Our solution:

Problem:

- HEX-programs extend ASP by external sources:
  - Rule bodies may contain external atoms of the form \( \varphi[q_1, \ldots, q_n](t_1, \ldots, t_k) \),
  - where \( p \) . . . external predicate name,
  - \( q_1 \ldots \) predicate names or constants:

Although inlining leads to an exponential blowup in the worst case, it is

- Reasons include both algorithmic and technical overhead
- (e.g. caching effects).

Let our encoding is based on the saturation technique.

Formally:

- A positive support set of \( \vec{e} \) is a consistent set \( A \) that
  - contains all possibilities how to satisfy resp. falsify the external atom.

Example: Set Partitioning

\[
\begin{align*}
P = & \{ d(a_1), \ldots, d(a_e), \} \\
r_1: p(X) \iff d(X), & \text{ \& diffusion}[d, q](X) \\
r_2: q(X) \iff d(X), & \text{ \& diffusion}[d, p](X).
\end{align*}
\]

Problem:

- Calling external sources during solving is expensive.
- This is in particular the case for cyclic external sources.
- Reasons include both algorithmic and technical overhead

Our solution:

- Compile the HEX-program to an ordinary ASP-program by inlining external sources.
- To this end, we employ support sets
  - (i.e., sets of input atoms which make the external atom true).

Although inlining leads to an exponential blowup in the worst case, it is

- known that for certain types of external sources this is not the case!

2. Support Sets

Let \( e = \{ q[f(x)](x) \} \) be an external atom in a program \( P \).

Intuition:

A positive (resp. negative) support set is a set of positive or negated input atoms of \( e \), whose satisfaction implies that \( e \) is true (resp. false).

Formally:

A support set for \( e \) is a consistent set \( S_e = S^+ \cup S^- \) with \( \sigma \in \{ T, F \} \). \( S^+ \subseteq \text{HB}_e(P) \) and \( S^- \subseteq \text{HB}_e(P) \) s.t.

- \( \sigma \cap S^+ = 0 \) and \( \sigma \cap S^- = 0 \)
- \( \text{HB}_e(P) \) implies \( \sigma \) if \( \sigma = T \) and \( \sigma \neq e \) if \( \sigma = F \) for all assignments \( \sigma \).

Example: Set Partitioning (cont’d)

A positive support set of \( \text{diff}[d, q](b) \) in \( P \) is \( S_1 = \{ Td(b), Fq(b) \} \) since for all \( A: A \ |= d(b) \) and \( A \not|= q(b) \) implies \( A \ |= \text{diff}[d, q](b) \).

Important concept: complete families of support sets:

- A family (set) of support sets \( S_e \) for external atom \( e \) is complete, if it
  - contains all possibilities how to satisfy resp. falsify the external atom.

Formally:

A positive resp. negative family of support sets \( S_e \) with \( \sigma \in \{ T, F \} \) for external atom \( e \) is a set of positive resp. negative support sets of \( e \); \( S_e \) is complete if for each assignment \( A \) with \( A \ |= e \) resp. \( A \not|= e \) there is an \( S_e \in S_e \) s.t.

\[
\begin{align*}
A & \subseteq S^+_e \\
A \cap S^-_e & = 0.
\end{align*}
\]

3. Inlining of External Atoms – Our Encoding

A positive external atom \( e \) in a program \( P \) with a complete family of positive support sets \( S_e \) is inlined as follows (negative ones are handled similarly):

\[
P_X = \{ x_e \iff S^+_e \cup \{ \bar{a}, \bar{a} \iff x_e \} \} \cup S_X \cup S_e
\]

where \( \bar{a} \) is a new atom for each \( a \) and \( x_e \) are new atoms for \( e \) and \( P_{\bar{a} \iff x_e} = \bigcup_{e \in e} P_{e \iff x_e} \) where \( r_{e \iff x_e} \) denotes \( r \) with \( e \) replaced by \( x_e \).

5. Implementation and Experiments

We implemented our novel inlining approach in the DLVHEX solver and compared it to two previous evaluation approaches for HEX-programs:

- traditional: Respect external atoms in the core algorithms.
- sup.sets: Use support sets only for external atom verification.

We considered several benchmark problems, including:

1. House problem (abstraction of configuration problems):

<table>
<thead>
<tr>
<th>( n )</th>
<th>all answer sets</th>
<th>traditional</th>
<th>sup.sets</th>
<th>inlining</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>251.68 (81)</td>
<td>22.21 (2)</td>
<td>31.9 (0)</td>
<td>1.53 (0)</td>
</tr>
<tr>
<td>8</td>
<td>266.22 (85)</td>
<td>15.4 (10)</td>
<td>22.22 (2)</td>
<td>3.10 (0)</td>
</tr>
<tr>
<td>9</td>
<td>272.70 (85)</td>
<td>16.7 (12)</td>
<td>6.13 (10)</td>
<td>2.33 (2)</td>
</tr>
<tr>
<td>10</td>
<td>278.26 (85)</td>
<td>16.7 (12)</td>
<td>76.7 (12)</td>
<td>1.21 (0)</td>
</tr>
<tr>
<td>11</td>
<td>292.03 (100)</td>
<td>16.7 (12)</td>
<td>127.1 (18)</td>
<td>1.97 (0)</td>
</tr>
</tbody>
</table>

2. DL-programs (integration of ASP with description logics):

<table>
<thead>
<tr>
<th>( n )</th>
<th>all answer sets</th>
<th>traditional</th>
<th>sup.sets</th>
<th>inlining</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>1.08 (0)</td>
<td>0.34 (0)</td>
<td>0.31 (0)</td>
<td>0.31 (0)</td>
</tr>
<tr>
<td>30</td>
<td>27.73 (3)</td>
<td>0.34 (0)</td>
<td>0.34 (0)</td>
<td>0.34 (0)</td>
</tr>
<tr>
<td>40</td>
<td>145.06 (35)</td>
<td>0.48 (0)</td>
<td>64.73 (14)</td>
<td>0.40 (0)</td>
</tr>
<tr>
<td>50</td>
<td>249.78 (76)</td>
<td>0.48 (0)</td>
<td>213.45 (60)</td>
<td>0.47 (0)</td>
</tr>
<tr>
<td>60</td>
<td>265.73 (80)</td>
<td>0.57 (0)</td>
<td>296.61 (47)</td>
<td>0.70 (0)</td>
</tr>
<tr>
<td>70</td>
<td>298.13 (99)</td>
<td>0.72 (0)</td>
<td>297.17 (99)</td>
<td>0.72 (0)</td>
</tr>
</tbody>
</table>

6. Conclusion and Outlook

Main results:

- Novel evaluation algorithm for HEX-programs and an implementation.
- Experiments show a significant (up to exponential) speedup.

Future work:

- Refinements and optimizations of the rewriting.
- Heuristics for deciding when to rewrite.

8. References