for HEX-programs an unfounded set check is launched which may eliminate a candidate answer set. For example, CLINGO 5 would return $\mathbf{A}_2 = \{\mathbf{T}p\}$ as an answer set for the program $\Pi = \{p \leftarrow \&id[p]().\}$ (adapted to the different formalism), which is eliminated by the unfounded set check for HEX-programs. Notably, the need for external evaluation calls in the unfounded set check leaves room for different evaluation heuristics as presented in Section 6.

In ordinary ASP solving, one can distinguish unfounded sets due to positive cycles and unfounded sets due to disjunctions with head cycles. The check for the former is done *a priori* over partial assignments and is feasible in linear time; the check for the latter is coNP-hard and typically done *a posteriori* under complete assignments only. We performed some experiments with UFS checking over partial assignments, i.e., running Algorithm 4 also over partial input assignments, but it soon turned out that the additional cost of more UFS checks exceeded the benefits by far. Also Gebser, Kaufmann, and Schaub (2013) conducted some experiments with unfounded set checking for disjunctive ordinary ASP over partial assignments. However, they also reported only moderate computational benefits, although their UFS check is cheaper than ours due to absence of external calls. Therefore, we did not pursue the idea of UFS checking over partial assignments further.

Related to techniques for nogood minimization, nowadays most SAT-solvers integrate techniques for *learned clause minimization*, which remove redundancies from learned clauses by computationally inexpensive procedures (Sörensson & Biere, 2009). However, the role of io-nogoods learned from external source evaluations in HEX and the role of nogoods learned from conflicts by a respective ASP-solver, even though both serve the purpose of guiding the search procedure, are very different. While conflict nogoods are usually obtained by resolution based on an available implication graph such that minimization techniques as the ones by Sörensson and Biere can be applied, io-nogoods are not learned using an implication graph but from single oracle calls. Due to the black box nature of these oracles, they can only be used to retrieve all correct outputs for external atoms wrt. a given assignment. Hence, techniques for learned clause minimization are not directly applicable for learning smaller io-nogoods. Nevertheless, such techniques can still be exploited in our approach for conflict nogood minimization, depending on whether the respective solver used for performing the main CDNL-search implements them.

Other related work comprises alternative solving techniques, such as the one by Eiter et al. (2014b), where the semantics of external atoms is captured by so-called *support sets*, which are similar to our faithful io-nogoods and related to implicants of logical theories (Darwiche & Marquis, 2002; Reiter & de Kleer, 1987). However, different from our approach, the main idea there is to learn all or sufficiently many support sets at the beginning of the solving process, such that satisfaction and unsatisfaction of an external atom under given input is completely covered. External

atom evaluation can then be accomplished by matching the support sets against the interpretation; this eliminates external calls during solving entirely, but comes at the price of learning up to exponentially many support sets. A related support-set based approach goes a step further and encodes the semantics of external atoms straight into the ASP-program (Redl, 2017). The exponential worst-case blowup suggests to use these approaches only for external atoms with a compact and small representation by support sets. Moreover, since they genuinely depend on exhaustive learning of support sets at the beginning, they cannot directly benefit from the possibility of partial evaluation as presented in the previous sections.

The notion of three-valued oracle functions has also been used to extend HEX with *lazy-grounding* techniques (Eiter, Kaminski, & Weinzierl, 2017), where not only the evaluation of external atoms is postponed, but also the grounding of the HEX-program itself, by employing as backend-solver the lazy-grounding ASP solver ALPHA (Weinzierl, 2017).

Finally, Antic, Eiter, and Fink (2013) considered partial HEX-semantics before, by employing *Approximation Fixpoint Theory* (*AFT*) (Denecker, Marek, & Truszczyński, 2000; Denecker, Marek, & Truszczynski, 2004) that works on intervals in the power set lattice. While our partial oracle functions amount to their three-valued oracle functions, we only consider two-valued answer sets and we do not apply a fixpoint construction to define the answer set semantics. Similarly, Pelov, Denecker, and Bruynooghe (2004) have defined a family of partial stable model semantics for logic programs with aggregates using AFT. Assignment-monotonic oracle functions are also related to their *approximating aggregate relations* which must be *precision-monotone* and generalize ordinary aggregate relations to a three-valued semantics.

8.2 Summary and Outlook

In this article, we have pushed efficient evaluation techniques for ASP with external source access, by introducing three-valued evaluation of external atoms under partial (incomplete) truth value assignments. The techniques we introduced yield a full-fledged clear-box integration. Moreover, due to automatic nogood minimization, developers of external sources do not need to manually describe implied truth values or inconsistency reasons, but only need to implement a three-valued oracle function, which keeps the integration of sources simple.

In our experiments, the new techniques yielded a speedup of up to two orders of magnitude; unsurprisingly, their ranking depends on the instances. This is similar to the observations by Ostrowski and Schaub (2012), who reported mixed results for different propagation delays. Our results are also in line with results in SMT, where theory propagation, if doable with small overhead, is crucial for performance (Dutertre & de Moura, 2006; Lahiri, Nieuwenhuis, & Oliveras, 2006; Nieuwenhuis & Oliveras, 2005). We observed that in most cases learning from complete assignments plus minimization of conflicting nogoods (based on partial assignments) outperforms learning during search; hence, this setting is suggestive as a default. This is explained by the fact that in this case, learning focuses on nogoods that are useful for conflict resolution, thus the information gain is similar and the overhead much smaller. This is in line with the observation by Nieuwenhuis et al. (2006) that conflict analysis uses only a small fraction of the lemmas learned by theory propagation, which can be addressed with *lazy explanations* (Gent, Miguel, & Moore, 2010). The speedup can be exponential, as evidenced by an external atom whose truth value is definite after assigning a single input atom, e.g. $\&empty[p]()$ to check whether an atom over p is true. Each naive nogood eliminates one of exponentially many assignments, but a linear number of minimized ones eliminate all wrong guesses.

There are different directions for ongoing and future work. One topic is to include further heuristics for deciding whether external evaluation is invoked or skipped at some point. This decision might be based, for instance, on the past information gain; other criteria are conceivable. Another topic is further improvement of nogood minimization. To this end, the divide-and-conquer strategy borrowed from Junker's QUICKXPLAIN algorithm (2004) might be replaced by a more sophisticated one, e.g. the one that Shchekotykhin et al. (2015) developed for their MERGEXPLAIN algorithm. By the latter, multiple minimal conflict sets (resp., nogoods) can be found during one program run; this could be integrated into our approach for obtaining multiple minimal io-nogoods.

Acknowledgments

We would like to thank the reviewers for their helpful and constructive comments to improve the presentation of this article. This work has been supported by the Austrian Science Fund (FWF) via the projects P24090 and W1225-N23. Antonius Weinzierl is currently supported by the Academy of Finland, project 251170.

Appendix A. Proofs

Proposition 1. *For every* HEX*-program* Π *and external predicate &g defined by a two-valued oracle function, one can redefine &g by an assignment-monotonic three-valued oracle function without changing the answer sets of* Π.

Proof. For each external predicate $\&g$ we introduce a new external predicate $\&g'$, construct program Π' by replacing all occurrences of $\&g$ in Π by $\&g'$, and define $f_{\&g'}(\mathbf{A},\cdot,\cdot) = f_{\&g}(\mathbf{A},\cdot,\cdot)$ if $\mathbf{A}$ is complete over Π and $f_{\&g'}(\mathbf{A},\cdot,\cdot) = \mathbf{U}$ otherwise. Since under complete assignments all external atoms in Π have the same truth values as the corresponding external atoms in Π', and answer sets are complete assignments by definition, it follows immediately that $\mathcal{AS}(\Pi) = \mathcal{AS}(\Pi')$. □

Theorem 1 (Soundness and Completeness of Algorithm 1)**.** *If Algorithm 1 returns for an input program* Π *(i) an assignment* $\mathbf{A}$*, then* $\mathbf{A}$ *is an answer set of* Π*; (ii) the symbol* $\bot$*, then* Π *is inconsistent.*

Proof. The algorithm extends the conflict-driven algorithm for ordinary ASP as follows:

- The check for compatibility of $\hat{\mathbf{A}}$ and for minimality wrt. $f\Pi^{\mathbf{A}}$ in the if-block of Part (c) is added.
- The evaluation of external atoms and addition of nogoods in Part (e).

Without these changes, soundness and completeness of the algorithm for ordinary ASP (as presented by Drescher et al., 2008) implies that the algorithm returns the projection of some answer set $\hat{\mathbf{A}}$ of $\hat{\Pi}$ to $A(\Pi)$ if $\hat{\Pi}$ is consistent, and $\bot$ otherwise. We now show that the changes adopt the behavior as desired.

First, the added if-block in Part (c) eliminates those answer sets of $\hat{\Pi}$ which are either not compatible sets of Π, or not minimal models wrt. $f\Pi^{\mathbf{A}}$. The remaining answer sets of $\hat{\Pi}$ projected to the atoms in Π are exactly the answer set of Π (cf. Definition 2). Thus, the algorithm with the added if-block in Part (c) but without the addition of Part (e) has exactly the desired behavior.

Second, the addition of Part (e) is only an optimization and we need to justify that it does not eliminate answer sets of Π. But this follows from the correctness of $\Lambda(\cdot,\cdot)$, which implies that assignments forbidden by such nogoods would be incompatible with the external sources anyway; therefore they cannot be compatible sets and also not answer sets. □

Proposition 2. *If N is a faithful io-nogood such that $N_O = \{\sigma_{n+1} e_{\&g[\vec{p}]}(\vec{c})\}$, then N is correct wrt. all programs Π that use $e_{\&g[\vec{p}]}(\vec{c})$.*

Proof. Consider a faithful io-nogood $N = \{\sigma_1 a_1, \ldots, \sigma_n a_n\} \cup \{\sigma_{n+1} e_{\&g[\vec{p}]}(\vec{c})\}$. Then faithfulness implies $f_{\&e}(N_I, \vec{p}, \vec{c}) = \overline{\sigma(N_O)}$. Suppose an assignment $\mathbf{A}$ violates N. Then $\mathbf{A} \supseteq N_I$ and thus $f_{\&e}(\mathbf{A}, \vec{p}, \vec{c}) = \overline{\sigma(N_O)} = \overline{\sigma_{n+1}}$. However, since $\sigma_{n+1} e_{\&g[\vec{p}]}(\vec{c}) \in \mathbf{A}$, it follows that $\mathbf{A}$ cannot be a compatible set of any program. □

Proposition 3. *Let $\&g[\vec{p}](\cdot)$ be an external atom in a* HEX*-program Π. Then for all assignments $\mathbf{A}$, the nogoods $\Lambda_u(\&g[\vec{p}], \mathbf{A})$ in Definition 9 are correct wrt. Π.*

Proof. The added nogood for an output tuple $\vec{c}$ such that $f_{\&g}(\mathbf{A}, \vec{p}, \vec{c}) = \sigma$, $\sigma \in \{\mathbf{T}, \mathbf{F}\}$, is $\{\overline{\sigma e_{\&g[\vec{p}]}(\vec{c})}\} \cup \{\sigma' p(\vec{c}') \in \mathbf{A} \mid p \in \vec{p}, \sigma' \neq \mathbf{U}\}$. If the nogood is violated by an assignment $\mathbf{A}'$, then the guess for $e_{\&g[\vec{p}]}(\vec{c})$ wrt. $\mathbf{A}'$ was wrong as the replacement atom is guessed false (resp. true) but the tuple $\vec{c}$ is in the output (resp. not in the output). Hence, the assignment $\mathbf{A}'$ is not compatible and cannot be a compatible set anyway. □

Proposition 4. *Let $\&g[\vec{p}](\cdot)$ be an external atom in a* HEX*-program Π. Then for all assignments $\mathbf{A}$, the nogoods $\Lambda_{mu}(\&g[\vec{p}], \mathbf{A})$ in Definition 10 are correct wrt. Π.*

Proof. The added nogood for an output tuple $\vec{c}$ such that $f_{\&g}(\mathbf{A}, \vec{p}, \vec{c}) = \sigma$, $\sigma \in \{\mathbf{T}, \mathbf{F}\}$, is $\{\overline{\sigma e_{\&g[\vec{p}]}(\vec{c})}\} \cup \{\sigma' p(\vec{c}') \in \mathbf{A} \mid p \in \vec{p}, p \notin \vec{p_m}, \sigma' \neq \mathbf{U}\} \cup \{\sigma p(\vec{c}') \in \mathbf{A} \mid p \in \vec{p_m}\}$. If the nogood is violated by an assignment $\mathbf{A}'$, then the guess for $e_{\&g[\vec{p}]}(\vec{c})$ wrt. $\mathbf{A}'$ was wrong as the replacement atom is guessed false (resp. true) but the tuple $\vec{c}$ is in the output (resp. not in the output). The previous holds despite the fact that literals of form $\mathbf{T}p(\vec{c})$ (resp. $\mathbf{F}p(\vec{c})$), where $p \in \vec{p_m}$, are omitted from io-nogoods implying a false (resp. true) evaluation of the oracle function because $\{\mathbf{T}p(\vec{c}) \mid \mathbf{T}p(\vec{c}) \in \mathbf{A}'\} \geq \{\mathbf{T}p(\vec{c}) \mid \mathbf{T}p(\vec{c}) \in \mathbf{A}\}$ (resp. $\{\mathbf{F}p(\vec{c}) \mid \mathbf{F}p(\vec{c}) \in \mathbf{A}'\} \supseteq \{\mathbf{F}p(\vec{c}) \mid \mathbf{F}p(\vec{c}) \in \mathbf{A}\}$) must hold for all $p \in \vec{p_m}$, and we have that $f_{\&g}(\mathbf{A}, \vec{p}, \vec{c}) = \mathbf{T}$ implies that $f_{\&g}(\mathbf{A}'', \vec{p}, \vec{c}) = \mathbf{T}$ for every $\mathbf{A}'' \geq \mathbf{A}$ (resp. that $f_{\&g}(\mathbf{A}, \vec{p}, \vec{c}) = \mathbf{F}$ implies that $f_{\&g}(\mathbf{A}'', \vec{p}, \vec{c}) = \mathbf{F}$ for every $\mathbf{A}''$ s.t. $\{\mathbf{F}p(\vec{c}) \mid \mathbf{F}p(\vec{c}) \in \mathbf{A}''\} \supseteq \{\mathbf{F}p(\vec{c}) \mid \mathbf{F}p(\vec{c}) \in \mathbf{A}\}$), due to the definition of monotonic input parameters. Hence, the assignment $\mathbf{A}'$ is not compatible and cannot be a compatible set anyway. □

Proposition 5. *Let $\mathbf{A}$ be a partial assignment and N be a faithful io-nogood for $\&g[\vec{p}]$ over the atoms in $\mathbf{A}$. Then some $N' \in minimize(\Lambda_u(\&g[\vec{p}], \mathbf{A}))$ exists such that $N' \subseteq N$.*

Proof. The nogood N can be reduced to a subset-minimal set M such that M is a faithful io-nogood but $f_{\&g}(N'', \vec{p}, \vec{c}) = \mathbf{U}$ for all N'' with $N''_I \subsetneq M_I$, $N''_O = M_O$. We observe that $M \in minimize(\Lambda_u(\&g[\vec{p}], \mathbf{A}))$ and $M \subseteq N$. □

Proposition 6. *Let Λ_l be an io-complete theory-specific learning function for an external source $\&g$. Then, for all partial assignments $\mathbf{A}$ and input lists $\vec{p}$ we have $minimize(\Lambda_u(\&g[\vec{p}], \mathbf{A})) = min_{\subseteq}(\Lambda_l(\&g[\vec{p}], \mathbf{A}))$.*

Proof. Let $\mathbf{A}$ be a partial assignment and let $\vec{p}$ be an input list.

($\Rightarrow$) Let $N \in \mathit{minimize}(\Lambda_u(\&g[\vec{p}], \mathbf{A}))$ be an io-nogood learned from Λ_u after minimization. Since $f_{\&g}(N_I, \vec{p}, \vec{c}) = \overline{\sigma(N_O)}$ by faithfulness, it follows from completeness of Λ_l that $N \in \Lambda_l(\&g[\vec{p}], \mathbf{A})$. Moreover, since N is minimal, it follows that $f_{\&g}(N'_I, \vec{p}, \vec{c}) = \mathbf{U}$ for all $N' = N'_I \cup N_O$ with $N'_I \subsetneq N_I$. Therefore, there can be no $N' \subsetneq N$ with $N' \in \Lambda_l(\&g[\vec{p}], \mathbf{A})$, thus $N \in \mathit{min}_{\subseteq}(\Lambda_l(\&g[\vec{p}]))$.

($\Leftarrow$) Let $N \in \mathit{min}_{\subseteq}(\Lambda_l(\&g[\vec{p}], \mathbf{A}))$ be a subset-minimal nogood learned from Λ_l. Since $f_{\&g}(N_I, \vec{p}, \vec{c}) = \overline{\sigma(N_O)}$ (due to faithfulness) we have $N \in \Lambda_u(\&g[\vec{p}], \mathbf{A})$ by definition of Λ_u. Moreover, since N is subset-minimal among the nogoods $\Lambda_l(\&g[\vec{p}], \mathbf{A})$ and Λ_l is io-complete, we have that $f_{\&g}(N'_I, \vec{p}, \vec{c}) = \mathbf{U}$ for all $N' = N'_I \cup \{\sigma(N_O)e_{\&g[\vec{p}]}(\vec{c})\}$. But then no atom from N can be removed as by Definition 11, thus $N \in \mathit{minimize}(\Lambda_u(\&g[\vec{p}], \mathbf{A}))$. □

Proposition 7. *For a set S of faithful io-nogoods with equal input parts and distinct output parts, Algorithm 2 yields exactly one faithful io-nogood $N' \in \mathit{minimize}(N)$ for each $N \in S$.*

Proof. Let S be a set of faithful io-nogoods with identical input parts and distinct output parts. To distinguish between the input and the output of Algorithm 2, we denote by S^o the altered set which is returned by the algorithm given input S.

First of all, S is only manipulated in Part (c) by replacing the input part of nogoods in S. Hence, it holds that $|S| = |S^o|$. Moreover, all $N \in S$ have different output parts which are not changed in Part (c). As a result, after every replacement exactly one element in the resulting set S' can be associated with each nogood in the initial set S. This proves that each input has a corresponding output nogood. It remains to show that these are minimal faithful io-nogoods.

Let $N \in S$ and $N^o \in S^o$ the corresponding output nogood, i.e. $N_O = N^o_O = \sigma_{n+1}e_{\&g[\vec{p}]}(\vec{c})$. We have to show that $N^o \in \mathit{minimize}(N)$. According to Definition 11, this means we have to show that $N^o \subseteq N$, that N^o is a faithful io-nogood, and that $f_{\&g}(N'', \vec{p}, \vec{c}) = \mathbf{U}$ for all $N'' \subsetneq N^o_I$. Clearly, it holds that $N^o \subseteq N$ since elements are only *removed* from N_I by Algorithm 2.

Next, we prove that N^o is a faithful io-nogood by showing that faithful io-nogoods in S are only replaced by faithful io-nogoods, in Algorithm 2. Let S' be an arbitrary state of S during the execution of Algorithm 2, and $N' \in S'$ a faithful io-nogood. We need to show that $N^s \cup N'_O$ is also a faithful io-nogood for $N^s = N'_I \setminus \sigma_i a_i$ where $\sigma_i a_i \in N'_I$. Since $N^s \subset N'_I$, it holds that $N^s \cup N'_O$ is an io-nogood. In Part (c), N'_I is replaced by N^s in S only if $\overline{N'_O} \in \mathit{output}$ for $\langle N^s, \mathit{output}\rangle \in \mathit{ch}$, which is the case only if $f_{\&g}(N^s, \vec{p}, \vec{c}) = \overline{\sigma(N'_O)} \neq \mathbf{U}$, due to Part (b) of the algorithm. Note that it is ensured in Part (b) that for N^s there is exactly one $\langle N^s, \mathit{output}\rangle \in \mathit{ch}$. Further, we know that $\overline{\sigma(N'_O)} \neq \mathbf{U}$ as N' is an io-nogood. Due to assignment-monotonicity of $f_{\&g}$, we have that $f_{\&g}(N^s, \vec{p}, \vec{c}) = X$, $X \in \{\mathbf{T}, \mathbf{F}\}$, implies $f_{\&g}(\mathbf{A}, \vec{p}, \vec{c}) = X$ for all partial assignments $\mathbf{A} \succeq N^s$, by Definition 6. We derive that $f_{\&g}(\mathbf{A}, \vec{p}, \vec{c}) = \overline{\sigma(N'_O)}$ for all partial assignments $\mathbf{A} \supseteq N^s$ and thus, that $N^s \cup N'_O$ is in fact a faithful io-nogood. Since we know that N is a faithful io-nogood, we conclude that N^o is a faithful io-nogood as well.

Finally, we prove that $f_{\&g}(N'', \vec{p}, \vec{c}) = \mathbf{U}$ for all $N'' \subsetneq N^o_I$. Assume to the contrary that we have $f_{\&g}(N'', \vec{p}, \vec{c}) \neq \mathbf{U}$ for some $N'' \subsetneq N^o_I$, and let $\sigma_i a_i \in N^o_I \setminus N''$. Since $\sigma_i a_i \in N^o_I$, we have that $N^s = N'_I \setminus \sigma_i a_i$ holds for some N' that is chosen during the execution of Algorithm 2 in Part (a) (i.e. in the iteration when it is tried to obtain smaller nogoods by removing $\sigma_i a_i$), with $N'_O = \sigma_{n+1}e_{\&g[\vec{p}]}(\vec{c})$ and $N'_I \subseteq N^o_I$. As we have that $\sigma_i a_i \in N^o_I$, we derive that $f_{\&g}(N^s, \vec{p}, \vec{c}) = \mathbf{U}$. Otherwise N' would be replaced by $N^s \cup N'_O$ in Part (c), and we would obtain $\sigma_i a_i \notin N^o_I$. However, we obtain that $f_{\&g}(N'', \vec{p}, \vec{c}) \neq \mathbf{U}$, $f_{\&g}(N^s, \vec{p}, \vec{c}) = \mathbf{U}$ and $N^s \supset N''$, which together contradicts

that $f_{\&g}$ is assignment-monotonic. This proves that indeed $f_{\&g}(N'', \vec{p}, \vec{c}) = \mathbf{U}$ for all $N'' \subsetneq N_I^o$ and thus, $N^o \in \mathit{minimize}(N)$. □

Proposition 9. *Let $\mathbf{A}$ be a complete assignment, $\mathbf{X}$ be a partial assignment, and $f_{\&g}$ be an assignment monotonic three-valued oracle function. Then, $f_{\&g}(\mathbf{A} \dot\cup \neg.\mathbf{X}, \vec{p}, \vec{c}) = X$, $X \in \{\mathbf{T}, \mathbf{F}\}$, implies for every assignment $\mathbf{X}' \succeq \mathbf{X}$ that $f_{\&g}(\mathbf{A} \dot\cup \neg.\mathbf{X}', \vec{p}, \vec{c}) = X$.*

Proof. Let $\mathbf{A}$ be a complete assignment, $\mathbf{X}$ a partial assignment, and $f_{\&g}$ an assignment monotonic three-valued oracle function. Further let $f_{\&g}(\mathbf{A} \dot\cup \neg.\mathbf{X}, \vec{p}, \vec{c}) = X$, where $X \in \{\mathbf{T}, \mathbf{F}\}$, and let $\mathbf{X}'$ be an arbitrary partial assignment s.t. $\mathbf{X}' \succeq \mathbf{X}$. We need to show that $f_{\&g}(\mathbf{A} \dot\cup \neg.\mathbf{X}', \vec{p}, \vec{c}) = X$. First, we show that $\mathbf{A} \dot\cup \neg.\mathbf{X}' \succeq \mathbf{A} \dot\cup \neg.\mathbf{X}$. Recall that $\mathbf{A} \dot\cup \neg.\mathbf{X} = (\mathbf{A} \setminus \{\mathbf{T}a \mid \mathbf{T}a \in \mathbf{X} \text{ or } \mathbf{U}a \in \mathbf{X}\}) \cup \{\mathbf{F}a \mid \mathbf{T}a \in \mathbf{X}\} \cup \{\mathbf{U}a \mid \mathbf{U}a \in \mathbf{X} \text{ and } \mathbf{T}a \in \mathbf{A}\}$, according to Definition 13. Since $\mathbf{X}' \succeq \mathbf{X}$, we have that $\{\mathbf{T}a \in \mathbf{X}\} \cup \{\mathbf{F}a \mid \mathbf{F}a \in \mathbf{X}\} \subseteq \mathbf{X}'$, due to the definition of "$\succeq$". Hence, we derive that $\{\mathbf{T}a \mid \mathbf{T}a \in \mathbf{X} \text{ or } \mathbf{U}a \in \mathbf{X}\} \succeq \{\mathbf{T}a \mid \mathbf{T}a \in \mathbf{X}' \text{ or } \mathbf{U}a \in \mathbf{X}'\}$. It follows that $(\mathbf{A} \setminus \{\mathbf{T}a \mid \mathbf{T}a \in \mathbf{X}' \text{ or } \mathbf{U}a \in \mathbf{X}'\}) \succeq (\mathbf{A} \setminus \{\mathbf{T}a \mid \mathbf{T}a \in \mathbf{X} \text{ or } \mathbf{U}a \in \mathbf{X}\})$. It is also easy to see that $\{\mathbf{F}a \mid \mathbf{T}a \in \mathbf{X}'\} \succeq \{\mathbf{F}a \mid \mathbf{T}a \in \mathbf{X}\}$ and $\{\mathbf{U}a \mid \mathbf{U}a \in \mathbf{X}' \text{ and } \mathbf{T}a \in \mathbf{A}\} \succeq \{\mathbf{U}a \mid \mathbf{U}a \in \mathbf{X} \text{ and } \mathbf{T}a \in \mathbf{A}\}$. Consequently, we infer that $\mathbf{A} \dot\cup \neg.\mathbf{X}' \succeq \mathbf{A} \dot\cup \neg.\mathbf{X}$. Because we have that $f_{\&g}(\mathbf{A} \dot\cup \neg.\mathbf{X}, \vec{p}, \vec{c}) = X$, and due to assignment monotonicity according to Definition 6, from $\mathbf{A} \dot\cup \neg.\mathbf{X}' \succeq \mathbf{A} \dot\cup \neg.\mathbf{X}$ it follows that $f_{\&g}(\mathbf{A} \dot\cup \neg.\mathbf{X}', \vec{p}, \vec{c}) = X$. □

Proposition 10. *Let Π be a* HEX*-program and let $\mathbf{A}$ be a complete assignment over $A(\Pi)$. If there is a solution $\mathbf{S}$ for Ω_Π with assumptions $\mathcal{A}_\mathbf{A}$ such that for all external atoms $\&g[\vec{p}](\vec{c})$ in Π it holds that*

(1) $\mathbf{T}e_{\&g[\vec{p}]}(\vec{c}) \in \mathbf{S}$ and $\mathbf{A} \not\models \&g[\vec{p}](\vec{c})$ implies $\mathbf{A} \dot\cup \neg.\mathbf{S} \not\models \&g[\vec{p}](\vec{c})$, and

(2) $\mathbf{F}e_{\&g[\vec{p}]}(\vec{c}) \in \mathbf{S}$ and $\mathbf{A} \models \&g[\vec{p}](\vec{c})$ implies $\mathbf{A} \dot\cup \neg.\mathbf{S} \models \&g[\vec{p}](\vec{c})$,

then $\mathbf{U} = \{Xa \mid a \in A(\Pi), Xa \in \mathbf{S}, X \in \{\mathbf{T}, \mathbf{F}\}\}$ is an unfounded set for Π wrt. $\mathbf{A}$.

Proof. The proof is identical to the proof for Theorem 10 by Eiter et al. (2014a) modulo the different representation of assignments by sets of ground atoms used by Eiter et al. (2014a). □

Proposition 11. *Let Π be a* HEX*-program, let $\mathbf{A}$ be a complete assignment over $A(\Pi)$ and suppose Algorithm 4 is executed with Π and $\mathbf{A}$ as inputs. If there is an unfounded set $\mathbf{U}$ for Π wrt. $\mathbf{A}$ s.t. $\{\mathbf{T}a \mid \mathbf{T}a \in \mathbf{A} \cap \mathbf{U}\} \neq \emptyset$, then there is a solution $\mathbf{S}$ for Ω_Π with assumptions $\mathcal{A}_\mathbf{A}$, s.t. $\{\mathbf{T}a \mid \mathbf{T}a \in \mathbf{A} \cap \mathbf{S}\} \neq \emptyset$, that satisfies conditions (1) and (2) of Proposition 10 and all transformed nogoods $\mathcal{T}_\Omega(N)$ added to Ω'_Π in Part (d) of Algorithm 4.*

Proof. Let Π be a HEX-program, let $\mathbf{A}$ be a complete assignment over $A(\Pi)$ and suppose Algorithm 4 is executed with Π and $\mathbf{A}$ as inputs. Further, let there be an unfounded set $\mathbf{U}$ for Π wrt. $\mathbf{A}$ s.t. $\{\mathbf{T}a \mid \mathbf{T}a \in \mathbf{A} \cap \mathbf{U}\} \neq \emptyset$. According to Proposition 8 by Eiter et al. (2014a), there is a solution $\mathbf{S}$ for Ω_Π with assumptions $\mathcal{A}_\mathbf{A}$ s.t. $\{\mathbf{T}a \mid \mathbf{T}a \in \mathbf{A} \cap \mathbf{S}\} \neq \emptyset$. In addition, it follows directly from Proposition 11 by Eiter et al. (2014a) that $\mathbf{S}$ satisfies conditions (1) and (2) of Proposition 10.

It is easy to see that any faithful io-nogood as defined in Definition 8 is also a *valid input-output-relationship* according to Definition 9 by Eiter et al. (2014a). Moreover, we only consider faithful io-nogoods returned by the learning function Λ. Consequently, we infer that $\mathbf{S}$ also satisfies all transformed nogoods $\mathcal{T}_\Omega(N)$ added to Ω'_Π in Part (d) of Algorithm 4 according to Proposition 15 by Eiter et al. (2014a). □

Theorem 2 (Soundness and Completeness of Algorithm 4)**.** *Given a* HEX*-program* Π *and a complete assignment* $\mathbf{A}$ *over* $A(\Pi)$ *as inputs, Algorithm 4 returns* $true$ *if there is an unfounded set* $\mathbf{U}$ *for* Π *wrt.* $\mathbf{A}$ *s.t.* $\{\mathbf{T}a \mid \mathbf{T}a \in \mathbf{A} \cap \mathbf{U}\} \neq \emptyset$*, and* $false$ *otherwise.*

Proof. Let Π be a HEX-program, let $\mathbf{A}$ be a complete assignment over $A(\Pi)$ and suppose Algorithm 4 is executed with Π and $\mathbf{A}$ as inputs.

We first show that Algorithm 4 returns $true$ if there is an unfounded set $\mathbf{U}$ for Π wrt. $\mathbf{A}$ such that $\{\mathbf{T}a \mid \mathbf{T}a \in \mathbf{A} \cap \mathbf{U}\} \neq \emptyset$. Consider the case that there is an unfounded set $\mathbf{U}$ for Π wrt. $\mathbf{A}$ such that $\{\mathbf{T}a \mid \mathbf{T}a \in \mathbf{A} \cap \mathbf{U}\} \neq \emptyset$. According to Proposition 11, a solution $\mathbf{S}$ for Ω_Π with assumptions $\mathcal{A}_\mathbf{A}$ exists, such that $\{\mathbf{T}a \mid \mathbf{T}a \in \mathbf{A} \cap \mathbf{S}\} \neq \emptyset$, which does not violate any nogood added to Ω'_Π in Part (d) of Algorithm 4. Consequently, the complete assignment $\mathbf{S}$ is generated by Algorithm 4 and Part (c) is executed. Since $\mathbf{S}$ satisfies conditions (1) and (2) of Proposition 10, according to Proposition 11, the variable $isUFS$ is not set to $false$ in Part (c), and because $\{\mathbf{T}a \mid \mathbf{T}a \in \mathbf{A} \cap \mathbf{S}\} \neq \emptyset$ the algorithm returns $true$.

Now we show that Algorithm 4 returns $false$ if there is no unfounded set $\mathbf{U}$ for Π wrt. $\mathbf{A}$ such that $\{\mathbf{T}a \mid \mathbf{T}a \in \mathbf{A} \cap \mathbf{U}\} \neq \emptyset$. Towards contradiction, suppose that there is no unfounded set $\mathbf{U}$ for Π wrt. $\mathbf{A}$ such that $\{\mathbf{T}a \mid \mathbf{T}a \in \mathbf{A} \cap \mathbf{U}\} \neq \emptyset$ and that Algorithm 4 does not return $false$. This means that $false$ is not returned in Part (b) because $true$ is returned before the search space has been completely explored. Accordingly, a complete assignment $\mathbf{S}$ is generated by Algorithm 4 and Part (c) is executed, which satisfies conditions (1) and (2) of Proposition 10. Moreover, it must hold that $\{\mathbf{T}a \mid \mathbf{T}a \in \mathbf{A} \cap \mathbf{S}\} \neq \emptyset$ because otherwise $true$ would not be returned in Part (c). However, from Proposition 10 we know that $\mathbf{U} = \{Xa \,|\, a \in A(\Pi), Xa \in \mathbf{S}, X \in \{\mathbf{T}, \mathbf{F}\}\}$ is an unfounded set for Π wrt. $\mathbf{A}$. Since we have that $\{\mathbf{T}a \mid \mathbf{T}a \in \mathbf{A} \cap \mathbf{S}\} \neq \emptyset$, we have that $\{\mathbf{T}a \mid \mathbf{T}a \in \mathbf{A} \cap \mathbf{U}\} \neq \emptyset$ and hence, we infer that there is an unfounded set $\mathbf{U}$ for Π wrt. $\mathbf{A}$ such that $\{\mathbf{T}a \mid \mathbf{T}a \in \mathbf{A} \cap \mathbf{U}\} \neq \emptyset$. Thus, we derive a contradiction, and infer that Algorithm 4 returns $false$ if there is no unfounded set $\mathbf{U}$ for Π wrt. $\mathbf{A}$ such that $\{\mathbf{T}a \mid \mathbf{T}a \in \mathbf{A} \cap \mathbf{U}\} \neq \emptyset$.

□

Appendix B. Line Chart Representations of Benchmark Results

In this section, we present line chart representations of our empirical results from Section 7 as an alternative visualization. At this, each chart corresponds to one table from Section 7. The instance sizes are shown on the x-axis, and the running times in seconds on the y-axis.

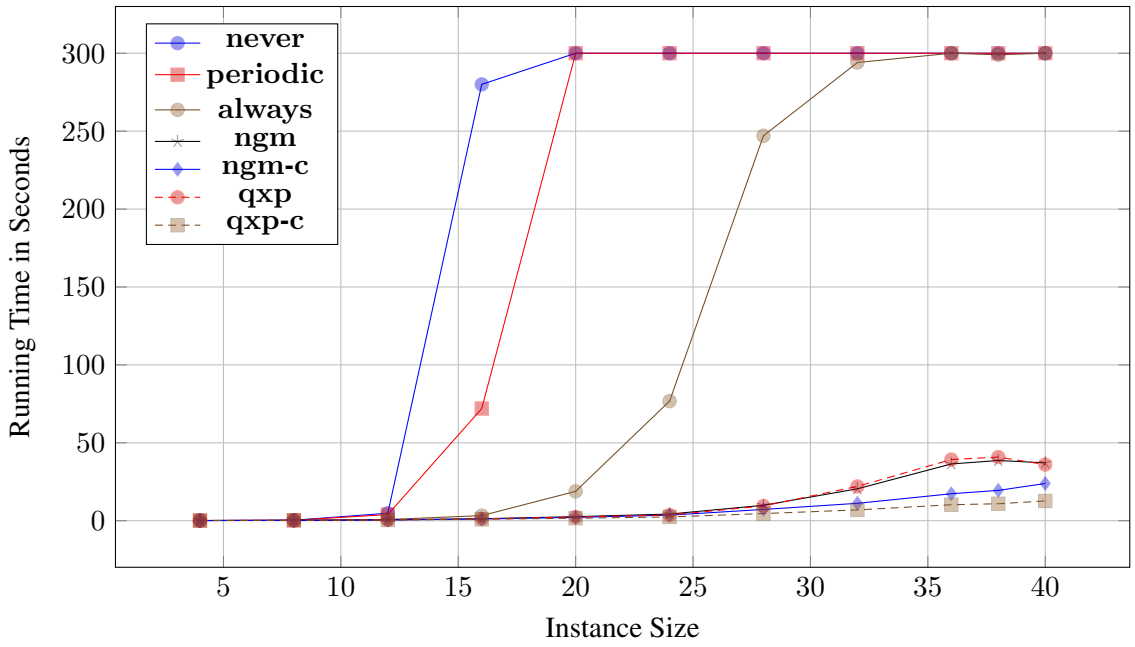

Figure 6: Results for random PB-problems with 4 to 40 variables (all answer sets).

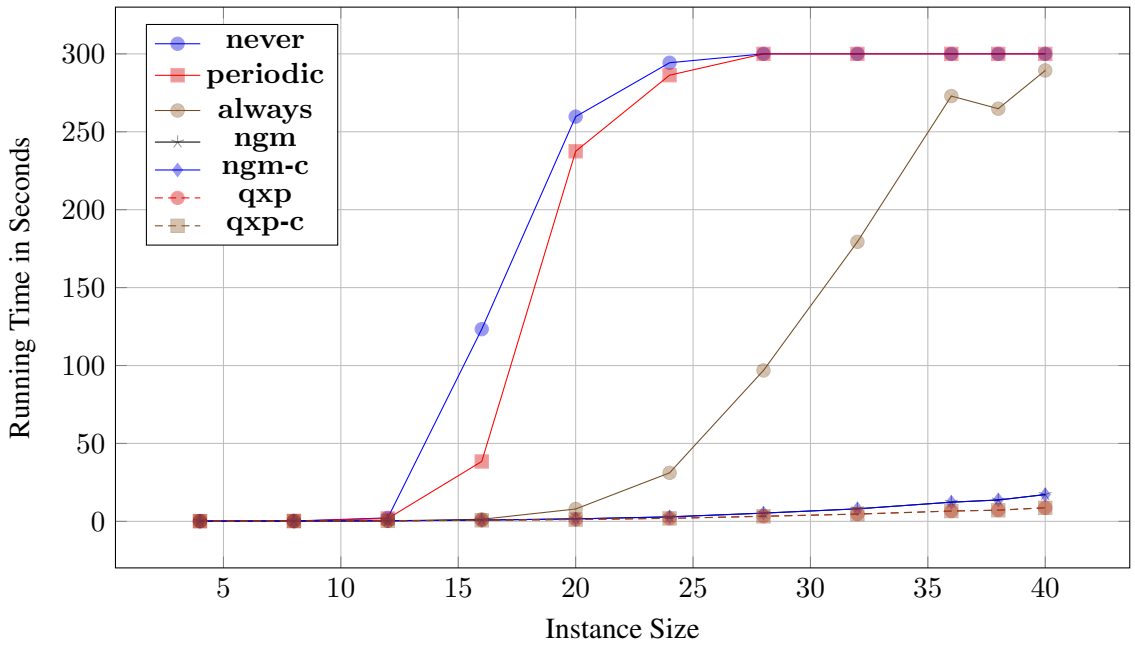

Figure 7: Results for random PB-problems with 4 to 40 variables (first answer set).

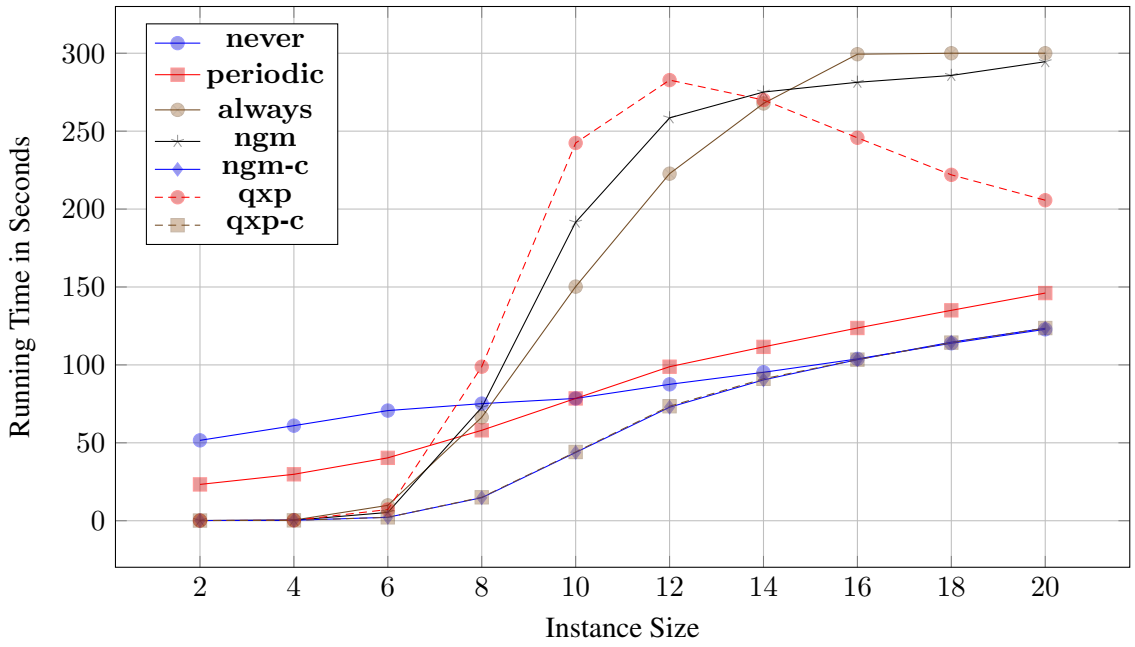

Figure 8: Results for random PB-problems with PB-constraint length of 2 to 20 (all answer sets).

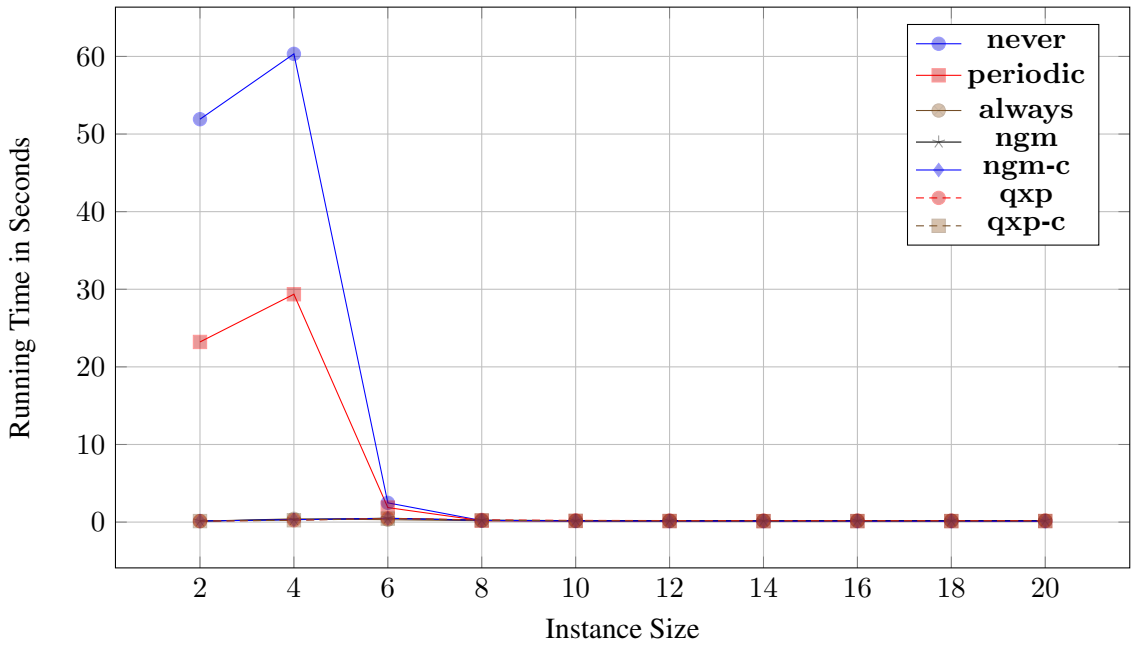

Figure 9: Results for random PB-problems with PB-constraint length of 2 to 20 (first answer set).

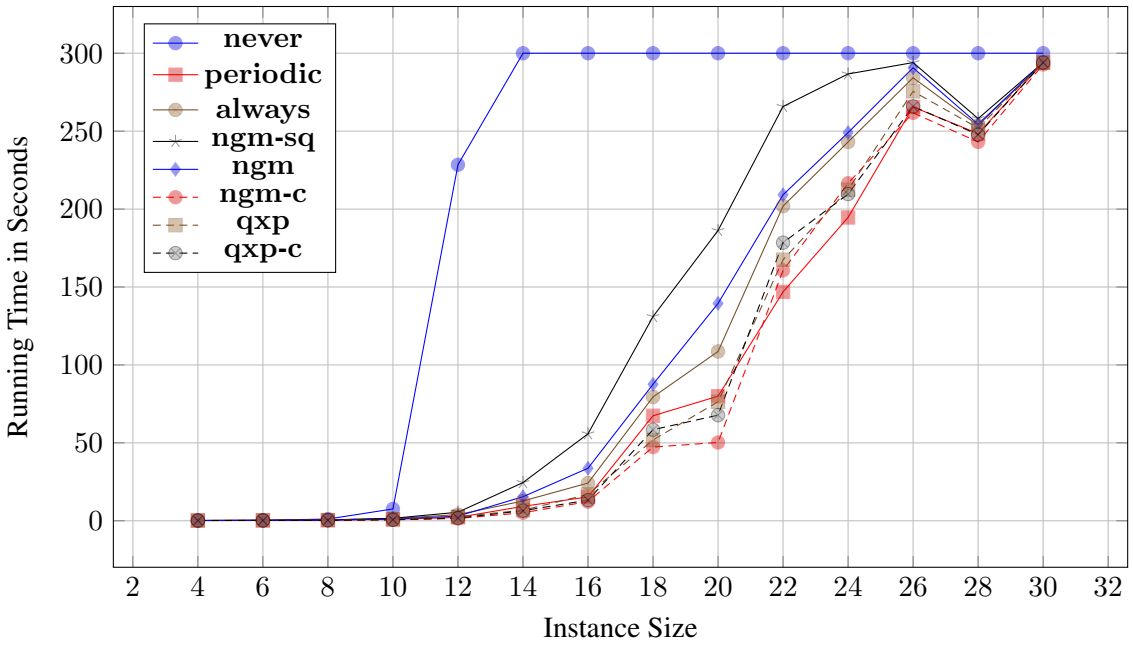

Figure 10: Results for taxi assignment with ontology access (all answer sets).

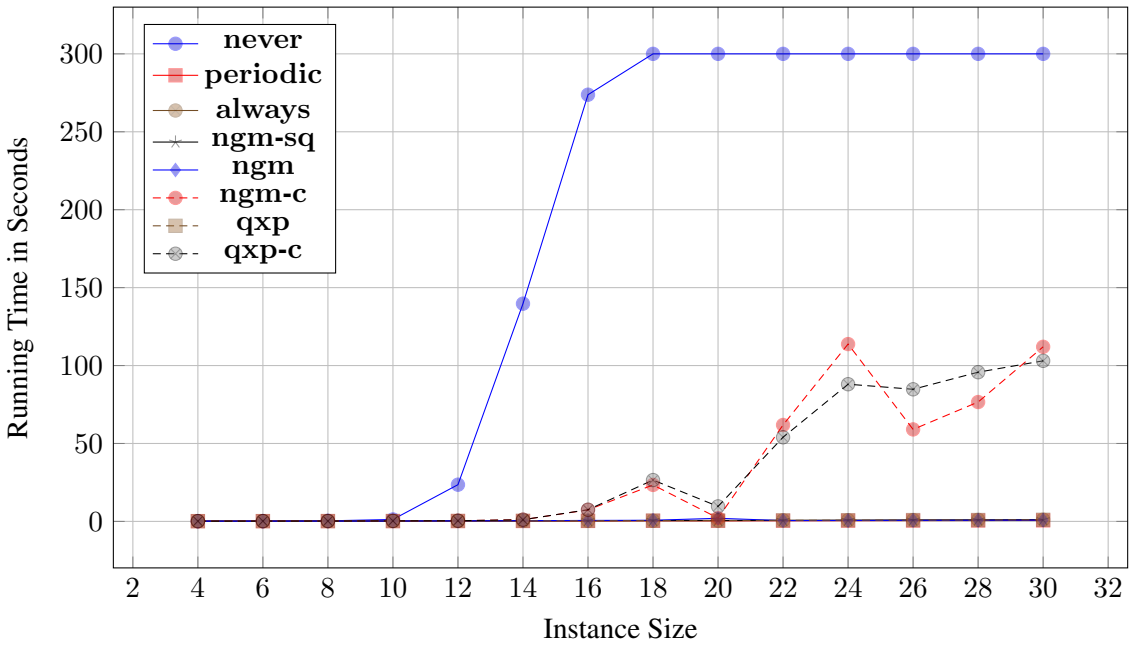

Figure 11: Results for taxi assignment with ontology access (first answer set).

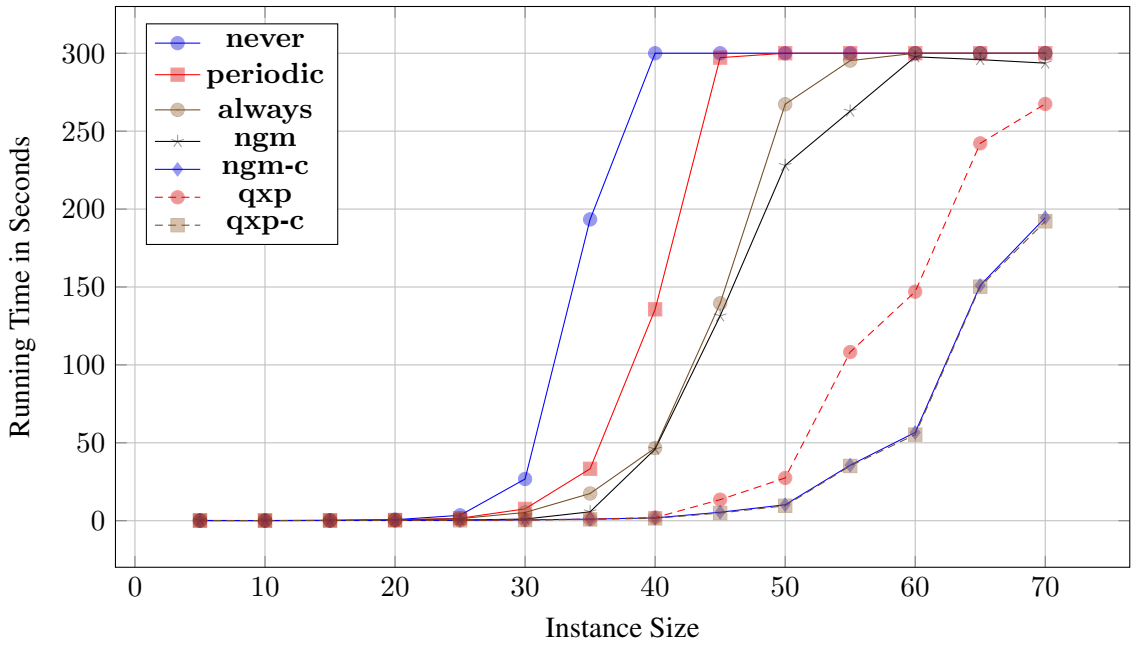

Figure 12: Results for conflicting strategic companies (all answer sets).

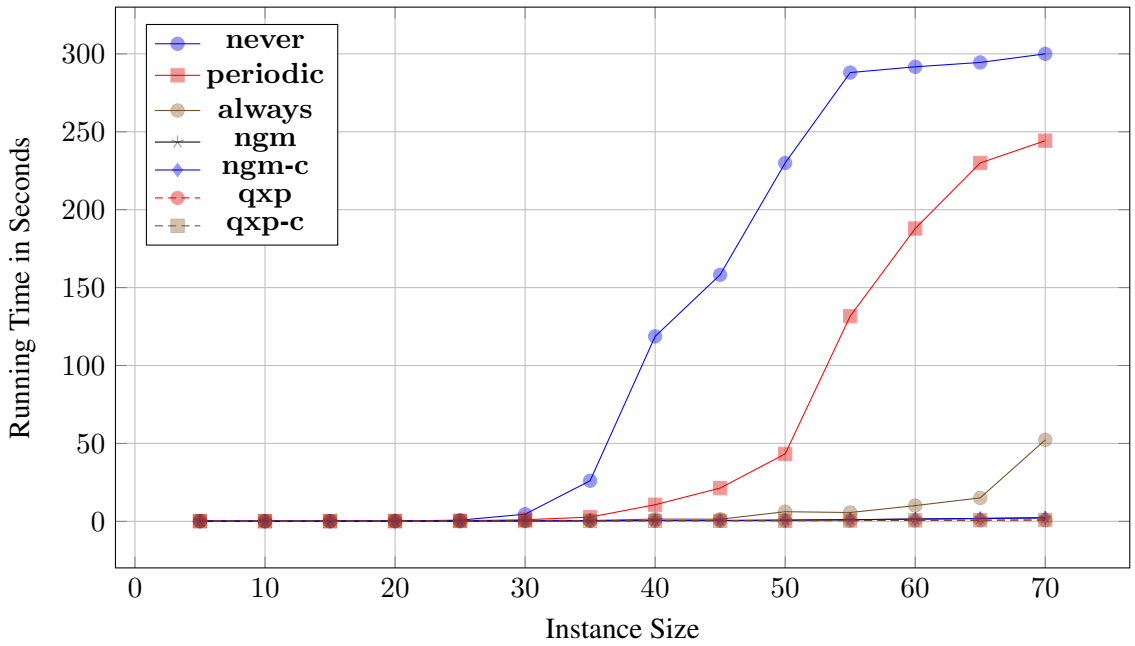

Figure 13: Results for conflicting strategic companies (first answer set).

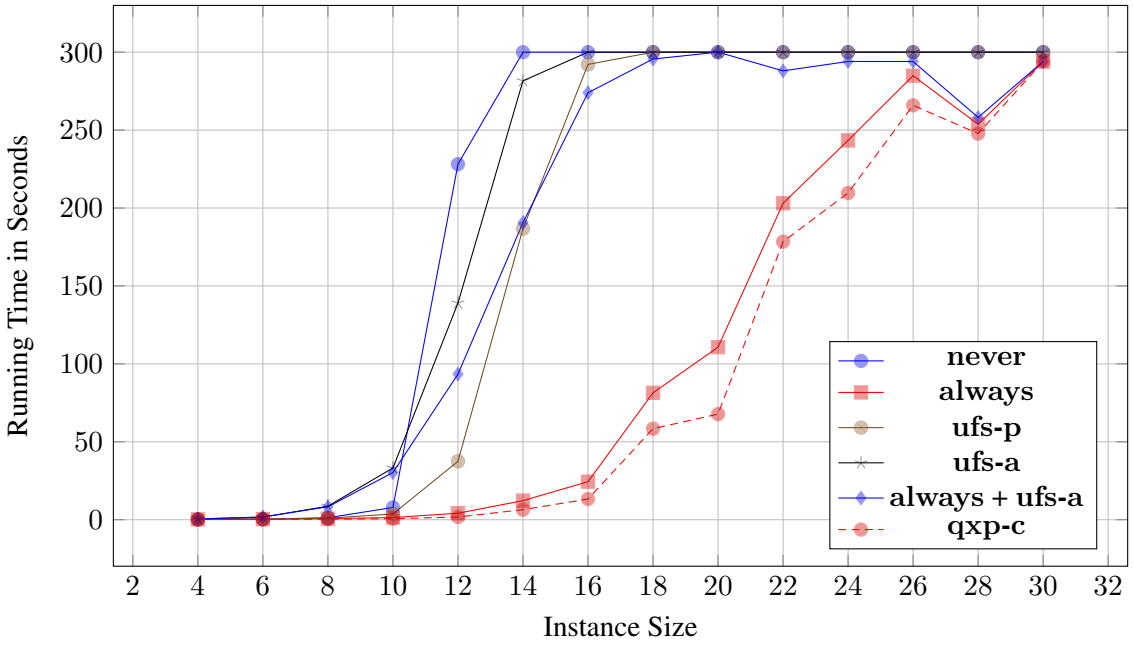

Figure 14: Results for taxi assignment with ontology access wrt. partial evaluation during unfounded set search (all answer sets).

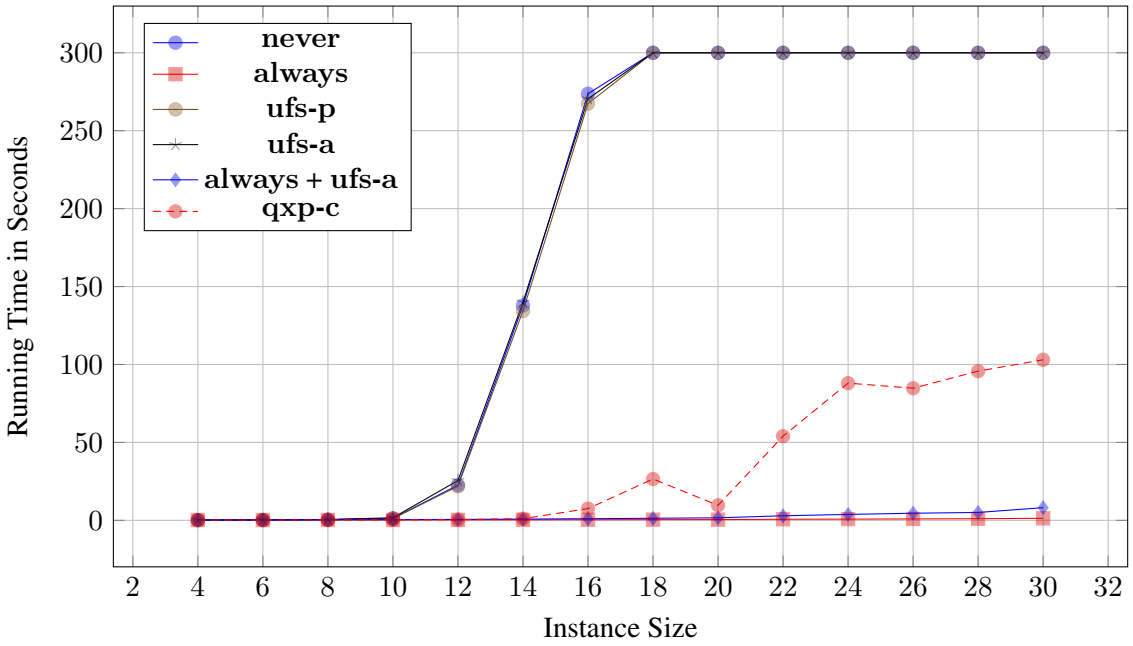

Figure 15: Results for taxi assignment with ontology access wrt. partial evaluation during unfounded set search (first answer set).

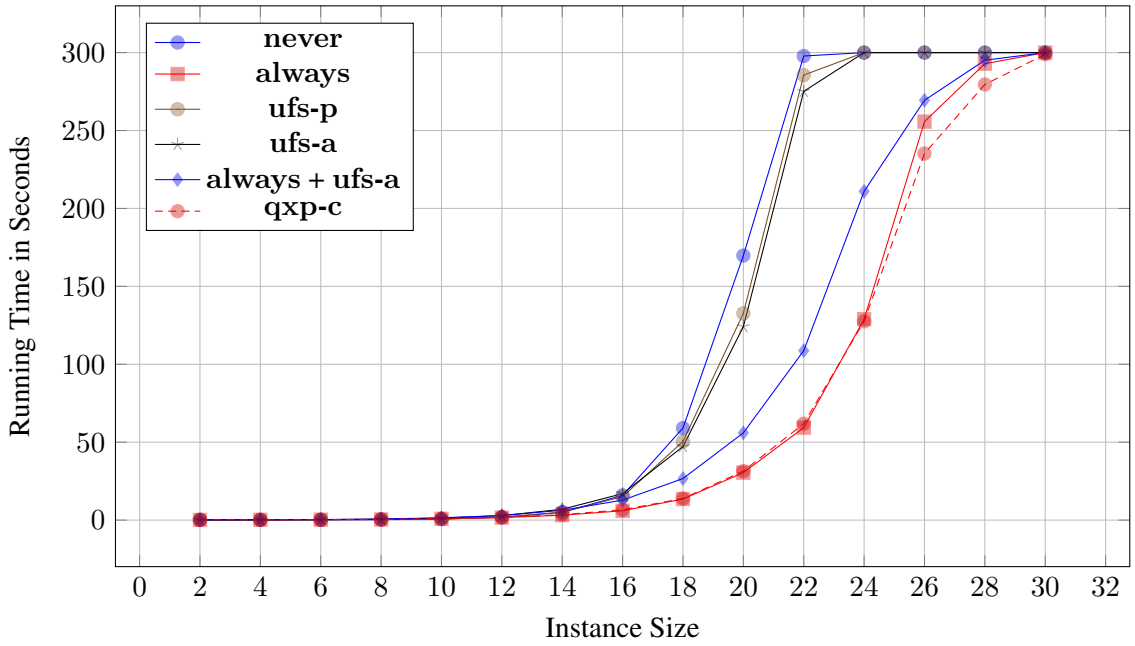

Figure 16: Results for strategic companies with external controls relation (all answer sets).

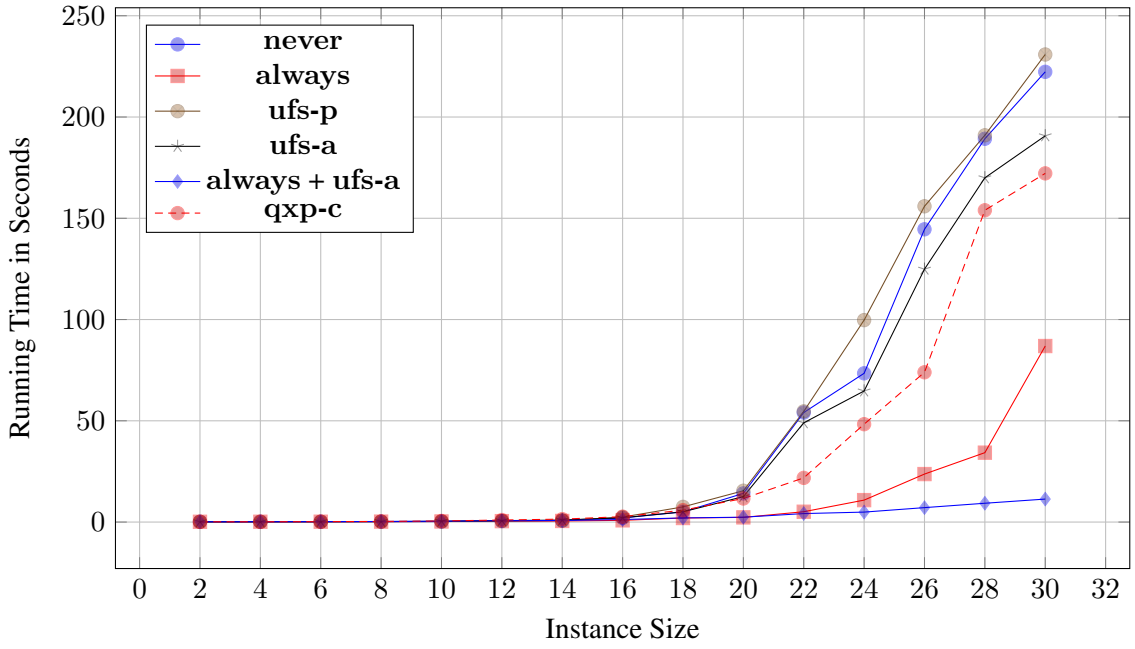

Figure 17: Results for strategic companies with external controls relation (first answer set).

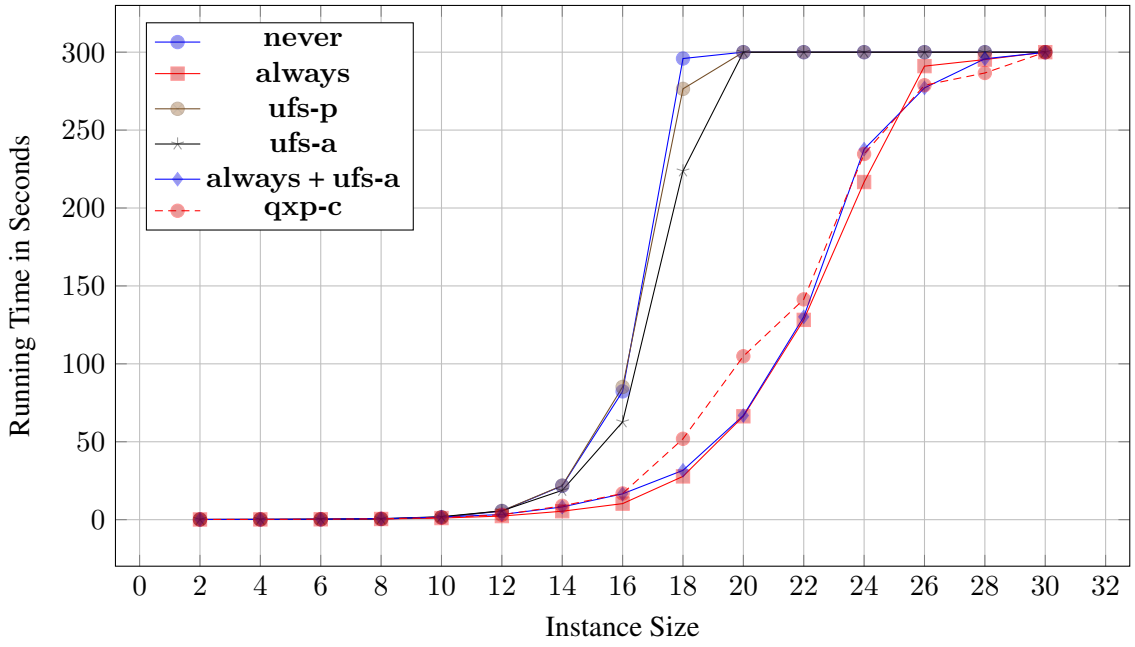

Figure 18: Results for strategic companies with nonmonotonic external controls relation (all answer sets).

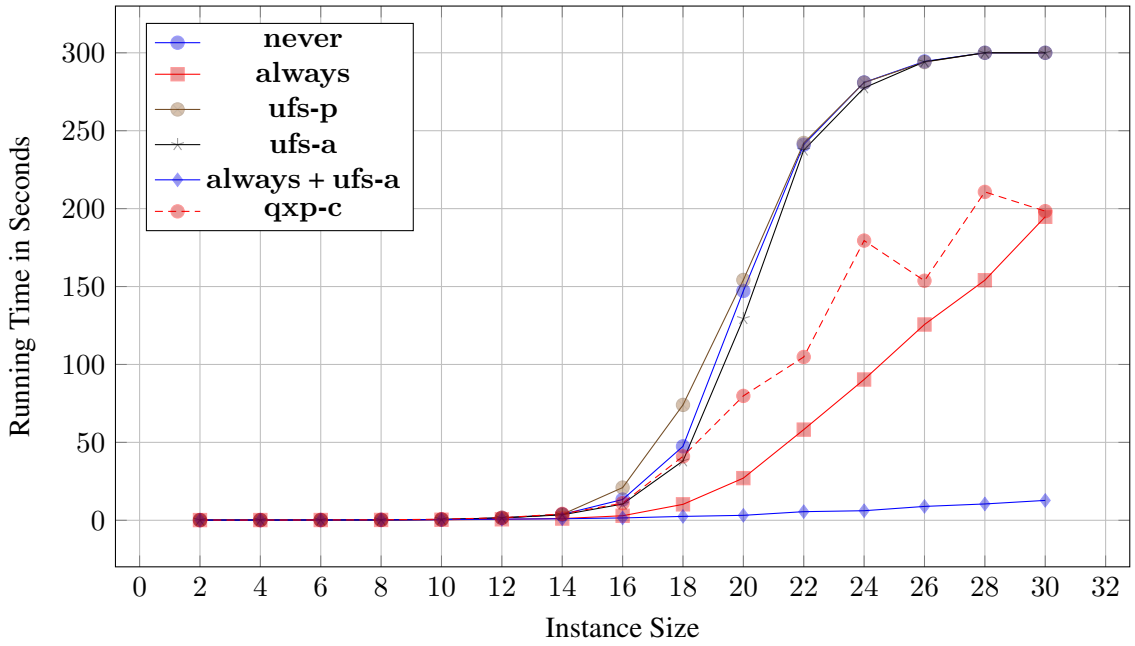

Figure 19: Results for strategic companies with nonmonotonic external controls relation (first answer set).

References

Alviano, M., Dodaro, C., Leone, N., & Ricca, F. (2015a). Advances in WASP. In Calimeri, F., Ianni, G., & Truszczynski, M. (Eds.), *Logic Programming and Nonmonotonic Reasoning - 13th International Conference, LPNMR 2015, Lexington, KY, USA, September 27-30, 2015. Proceedings*, Vol. 9345 of *Lecture Notes in Computer Science*, pp. 40–54. Springer.

Alviano, M., Faber, W., & Gebser, M. (2015b). Rewriting recursive aggregates in answer set programming: back to monotonicity. *TPLP*, *15*(4-5), 559–573.

Antic, C., Eiter, T., & Fink, M. (2013). HEX semantics via approximation fixpoint theory. In Cabalar, P., & Son, T. C. (Eds.), *Logic Programming and Nonmonotonic Reasoning, 12th International Conference, LPNMR 2013, Corunna, Spain, September 15-19, 2013. Proceedings*, Vol. 8148 of *Lecture Notes in Computer Science*, pp. 102–115. Springer.

Balduccini, M. (2009). Representing constraint satisfaction problems in answer set programming. In *Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP) at ICLP*.

Balduccini, M., & Lierler, Y. (2013a). Hybrid automated reasoning tools: from black-box to clear-box integration. *CoRR*, *abs/1312.6105*.

Balduccini, M., & Lierler, Y. (2013b). Integration schemas for constraint answer set programming: a case study. *TPLP*, *13*(4-5-Online-Supplement).

Barrett, C. W., Sebastiani, R., Seshia, S. A., & Tinelli, C. (2009). Satisfiability modulo theories. In Biere, A., Heule, M., van Maaren, H., & Walsh, T. (Eds.), *Handbook of Satisfiability*, Vol. 185 of *Frontiers in Artificial Intelligence and Applications*, pp. 825–885. IOS Press.

Brewka, G., Eiter, T., & Truszczynski, M. (2011). Answer set programming at a glance. *Commun. ACM*, *54*(12), 92–103.

Cabalar, P., Kaminski, R., Ostrowski, M., & Schaub, T. (2016). An ASP semantics for default reasoning with constraints. In Kambhampati, S. (Ed.), *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pp. 1015–1021. IJCAI/AAAI Press.

Cadoli, M., Eiter, T., & Gottlob, G. (1997). Default logic as a query language. *IEEE Trans. Knowl. Data Eng.*, *9*(3), 448–463.

Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., & Rosati, R. (2007). Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. *J. Autom. Reasoning*, *39*(3), 385–429.

Darwiche, A., & Marquis, P. (2002). A knowledge compilation map. *J. Artif. Intell. Res.*, *17*, 229–264.

Denecker, M., Marek, V., & Truszczyński, M. (2000). Approximations, stable operators, well-founded fixpoints and applications in nonmonotonic reasoning. In Minker, J. (Ed.), *Logic-Based Artificial Intelligence,* volume 597 of *The Springer International Series in Engineering and Computer Science*, pp. 127–144. Kluwer Academic Publishers, Norwell, Massachusetts.

Denecker, M., Marek, V. W., & Truszczynski, M. (2004). Ultimate approximation and its application in nonmonotonic knowledge representation systems. *Inf. Comput.*, *192*(1), 84–121.

Drescher, C., Gebser, M., Grote, T., Kaufmann, B., König, A., Ostrowski, M., & Schaub, T. (2008). Conflict-driven disjunctive answer set solving. In Brewka, G., & Lang, J. (Eds.), *Principles of Knowledge Representation and Reasoning: Proceedings of the Eleventh International Conference, KR 2008, Sydney, Australia, September 16-19, 2008*, pp. 422–432. AAAI Press.

Dutertre, B., & de Moura, L. M. (2006). A fast linear-arithmetic solver for DPLL(T). In Ball, T., & Jones, R. B. (Eds.), *Computer Aided Verification, 18th International Conference, CAV 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, Vol. 4144 of *Lecture Notes in Computer Science*, pp. 81–94. Springer.

Eiter, T., Fink, M., Ianni, G., Krennwallner, T., Redl, C., & Schüller, P. (2016). A model building framework for answer set programming with external computations. *TPLP*, *16*(4), 418–464.

Eiter, T., Fink, M., Krennwallner, T., & Redl, C. (2012). Conflict-driven ASP solving with external sources. *TPLP*, *12*(4-5), 659–679.

Eiter, T., Fink, M., Krennwallner, T., & Redl, C. (2016). Domain expansion for ASP-programs with external sources. *Artif. Intell.*, *233*, 84–121.

Eiter, T., Fink, M., Krennwallner, T., Redl, C., & Schüller, P. (2014a). Efficient HEX-program evaluation based on unfounded sets. *J. Artif. Intell. Res.*, *49*, 269–321.

Eiter, T., Fink, M., Redl, C., & Stepanova, D. (2014b). Exploiting support sets for answer set programs with external evaluations. In Brodley, C. E., & Stone, P. (Eds.), *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, pp. 1041–1048. AAAI Press.

Eiter, T., Fink, M., & Stepanova, D. (2016). Data repair of inconsistent nonmonotonic description logic programs. *Artif. Intell.*, *239*, 7–53.

Eiter, T., Ianni, G., Lukasiewicz, T., Schindlauer, R., & Tompits, H. (2008). Combining answer set programming with description logics for the semantic web. *Artif. Intell.*, *172*(12-13), 1495–1539.

Eiter, T., Ianni, G., Schindlauer, R., & Tompits, H. (2005a). NLP-DL: A KR system for coupling nonmonotonic logic programs with description logics. In Mizoguchi, R. (Ed.), *Poster & Demonstration Proceedings of the 4th International Semantic Web Conference (ISWC 2005)*, p. PID 67. System poster.

Eiter, T., Ianni, G., Schindlauer, R., & Tompits, H. (2005b). A uniform integration of higher-order reasoning and external evaluations in answer-set programming. In Kaelbling, L. P., & Saffiotti, A. (Eds.), *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30 - August 5, 2005*, pp. 90–96. Professional Book Center.

Eiter, T., Kaminski, T., Redl, C., Schüller, P., & Weinzierl, A. (2017). Answer set programming with external source access. In Ianni, G., Lembo, D., Bertossi, L. E., Faber, W., Glimm, B., Gottlob, G., & Staab, S. (Eds.), *Reasoning Web. Semantic Interoperability on the Web - 13th International Summer School 2017, London, UK, July 7-11, 2017, Tutorial Lectures*, Vol. 10370 of *Lecture Notes in Computer Science*, pp. 204–275. Springer.

Eiter, T., Kaminski, T., Redl, C., & Weinzierl, A. (2016). Exploiting partial assignments for efficient evaluation of answer set programs with external source access. In Kambhampati, S.

(Ed.), *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pp. 1058–1065. IJCAI/AAAI Press.

Eiter, T., Kaminski, T., & Weinzierl, A. (2017). Lazy-grounding for answer set programs with external source access. In Sierra, C. (Ed.), *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pp. 1015–1022. ijcai.org.

Eiter, T., Lukasiewicz, T., Schindlauer, R., & Tompits, H. (2004). Combining answer set programming with description logics for the semantic web. In Dubois, D., Welty, C. A., & Williams, M. (Eds.), *Principles of Knowledge Representation and Reasoning: Proceedings of the Ninth International Conference (KR2004), Whistler, Canada, June 2-5, 2004*, pp. 141–151. AAAI Press.

Eiter, T., Redl, C., & Schüller, P. (2016). Problem solving using the HEX family. In Beierle, C., Brewka, G., & Thimm, M. (Eds.), *Computational Models of Rationality, Essays dedicated to Gabriele Kern-Isberner on the occasion of her 60th birthday*, pp. 150–174. College Publications.

Erdem, E., Gelfond, M., & Leone, N. (2016). Applications of answer set programming. *AI Magazine*, *37*(3), 53–68.

Faber, W. (2005). Unfounded sets for disjunctive logic programs with arbitrary aggregates. In Baral, C., Greco, G., Leone, N., & Terracina, G. (Eds.), *Logic Programming and Nonmonotonic Reasoning, 8th International Conference, LPNMR 2005, Diamante, Italy, September 5-8, 2005, Proceedings*, Vol. 3662 of *Lecture Notes in Computer Science*, pp. 40–52. Springer.

Faber, W., Pfeifer, G., & Leone, N. (2011). Semantics and complexity of recursive aggregates in answer set programming. *Artif. Intell.*, *175*(1), 278–298.

Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T., & Wanko, P. (2016). Theory solving made easy with clingo 5. In Carro, M., King, A., Saeedloei, N., & Vos, M. D. (Eds.), *Technical Communications of the 32nd International Conference on Logic Programming, ICLP 2016 TCs, October 16-21, 2016, New York City, USA*, Vol. 52 of *OASICS*, pp. 2:1–2:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik.

Gebser, M., Kaufmann, B., Kaminski, R., Ostrowski, M., Schaub, T., & Schneider, M. T. (2011). Potassco: The potsdam answer set solving collection. *AI Commun.*, *24*(2), 107–124.

Gebser, M., Kaufmann, B., Neumann, A., & Schaub, T. (2007). Conflict-driven answer set enumeration. In Baral, C., Brewka, G., & Schlipf, J. S. (Eds.), *Logic Programming and Nonmonotonic Reasoning, 9th International Conference, LPNMR 2007, Tempe, AZ, USA, May 15-17, 2007, Proceedings*, Vol. 4483 of *Lecture Notes in Computer Science*, pp. 136–148. Springer.

Gebser, M., Kaufmann, B., & Schaub, T. (2012). Conflict-driven answer set solving: From theory to practice. *Artif. Intell.*, *187*, 52–89.

Gebser, M., Kaufmann, B., & Schaub, T. (2013). Advanced conflict-driven disjunctive answer set solving. In Rossi, F. (Ed.), *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, pp. 912–918. IJCAI/AAAI.

Gebser, M., Ostrowski, M., & Schaub, T. (2009). Constraint answer set solving. In Hill, P. M., & Warren, D. S. (Eds.), *Logic Programming, 25th International Conference, ICLP 2009,*

Pasadena, CA, USA, July 14-17, 2009. Proceedings, Vol. 5649 of *Lecture Notes in Computer Science*, pp. 235–249. Springer.

Gebser, M., & Schaub, T. (2016). Modeling and language extensions. *AI Magazine*, *37*(3), 33–44.

Gelfond, M., & Lifschitz, V. (1991). Classical negation in logic programs and disjunctive databases. *New Generation Comput.*, *9*(3/4), 365–386.

Gent, I. P., Miguel, I., & Moore, N. C. A. (2010). Lazy explanations for constraint propagators. In Carro, M., & Peña, R. (Eds.), *Practical Aspects of Declarative Languages, 12th International Symposium, PADL 2010, Madrid, Spain, January 18-19, 2010. Proceedings*, Vol. 5937 of *Lecture Notes in Computer Science*, pp. 217–233. Springer.

HTCondor Website (2018) http://research.cs.wisc.edu/htcondor. Accessed: 2018-06-27.

Janhunen, T., Liu, G., & Niemelä, I. (2013). Tight integration of non-ground answer set programming and satisfiability modulo theories. In Cabalar, P., Mitchell, D., Pearce, D., & Ternovska, E. (Eds.), *Informal Proceedings of the 1st Workshop on Grounding and Transformations for Theories with Variables (GTTV'11), LPNMR, Vancouver, BC, Canada May 16th, 2011*, pp. 1–14.

Junker, U. (2004). QUICKXPLAIN: Preferred explanations and relaxations for over-constrained problems. In McGuinness, D. L., & Ferguson, G. (Eds.), *Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence, July 25-29, 2004, San Jose, California, USA*, pp. 167–172. AAAI Press / The MIT Press.

Karp, R. M. (1972). Reducibility among combinatorial problems. In Miller, R. E., & Thatcher, J. W. (Eds.), *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York.*, The IBM Research Symposia Series, pp. 85–103. Plenum Press, New York.

Lahiri, S. K., Nieuwenhuis, R., & Oliveras, A. (2006). SMT techniques for fast predicate abstraction. In Ball, T., & Jones, R. B. (Eds.), *Computer Aided Verification, 18th International Conference, CAV 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, Vol. 4144 of *Lecture Notes in Computer Science*, pp. 424–437. Springer.

Lee, J., & Meng, Y. (2013). Answer set programming modulo theories and reasoning about continuous changes. In Rossi, F. (Ed.), *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, pp. 990–996. IJCAI-/AAAI.

Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., & Scarcello, F. (2006). The DLV system for knowledge representation and reasoning. *ACM Trans. Comput. Log.*, *7*(3), 499–562.

Leone, N., Rullo, P., & Scarcello, F. (1997). Disjunctive stable models: Unfounded sets, fixpoint semantics, and computation. *Inf. Comput.*, *135*(2), 69–112.

Lierler, Y., Maratea, M., & Ricca, F. (2016). Systems, engineering environments, and competitions. *AI Magazine*, *37*(3), 45–52.

Manquinho, V. M., & Silva, J. P. M. (2005). Effective lower bounding techniques for pseudo-boolean optimization. In *2005 Design, Automation and Test in Europe Conference and Ex-*

position (DATE 2005), 7-11 March 2005, Munich, Germany, pp. 660–665. IEEE Computer Society.

Marques-Silva, J. P., Lynce, I., & Malik, S. (2009). Conflict-driven clause learning SAT solvers. In Biere, A., Heule, M., van Maaren, H., & Walsh, T. (Eds.), *Handbook of Satisfiability*, Vol. 185 of *Frontiers in Artificial Intelligence and Applications*, pp. 131–153. IOS Press.

Merriam-Webster Website (2018) https://www.merriam-webster.com/thesaurus. Accessed: 2018-06-27.

Nieuwenhuis, R., & Oliveras, A. (2005). DPLL(T) with exhaustive theory propagation and its application to difference logic. In Etessami, K., & Rajamani, S. K. (Eds.), *Computer Aided Verification, 17th International Conference, CAV 2005, Edinburgh, Scotland, UK, July 6-10, 2005, Proceedings*, Vol. 3576 of *Lecture Notes in Computer Science*, pp. 321–334. Springer.

Nieuwenhuis, R., Oliveras, A., & Tinelli, C. (2006). Solving SAT and SAT modulo theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T). *J. ACM*, *53*(6), 937–977.

Ostrowski, M., & Schaub, T. (2012). ASP modulo CSP: The clingcon system. *TPLP*, *12*(4-5), 485–503.

Pelov, N., Denecker, M., & Bruynooghe, M. (2004). Partial stable models for logic programs with aggregates. In Lifschitz, V., & Niemelä, I. (Eds.), *Logic Programming and Nonmonotonic Reasoning, 7th International Conference, LPNMR 2004, Fort Lauderdale, FL, USA, January 6-8, 2004, Proceedings*, Vol. 2923 of *Lecture Notes in Computer Science*, pp. 207–219. Springer.

Redl, C. (2017). Efficient evaluation of answer set programs with external sources based on external source inlining. In Singh, S. P., & Markovitch, S. (Eds.), *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA.*, pp. 1222–1228. AAAI Press.

Reiter, R., & de Kleer, J. (1987). Foundations of assumption-based truth maintenance systems: Preliminary report. In Forbus, K. D., & Shrobe, H. E. (Eds.), *Proceedings of the 6th National Conference on Artificial Intelligence. Seattle, WA, July 1987.*, pp. 183–189. Morgan Kaufmann.

Roussel, O., & Manquinho, V. M. (2009). Pseudo-boolean and cardinality constraints. In Biere, A., Heule, M., van Maaren, H., & Walsh, T. (Eds.), *Handbook of Satisfiability*, Vol. 185 of *Frontiers in Artificial Intelligence and Applications*, pp. 695–733. IOS Press.

Shchekotykhin, K. M., Jannach, D., & Schmitz, T. (2015). MergeXplain: Fast computation of multiple conflicts for diagnosis. In Yang, Q., & Wooldridge, M. (Eds.), *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pp. 3221–3228. AAAI Press.

Shen, Y., Wang, K., Eiter, T., Fink, M., Redl, C., Krennwallner, T., & Deng, J. (2014). FLP answer set semantics without circular justifications for general logic programs. *Artif. Intell.*, *213*, 1–41.

Simons, P., Niemelä, I., & Soininen, T. (2002). Extending and implementing the stable model semantics. *Artif. Intell.*, *138*(1-2), 181–234.

Sörensson, N., & Biere, A. (2009). Minimizing learned clauses. In Kullmann, O. (Ed.), *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings*, Vol. 5584 of *Lecture Notes in Computer Science*, pp. 237–243. Springer.

Susman, B., & Lierler, Y. (2016). Smt-based constraint answer set solver EZSMT (system description). In Carro, M., King, A., Saeedloei, N., & Vos, M. D. (Eds.), *Technical Communications of the 32nd International Conference on Logic Programming, ICLP 2016 TCs, October 16-21, 2016, New York City, USA*, Vol. 52 of *OASICS*, pp. 1:1–1:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik.

Valiant, L. G. (1984). A theory of the learnable. *Commun. ACM*, *27*(11), 1134–1142.

Weinzierl, A. (2017). Blending lazy-grounding and CDNL search for answer-set solving. In Balduccini, M., & Janhunen, T. (Eds.), *Logic Programming and Nonmonotonic Reasoning - 14th International Conference, LPNMR 2017, Espoo, Finland, July 3-6, 2017, Proceedings*, Vol. 10377 of *Lecture Notes in Computer Science*, pp. 191–204. Springer.