and (iii) evaluation with *trans-unit (tu-)propagation*. Our hypothesis is that (i) suffers a grounding and (ii) suffers (also) a solving bottleneck, whereas (iii) outperforms the other two approaches as it can restrict the grounding and propagate throughout the whole program at the same time. In the tables we show the overall runtime, grounding and solving time, averaged over all instances of the respective size; importantly, other computations besides grounding and solving (e.g. preprocessing and method-specific tasks such as splitting for (ii) and inconsistency analysis for (iii)) can cause the overall runtime to be higher than the sum of grounding and solving time. Numbers in parantheses show the number of timeout instances. The encodings of all presented benchmarks can be found in the official benchmark suite of the system.[5]

Configuration Problem. Consider the following configuration problem. We assemble a server cluster consisting of various components. For a given component selection the cluster has different properties such as its efficiency, power consumption, etc. Properties may depend not only on individual components but also on their interplay, and this dependency can be nonmonotonic (e.g. the selection of an additional component might make it lose the property of low energy consumption). We want the cluster to have certain properties.

In order to capture also similar configuration problems (such as Example 5), we use a more abstract formalization as a quadruple (D, P, m, C), where D is a *domain*, P is a set of *properties*, m is a function which associates each selection $S \subseteq D$ with a set of properties $m(S) \subseteq P$, and C is a set of constraints of form $C_i = (C_i^+, C_i^-)$, where $C_i^+ \subseteq 2^P$ and $C_i^- \subseteq 2^P$ define sets properties which must resp. must not be simultaneously given; a selection $S \subseteq D$ is a *solution* if for all $(C_i^+, C_i^-) \in C$ we have $C_i^+ \not\subseteq m(S)$ or $m(S) \cap C_i^- \neq \emptyset$.

For the experiment we consider for each size n ten randomly generated instances with n domain elements, $\lfloor \frac{n}{5} + 1 \rfloor$ properties, a random function m, and a random number of constraints, such that their count has an expected value of n.

The results are shown in Table 1. One can observe that for all shown instance sizes, the splitting approach is the slowest, monolithic evaluation is the second-slowest, and tu-propagation is the fastest configuration. A look at the grounding and solving times shows that for the monolithic approach, the main source of computation costs is grounding, while for splitting it is the solving phase (although also the grounding costs are high due to repetitive instantiation of program components). In contrast, with tu-propagation both grounding and solving is fast, while most of the computation time is spent on other computations (most importantly, the inconsistency analysis). The observations are in line with our hypotheses.

Diagnosis Problem. Next, we consider the diagnosis problem $\langle \mathcal{O}_d, \mathcal{O}_p, \mathcal{H}, \mathcal{C}, P \rangle$, where sets $\mathcal{O}_d$ and $\mathcal{O}_p$ are definite resp. potential observations, $\mathcal{H}$ is a set of hypotheses, $\mathcal{C}$ is a set of constraints over the hypotheses, and P is a logic program which defines the observations which follow from given hypotheses. Each constraint $C \in \mathcal{C}$ forbids certain combinations of hypotheses. A solution consists of a set $S_\mathcal{H} \subseteq \mathcal{H}$ of hypotheses and a set of potential observations $S_O \subseteq \mathcal{O}_d$ such that (i) all answer sets of $P \cup H$, which contain all of $S_O \cup \mathcal{O}_d$, contain also $S_\mathcal{H}$, and (ii) $C \not\subseteq S_\mathcal{H}$ for all $C \in \mathcal{C}$. Informally, $S_\mathcal{H}$ are necessary hypotheses to explain the observations.

[5] https://github.com/hexhex/core/tree/master/benchmarks

size	monolithic			splitting			tu-propagation		
	total	ground	solve	total	ground	solve	total	ground	solve
6	0.14 (0)	0.01	0.02	0.18 (0)	0.03	0.04	0.15 (0)	<0.005	0.01
8	0.20 (0)	0.05	0.04	0.49 (0)	0.15	0.20	0.19 (0)	<0.005	0.02
10	0.45 (0)	0.22	0.10	2.17 (0)	0.81	1.11	0.33 (0)	<0.005	0.05
12	1.51 (0)	0.90	0.44	9.50 (0)	3.59	5.17	0.86 (0)	0.01	0.10
14	4.84 (0)	3.68	0.82	44.23 (0)	16.58	24.68	1.48 (0)	0.02	0.20
16	19.52 (0)	16.55	2.08	217.46 (0)	86.17	119.93	3.80 (0)	0.07	0.57
18	143.09 (3)	68.89	10.89	300.00 (10)	122.23	165.44	44.76 (1)	0.43	6.29
20	300.00 (10)	300.00	n/a	300.00 (10)	126.40	162.73	37.44 (0)	0.36	3.44
22	300.00 (10)	300.00	n/a	300.00 (10)	122.41	165.68	224.83 (6)	1.03	11.74

Table 1: Configuration Problem

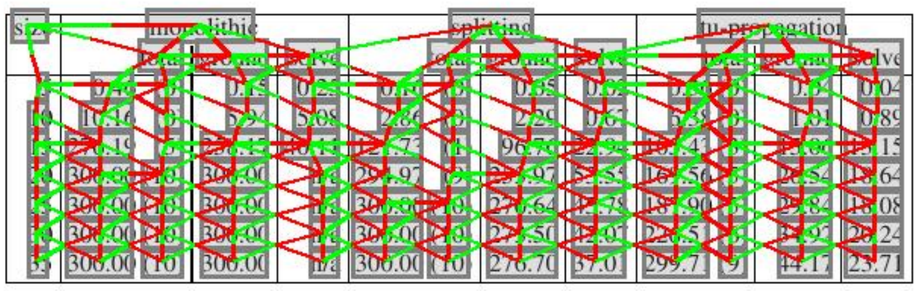

Table 2: Diagnosis Problem

The results are shown in Table 2. Unlike for the previous benchmark, monolithic is slower than splitting. This is because the evaluation of the external source (corresponding to evaluating P) is much more expensive now; since monolithic grounding requires exponentially many evaluations, while during solving this is not necessarily the case, the evaluation costs have a larger impact to monolithic than to splitting. However, as before tu-propagation is clearly the fastest configuration. Again, the observations are in line with our hypotheses.

Analysis of Best-Case Potential. Finally, in order to analyze the potential of our approach in the best case, we use a synthetic program. Our program of size n is as follows:

$$P = \{ dom(1..n).\ in(X) \vee out(X) \leftarrow dom(X).$$
$$someIn \leftarrow in(X).$$
$$r(X) \leftarrow \&diff[dom, out](X).\ \leftarrow r(X), someIn \}$$

It uses a domain of size n and guesses a subset thereof. It then uses an external atom to compute the complement set. The final constraint encodes that the guessed set and the complement must not be nonempty at the same time, i.e., there are only two valid guesses: either all elements are in or all are out.

The results are shown in Table 3. While splitting separates the rules in the third line from the others and must handle each guess independently, the monolithic approach must evaluate the external atom under all possible extensions of out. Both