— Sprint 5 —

Deadline: 19th of June

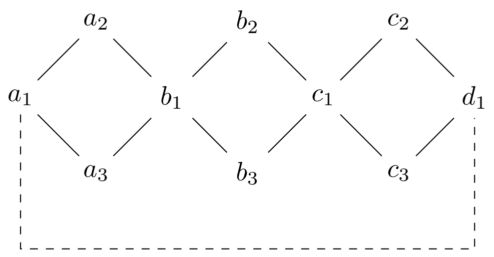
We have just completed a major refactoring of WolVeriNe. During this work our new DeepLearning framework has identified some interesting patterns that our encoders produce that might be of interest to you. Our guess is that you could pretty easily make your solver much faster on these particular examples.

Diamonds

One of our encoders commonly produces what we call a *diamond structure*. The formulas roughly look like this:

$$\begin{aligned}\varphi_{a,b} &= ((a_1 = a_2) \land (a_2 = b_1)) \lor ((a_1 = a_3) \land (a_3 = b_1)) \\ \varphi &= \varphi_{a,b} \land \varphi_{b,c} \land \varphi_{c,d} \land (a_1 \neq d_1)\end{aligned}$$

Our intern came up with this visual representation (where the dashed line is a disequality):



Obviously these are always unsatisfiable, but as the engineer responsible for this particular encoder is on a long vacation right now, we'd like you to take care of it. Apparently the SAT solver tries all possible ways to get from a_1 to d_1 (via a_2, b_2, c_2, a_2, b_2, c_3, a_2, b_3, c_2, a_2, b_3, c_3, ...) which are exponentially many (at least our intern claims that). Maybe you can teach the SAT solver to do it with a single theory call?

Hidden equalities

Another encoder generates a somewhat similar, but way more confusing pattern. We essentially observe that two variables must be equal but there is no direct equality and it can also not be derived right from the start. This is somewhat complicated to explain (in particular as I do not properly understand it myself). The examples all look somewhat like this:

$$\varphi = \bigvee_k \varphi_k \text{ with } \varphi_k \Rightarrow a = b \text{ for all k}$$

Essentially we have a big disjunction, but every disjunct by itself somehow implies that $a = b$. We are still investigating why we build these formulas in the first place, but we have no idea yet. If we could simply add $a = b$ right from the start, we might be a lot faster...

Dynamic lemma generation

One of our engineers had an interesting snapchat (or tinder?) conversation about SMT solving recently. He was told that modern solvers employ a technique they call *lemma generation*. Essentially the theory solver tries to find helpful information – that is not directly related to the current theory call – and provides this information to the SAT solver.

In your case, you could try to add information about constraints that the SAT solver has not yet decided upon. If you consider the formula

$$\varphi = (a = b) \wedge (b = c) \wedge ((a \neq c) \vee (b = d))$$

and your theory solver is called with $\{a = b, b = c\}$, maybe you can detect that $a = c$ and tell the SAT solver that $((a = b) \wedge (b = c)) \Rightarrow (a = c)$? We already talked to the SMT-RAT guys, and apparently there is a method `addLemma()` to do such stuff.

Flattening

When calling your solver, we noticed the command-line option `--disable-uf-flattening`. One of our applications yields formulas with nested function calls, but we never noticed because the SMT-RAT parser simply removed them using a technique called *flattening*. It basically replaced $f(f(a)) = b$ by $f(x) = b \wedge b = f(a)$ (with a fresh variable b).

Can you do any better if you disable flattening?