n	all answer sets						first answer set					
	traditional		sup.sets		inlining		traditional		sup.sets		inlining	
20	299.01	(99)	0.20	(0)	0.16	(0)	0.12	(0)	0.12	(0)	0.12	(0)
60	300.00	(100)	1.62	(0)	1.36	(0)	0.45	(0)	0.45	(0)	0.45	(0)
100	300.00	(100)	8.50	(0)	7.80	(0)	1.99	(0)	1.99	(0)	1.99	(0)
140	300.00	(100)	28.32	(0)	27.50	(0)	6.40	(0)	6.40	(0)	6.43	(0)
180	300.00	(100)	74.88	(0)	73.79	(0)	16.41	(0)	16.43	(0)	16.43	(0)
220	300.00	(100)	152.41	(21)	150.77	(20)	35.19	(0)	35.35	(0)	35.23	(0)

Table 2: Graph Coloring Problem

of support sets for $\&query[P_{check}, inp, inv]()$ exists. The size of the instances is the number of nodes n.

$$P_{col} = \left\{ \begin{array}{r} col(V,r) \lor col(V,g) \lor col(V,b) \leftarrow node(V), \\ inp(p,X,Y) \leftarrow p(X,Y) \mid p \in \{col, edge\}, \\ inval \leftarrow \&query[P_{check}, inp, inv](), \\ col(V,c) \leftarrow inval, node(V) \mid c \in \{r,g,b\} \end{array} \right\}$$

The results are shown in Table 2. While **sup.sets** already outperforms **traditional**, **inlining** leads to a further small speedup. Compared to the house problem, there are significantly fewer support sets, which makes candidate checking in **sup.sets** inexpensive. This explains the large speedup of **sup.sets** over **traditional**, and that avoiding the check in **inlining** does not lead to a large further speedup. However, due to a negligible additional overhead, **inlining** does not harm.

Taxi Assignment. We consider a program with access to an ontology, cf. *DL-atoms* (Eiter et al. 2008), to assign taxi *drivers* to *customers*. Each customer and driver is in a *region*. A customer may only be assigned to a driver in the same region. Up to four customers may be assigned to a driver. We let some customers be *e-customers* who use only electronic cars, and some drivers be *e-drivers* who drive electronic cars. The ontology stores information about individuals such as their locations (randomly chosen but balanced among regions). The encoding is from http://www.kr.tuwien.ac.at/research/projects/inthex/partialevaluation. An instance of size $4 \leq n \leq 9$ consists of n drivers, n customers including $n/2$ e-customers and $n/2$ regions.

Table 3 shows the results. Here, **sup.sets** is counterproductive compared to **traditional** because of the large number of solution candidates. Although we add support sets as constraints (cf. Section 2), positive resp. negative support sets prevent only wrong false resp. true guesses, but not vice versa. Hence, the costs of not learning from external calls exceed the benefit of faster checking. However, since **inlining** prevents wrong guesses, it does not suffer this problem.

LUBM Diamond. We consider default reasoning over the LUBM $DL\text{-}Lite_{\mathcal{A}}$ ontology (http://swat.cse.lehigh.edu/projects/lubm/). Defaults express that assistants are normally employees and students are normally not employees. The ontology entails that assistants are students, resembling Nixon's diamond. The instance size is the number of persons, which are randomly marked as students, assistants or employees.

Table 4 shows the results. As already observed by Eiter et al. [2014b] and different from the previous benchmark, **sup.sets** outperforms **traditional** due to a smaller number of model candidates, hence fewer wrong guesses occur and the more efficient check compensates the lost possibility to learn from external calls. Again, **inlining** is the most efficient configuration as it does not only prevent wrong guesses but also spares external calls. Thanks to a compact family support sets, the speedup is dramatic.

n	all answer sets						first answer set					
	traditional		sup.sets		inlining		traditional		sup.sets		inlining	
4	0.90	(0)	1.49	(0)	0.26	(0)	0.20	(0)	1.49	(0)	0.18	(0)
5	37.09	(3)	45.42	(0)	1.39	(0)	2.90	(0)	45.28	(0)	0.21	(0)
6	225.01	(59)	262.08	(74)	12.40	(0)	63.71	(17)	262.14	(75)	0.26	(0)
7	300.00	(100)	300.00	(100)	186.74	(29)	207.57	(67)	300.00	(100)	0.32	(0)
8	300.00	(100)	300.00	(100)	295.53	(97)	277.10	(92)	300.00	(100)	0.41	(0)
9	300.00	(100)	300.00	(100)	300.00	(100)	297.01	(99)	300.00	(100)	0.52	(0)

Table 3: Driver - Customer Assignment Problem

n	all answer sets						first answer set					
	traditional		sup.sets		inlining		traditional		sup.sets		inlining	
20	1.08	(0)	0.34	(0)	0.31	(0)	0.34	(0)	0.34	(0)	0.31	(0)
30	27.73	(3)	0.98	(0)	0.34	(0)	5.66	(0)	0.98	(0)	0.34	(0)
40	145.06	(35)	16.68	(2)	0.40	(0)	84.73	(14)	16.74	(2)	0.40	(0)
50	249.78	(76)	80.69	(15)	0.48	(0)	213.45	(60)	80.61	(15)	0.47	(0)
60	285.70	(90)	184.25	(47)	0.57	(0)	265.61	(85)	184.23	(47)	0.57	(0)
70	298.13	(99)	254.00	(74)	0.72	(0)	297.17	(99)	254.06	(73)	0.72	(0)

Table 4: Default Rules over LUBM in $DL\text{-}Lite_{\mathcal{A}}$

Experiments Summary. The size of the **inlining** encoding is linked to the size of the complete family of support sets. Although it is exponential in the worst case, many practical source have compact families of support sets. In this case, the **inlining** approach is clearly superior to **sup.sets** (which is superior to **traditional**) as it eliminates the compatibility check and minimality check wrt. external sources altogether, while it has only slightly higher initialization overhead.

6 Discussion and Conclusion

Related Work. Our approach is related to evaluation approaches for DL-programs (Eiter et al. 2008) (programs with ontologies), cf. e.g. Heymans, Eiter, and Xiao [2010], but is more general. The rewriting uses the saturation technique and is related to the one by Alviano, Faber, and Gebser [2015] who translated aggregates to disjunctions. However, they support only a fixed set of aggregates while our approach supports arbitrary sources. Moreover, it eliminates external atoms completely, while Alviano, Faber, and Gebser [2015] still use simplified (monotonic) aggregates in the result.

Conclusion and Outlook. We presented an approach for external source inlining based on support sets. The program can then be evaluated by an ordinary ASP solver and external atom guesses do not need to be verified at all. All our experiments show a clear improvement over the previous approach by Eiter et al. [2014b], which is explained by the fact that the slightly higher initialization costs are exceeded by the significant benefits of avoiding external calls altogether.