APPENDIX

A. DETAILED EVALUATION

This appendix provides a detailed description of the experimental study we conducted in the ICSE14 paper entitled "Lifting Model Transformations to Product Lines". The next subsection A.1 is meant to replace subsection 6.1 in the paper.

A.1 Methodology

Lifted rule application begins with finding a rule matching site which is a well-studied subgraph isomorphism problem [18]. For this step, our approach performs as well as the traditional graph transformations. Afterwards, the application algorithm in Fig. 6 is applied. Steps 4-9 of the algorithm can be completed in time linear in the size of the matching site K. We thus focus our examination to Steps 3 and 10 since they require solving the satisfiability problem, which is NP-complete. The former entails checking whether $\Phi_p \wedge \Phi_{apply}$ (Def. 8) is SAT and the latter – if $\Phi_p \wedge \Phi_d \wedge \neg\Phi_{apply}$ is UNSAT. In this section, we refer to these formulas as Φ_1 and Φ_2, respectively. Since Φ_1 and Φ_2 are similar in structure, we use the same experimental design whereby we generate random but realistic inputs to a SAT solver to measure the time required to check SAT and UNSAT, respectively. To generate realistic inputs, we simulated the execution of the algorithm in Fig. 6, replacing matching with random element selection. We varied input generation using two experimental variables: (a) the feature model, and (b) the transformation rule. In addition, based on pilot runs and the case study described in Sec. 5, we calibrated random input generation with additional parameters to ensure that the generated formulas correspond to realistic scenarios. We describe these below.

Varying the Feature Model. Each element in Φ_1 and Φ_2 is represented by its presence conditions which are expressed over the set of features. Moreover, as described in Sec. 3.1, the feature model of a product line P can be encoded in the propositional formula Φ_p expressed over the set of features [15]; Φ_p is a subformula of Φ_1 and Φ_2. Thus, the first experimental variable is the choice of a feature model. To get realistic values of this parameter, we used the collection of real feature models available in the S.P.L.O.T repository [30] . At the time of experimentation, the repository contained 359 real feature models, ranging from 9 to 290 features, with an average of 26 features each.

Varying the Transformation Rule. The subformula Φ_{apply} of Φ_1 and Φ_2 in Def. 8 gets more complex for larger sizes of the rule's LHS and NACs. Thus, our second experimental variable is the choice of the rule. To vary it, we use seven real graph transformation rules chosen from the literature and shown in Table 3. We specifically chose those that represent variety of transformation use cases (translation, refactoring, refinement, etc.) and have LHS, RHS and NACs of different sizes, ranging from 0 to 30.

Generating inputs. In order to generate realistic inputs, we simulate the rule application algorithm in Fig. 6. At each simulation step r, we produce a formula $\Phi_{apply}(r)$ that approximates the formula Φ_{apply} in Φ_1 and Φ_2. We resorted to simulating the algorithm due to the lack of readily available real examples of product line domain models. We simulated the matching and transformation steps of the algorithm by generating expressions that represent elements with randomly generated presence conditions.

The simulated algorithm iterates to represent subsequent rule applications. We distinguish two classes of model elements: First, those with simple presence conditions, symbolized as e_N, e_D and e_C. These represent elements of the untransformed model and are generated with a simple random presence condition which is either True or a single feature variable. Second, those with more complex presence conditions, symbolized as e_A. These are elements that are added over time via rule application and their presence condition is $\Phi_{apply}(r)$, where r is the iteration counter.

Given an initial model size s_0, at the first iteration step $r = 1$, we first construct a new $\Phi_{apply}(1)$ as shown in step 2 of the algorithm in Fig. 6. Its subformulas $\phi^{and}_{\mathbf{N}}(1)$, $\phi^{and}_{\mathbf{C}}(1)$, and $\phi^{and}_{\mathbf{D}}(1)$ are constructed using n e_N elements, c e_C elements, and d e_D elements. The constants n, c, and d represent the size of the respective parts of the rule being applied, as shown in Table 3. We then simulate the addition step of algorithm (step 5 in Fig. 6), by generating a e_A elements with presence condition $\Phi_{apply}(1)$ and calculating the size $s_1 = s_0 + a$. We keep track of all generated e_A elements in a global data structure B which also associates each e_A with a "chaining counter", which is initialized to a common value. Before the next iteration step we calculate the "selection probability" $p_1 = |B|/s_1$, where $|B|$ is the number of e_A elements in B.

In every subsequent step $r+1$, we repeat the above process modified as follows: Each element in the new $\Phi_{apply}(r+1)$ has probability p_r to be taken from the global data structure B and probability $1-p_r$ to be freshly generated with a simple presence condition. If an element e_A is reused from B, its chaining counter is decremented. Once an e_A element's chaining counter reaches zero it is removed from B. Next, a new e_A elements are added to B, the model size is increased to $s_{r+1} = s_r + a$, and the new selection probability is computed to $p_{r+1} = |B|/s_{r+1}$.

The process is iterated for a preset number of steps to simulate the growing complexity of presence conditions resulting from repeated rule application. Because of the way the selection probability is computed at each step, there is an increasing likelihood that existing e_A elements are reused from B. The decay of the chaining counter also ensures that presence conditions do not become arbitrarily large, thus approximating the behavior that we observed in the case study and pilot runs.

Calibrating input generation. The generation process described above requires calibration of a few additional experimental parameters. Rather than considering these as independent variables, we chose to fix their values based on pilot runs and observations of the case study in order to avoid the combinatoric explosion of possible experimental configurations. These parameters are:

a) The size s_0 of the original domain model has an impact on the calculation of the selection probability. We chose to start with a fixed size of 100 elements to simulate models that are reasonably large in size.

b) Whenever we generate a fresh random element with simple presence conditions, we must decide whether its presence condition will be True or a single feature variable. We fixed these probabilities to 0.4 and 0.6 respectively.

c) A real model transformation would stop once there would be no more application sites for which the applicability condition would hold. For our simulated generation of inputs, we set a maximum number of iterations after which we cut off execution. Based on our observations in the case study we fixed this parameter to 500.

d) In real transformations, an element would be likely to participate in more than one rule applications, especially in the case of layered rule application. To simulate this we used the global data structure B and defined the chaining counter for each element in B. Based on our pilot observations, we initialized it for each new added element to 4.

e) In pilot runs, we observed that usually NACs would tend to only match towards the end of a transformation chain, ultimately causing the transformation to stop. We thus made it so that the generated $\Phi_{apply}(r)$ would only contain the subformula $\phi^{and}_{\mathbf{N}}$ with a fixed probability 0.5, similarly to the behavior that we observed in pilot runs.