The Backend Provers

SMT Solvers

The backend provers that are the most useful are a class called SMT solvers. TLAPS can use several different SMT solver backends; the default one used by the Toolbox is called *SMT*. By default, *SMT* is the first backend prover TLAPS calls. The SMT solvers are much, much better than any other backend prover at arithmetical reasoning. They will all prove any valid formula about integers you are likely ever to write that contains only numbers, $+$, $-$, $<$, $\leq$, $>$ $\geq$, $=$ and the operators of propositional logic—for example:

$$(m \geq 0) \land (m \leq p) \land (n \geq q) \land (n \leq r) \land (p + 1 \leq q) \Rightarrow (m \neq n)$$

for integers m, n, p, q, and r. They're much better than you are at deciding if such a formula is true. They also know that $m \in p \,..\, q$ is equivalent to $(p \leq m) \land (m \leq q)$ if m, p, and q are integers, so they're also good at reasoning about integer intervals. However, they aren't very good at formulas involving $*$, $\%$, and $\div$.

The default SMT solver is called *CVC*3, so by default *SMT* and *CVC*3 are synonymous. It is included in the TLAPS distribution. Other SMT solvers that you can download yourself and use as TLAPS backend provers are *Z*3, *Yices*, and *veriT*.

Most of the time, a proof obligation will be provable by one SMT solver iff it is proved by all of them. However, they do vary in their ability to reason about multiplication. Yices and Z3 prove

$$\forall m, n \in \mathit{Int} \,:\, m * (n - 1) = m * n - m$$

but CVC3 does not. Only Z3 proves

$$\forall i \in \mathit{Int}, j \in \mathit{Nat} \setminus \{0\} \,:\, i\%j \in 0 \,..\, (j - 1)$$

None of the SMT backends have built-in knowledge about $\div$.

As this indicates, I currently find Z3 to work a little more often than the others. It also seems to be the fastest. It is not the default *SMT* backend because it is not free for commercial users, so it can't be distributed with TLAPS. However, you can download it yourself and make it the default. The default SMT solver can be changed from the Toolbox's preference window reached by File/Preferences/TLA+ Preferences/TLAPS/Additional Preferences. Instructions for how to do that are on this web page.

Zenon

Zenon is the second backend prover called by default. It knows nothing about numbers, but it knows about the built-in TLA$^+$ operators and is good at general mathematical reasoning.

Isa

Isa is short for Isabelle, the third backend prover that is called by default. It can do most of what Zenon does and some things Zenon can't do, but it's slower. I find it better than Zenon at reasoning about functions and records. It's also better than Zenon for reasoning about higher-order operators—more precisely, reasoning that requires substituting operators in ASSUME/PROVE facts in which the assumptions declare NEW operators. Examples of such facts are the library's rules for proofs by induction.

Isabelle knows a little bit about numbers, but isn't too good with them. It can prove $2 + 2 = 4$ but times out trying to prove $20 + 20 = 40$.

Isabelle uses methods, which you can specify with the *IsaM* operator. The standard method is called *auto*, and the BY "fact" *Isa* is synonymous with *IsaM*("auto"). Here are the other available methods, which can occasionally help.

blast This method is somewhat better than *auto* for reasoning about higher-order operators.

force This method is similar to *auto* but more aggressively tries instantiating quantifiers.

Experts in mechanical verification may have some idea what the following methods are good for. The rest of us can try them for fun, or when we get desperate.

fast A resolution-based prover.

simp Is the part of *auto* that does rewriting. It is superseded by *auto* except in borderline cases.

They have two variants: The *clarsimp* method combines *simp* with some basic proof rules such as reducing $A \Rightarrow B$ to ASSUME A PROVE B. The *fastsimp* method combines *simp* with *fast*, which is very similar to *auto*; it is unlikely to work if *auto* doesn't.

PTL

PTL stands for Propositional Temporal Logic. It is synonymous with *LS*4, which is a backend prover that can prove temporal formulas that do not contain quantification over temporal formulas. **As I type this, it has not yet been released.**

Giving a Prover More Time

When a backend prover fails by timing out, giving it more time seldom helps. However, it occasionally does. You can tell the *SMT* to timeout only after 30 seconds with the BY pseudo-fact *SMTT*(30). Similarly, *ZenonT*(45) calls

Zenon with a timeout of 45 seconds, and so on for the other backend provers. By default, the backend provers all time out after 5 seconds except for Isabelle, which times out after 30 seconds.

Increasing timeouts is most likely to help if you're using a slow machine. In that case, you can use the `--stretch` prover option to increase all timeouts by a factor. **Such advanced options and how to use them will appear on this web page when someone gets around to putting them there.**

CLOSE