Transforming Complete Families of Support Sets. One can change the polarity of complete families of support sets:

Proposition 4. *Let $\mathcal{S}_\sigma$ be a positive resp. negative complete family of support sets for some external atom e in a program P, where $\sigma \in \{\mathbf{T}, \mathbf{F}\}$. Then $\mathcal{S}_{\bar{\sigma}} = \{S_{\bar{\sigma}} \in \prod_{S_\sigma \in \mathcal{S}_\sigma} \neg S_\sigma \mid S_{\bar{\sigma}}$ is consistent$\}$ is a negative resp. positive complete family of support sets, where $\overline{\mathbf{T}} = \mathbf{F}$ and $\overline{\mathbf{F}} = \mathbf{T}$.*

However, similarly to a transformation of the formula in conjunctive normal form to disjunctive normal form or vice versa, this may result in an exponential blow-up. In the spirit of our initial assumption that compact complete families of support sets exist, it is suggested to construct families of support sets of the required polarity right from the beginning.

5 Implementation and Evaluation

We implemented external source inlining in the DLVHEX system, which is based on GRINGO and CLASP. External sources are supposed to provide a complete set of (possibly nonground) support sets. The approach from this paper allows for evaluating a HEX-program completely by the backend without any external calls during solving (external calls are only necessary at the beginning for support set learning).

The rewriting makes both the compatibility check (cf. Definition 3) and the minimality check wrt. the reduct and external sources (cf. Section 2 and Eiter et al. [2014a]) obsolete.

Experimental Setup. We compare three evaluation approaches. The **traditional** evaluation algorithm guesses the truth values of external atoms and verifies them by evaluation. During evaluation, conflict-driven learning techniques are applied to learn parts of the external atom's behavior. The approach based on **support sets** (**sup.sets**) learns support sets provided by the external source at the beginning. It then guesses external atoms as in the traditional approach, but verifies them by matching candidate compatible sets against support sets rather than by evaluation. The new **inlining** approach also learns support sets at the beginning, but uses them for rewriting external atoms as demonstrated in Section 3. Then all answer sets of the rewritten ASP program are accepted without the necessity for additional checks.

We present four benchmarks with 100 randomly generated instances each, which were run on a Linux server with two 12-core AMD 6176 SE CPUs/128GB RAM using a 300 secs timeout. Despite similar benchmarks, the runtimes are not directly comparable to those by Eiter et al. [2014b] because of other solver improvements in the meantime and, for the taxi benchmark, an adopted scenario; however, the trends concerning **sup.sets** and **traditional** are the same. Note that our goal is to show improvements compared to previous HEX-algorithms, but not to compare HEX to other formalisms.

Our hypothesis is that **inlining** outperforms both **traditional** and **sup.sets**. This is because the only significant costs when generating the rewriting come from support set learning. However, this is also necessary with **sup.sets**, which was already shown to outperform **traditional** if small complete families of support sets exist (as in our benchmarks).[3] On the other hand, with **inlining**, (i) no external calls and (ii) no additional minimality check are needed. Hence, we expect further benefits and negligible additional costs.

n	all answer sets						first answer set					
	traditional		sup.sets		inlining		traditional		sup.sets		inlining	
6	185.45	(35)	23.57	(1)	11.47	(0)	12.05	(0)	1.09	(0)	0.76	(0)
7	251.68	(81)	83.24	(3)	22.21	(2)	22.25	(2)	3.19	(0)	1.53	(0)
8	266.22	(85)	183.48	(43)	59.54	(11)	61.33	(10)	22.42	(1)	3.10	(0)
9	272.70	(85)	263.01	(85)	86.07	(13)	76.74	(12)	56.57	(12)	6.18	(0)
10	278.26	(83)	275.47	(83)	121.39	(16)	102.86	(12)	98.96	(12)	11.97	(0)
11	292.05	(85)	300.00	(100)	167.00	(45)	158.73	(41)	176.44	(49)	22.52	(0)
12	300.00	(100)	300.00	(100)	180.43	(41)	159.64	(47)	210.52	(51)	40.43	(0)

Table 1: House Configuration Problem

House Problem. An abstraction of configuration problems considers sets of *cabinets*, *rooms*, *objects* and *persons* and assigns cabinets to persons, cabinets to rooms, and objects to cabinets, such that there are no more than four cabinets in a room or more than five objects in a cabinet (Mayer et al. 2009). Objects belonging to a person must be stored in a cabinet belonging to the same person, and a room must not contain cabinets of more than one person. We assume that we have already a partial assignment to be completed. We use an existing guess-and-check encoding[4] which implements the check as external source. Instances of size n have n persons, $n+2$ cabinets, $n+1$ rooms, and $2n$ objects randomly assigned to persons; $2n-2$ objects are already stored.

Table 1 shows the results, where numbers in parentheses indicate timeout instances. We have that **sup.sets** clearly outperforms **traditional** when computing all answer sets due to faster candidate checking; **inlining** leads to a further speedup as it eliminates wrong guesses and the checking step altogether, while the additional initialization overhead is negligible. If only one answer set is computed, this initialization overhead even exceeds the benefits of **sup.sets** in some cases because the benefit is limited due to early abortion (as also observed by Eiter et al. [2014b]). However, the further performance boost by **inlining** compensates this drawback of **sup.sets** s.t. it is clearly the most efficient configuration.

Non 3-Colorability. We consider the problem of deciding if a given graph is *not* 3-colorable, i.e., if it is not possible to color the nodes s.t. adjacent nodes have different colors. We use an encoding which splits the guessing part P_{col} from the checking part P_{check}. The latter is used as an external source from the guessing part. For a color assignment, given by facts of kind $inp(col, v, c)$ where v is a vertex and c is a color, P_{check} derives the atom inv in its only answer set, otherwise it has an empty answer set. We then use the following program P_{col} to guess a coloring and check it using the external atom $\&query[P_{check}, inp, inv]()$, which is true iff P_{check}, extended with facts over predicate inp, delivers an answer set which contains inv. A compact complete family

[3] If they are not small, **traditional** might be faster than **sup.sets** and **inlining**. Consider $P = \{p(n+1) \leftarrow \&even[p]()\} \cup \{p(i) \leftarrow \mid 1 \leq i \leq n\}$ where $\&even[p]()$ is true iff the number of true atoms over p is even. Then $\hat{P}$ has only two candidates which are easily checked, while exponentially many support sets must be generated.

[4] from http://143.205.174.183/reconcile/tools.