Slot Machines: an approach to the Strategy Challenge in SMT solving (presentation only)

Stéphane Graham-Lengrand

Slot Machines: an approach to the Strategy Challenge in SMT solving (presentation only)

Stéphane Graham-Lengrand

SRI International - CNRS - École Polytechnique

Abstract In this short introduction we briefly describe the relevance of Psyche's system description [GL13], attached, to the Strategy Challenge set by L. de Moura and G. O. Passmore for SMT solving.

In [dMP13], L. de Moura and G. O. Passmore presented the Strategy Challenge to the SMT-community: "To build theoretical and practical tools allowing users to exert strategic control over core heuristic aspects of high-performance SMT solvers." They advocated for the adaptation of ideas of strategy prevalent in the Argonne and LCF theorem proving paradigms.

The same year, at the *Tableaux* conference, a system description was published of our *Proof-Search factorY for Collaborative HEuristics*, Psyche, entitled "A proof-search engine based on sequent calculus with an LCF-style architecture" [GL13]. Our highly modular system, designed for either interactive or automated theorem proving, provides a platform where users can experiment various techniques and strategies, to be implemented as *plugins* via an API with Psyche's *kernel*. Moreover, Psyche's framework features the possibility to call theory-specific decision procedures, so that the system can be used for SMT-solving: Indeed we showed in [FGLM13] that (an elementary version of) the $\mathsf{DPLL}(\mathcal{T})$ scheme is bi-similar (i.e. isomorphic) to a simple proof-search strategy scheme, an instance of which we implemented as one of Psyche's plugins.

In [dMP13] is emphasized the key role that is played in SMT-solving by what is not specified by the $\mathsf{DPLL}(\mathcal{T})$ scheme, such as heuristics. The ability to easily tune these and give users control over them, is a desirable feature to increase a solver's versatility. The Strategy Challenge suggests that this ambition deserves more than behaviour-controlling parameters to be set by the user; it may also require a more modular organisation of a solver's code, for instance to enhance the programmability of its heuristics. This is what we did with Psyche's API.

Of course, empowering users should not jeopardise the correctness of a solver's answers. But a modular architecture can in fact *help*, rather than jeopardise, the trust in a solver. And this is an extra motivation for the Strategy Challenge which adds on to [dMP13]. If the code that defines and controls the heuristics is separated from the code that implements only the high-level, non-deterministic, rule-based procedures such as $\mathsf{DPLL}(\mathcal{T})$, then it is possible, using the right software abstractions, that only the latter kind of code needs to be trusted (or, even better, verified) in order to trust the whole solver. Regardless of the user's own fiddling of heuristics.

This is where the LCF-inspired ideas come in: Psyche gives users full control in that they can write their own source code in the form of a plugin, as long as it implements a predefined interface. That interface requires their main function to output a value in the type of answers:

```
type answer = Provable of statement*proof | NotProvable of statement
```

The only limitation, which guarantees correctness, is that this type of answers is *private* to Psyche's kernel: in order to inhabit it, the plugin cannot use its constructors and has no choice but to use the kernel's API, hoping that, from this interaction, a value in this type arises that can be given as the final output.

The approach we took in Psyche is to model this interaction as a *slot machine*, as each call to the kernel finishes with a value of the following recursive type which the plugin can exploit:

```
type output = Jackpot of answer
            | InsertCoin of info*(coin -> output)
```

If the plugin is lucky, the kernel provides the answer the plugin is waiting for, otherwise it provides, besides some useful information, a function of type `coin -> output` which the plugin can use to 'put a new coin in the slot machine' instructing the kernel which rule to apply next (say in the DPLL($\mathcal{T}$) scheme). And for this the plugin can summon all imaginable heuristics without jeopardising the correctness of an eventual answer.

This 'slot machine' form of kernel/plugin interaction, using a private type of answers to guarantee correctness, can be set up for any *inference system* in the sense of of [GRS04,Sha05], where the heuristics to be programmed as plugins are called *reduction operations*. Inference systems are used in particular as the basis of a unifying approach to describe theory combination techniques, so a slot machine implementation of them would also allow users to control how theories are combined in SMT-solving. This is one of Psyche's next developments.

The slot machine architecture used in Psyche gives users a control over strategies that is powerful, in that they have access to the full expressivity of the programming language to implement plugins, and fine-grained, in that they can decide up to the details of e.g. how and when unit propagation is performed. This "small step" strategic control over reasoning engines was left as future work in [dMP13]. Of course, the expressivity of a full programming language and the control of the small steps might be "too much" for the user who may not want (or know how) to harness so much power. In that case, one can provid intermediate modules between the kernel and what the user actually manipulates, for instance by nesting slot machines within slot machines: the deepest one interacts directly with the kernel on all heuristical aspects, while the top-level one interacts with the end-user on the much coarser heuristical aspects that [dMP13] mostly discusses. If need be, one may also offer a limited tactic-programming language for the end-user to specify how to insert coins into the slot machine, without exposing him to the programming language in which the prover is written.

Also, a key aspect of [dMP13] is the preservation of efficiency, especially for the small-step strategic control which may rely on the use of clever data-structures. More work needs to be done with Psyche to show that the slot machine architecture does not intrinsically yield unreasonable performance drops.

References

dMP13. L. M. de Moura and G. O. Passmore. The strategy challenge in SMT solving. In M. P. Bonacina and M. E. Stickel, editors, *Automated Reasoning and Mathematics - Essays in Memory of William W. McCune*, volume 7788 of *LNCS*, pages 15–44. Springer-Verlag, 2013.

FGLM13. M. Farooque, S. Graham-Lengrand, and A. Mahboubi. A bisimulation between DPLL(T) and a proof-search strategy for the focused sequent calculus. In A. Momigliano, B. Pientka, and R. Pollack, editors, *Proc. of the 2013 Int. Work. on Logical Frameworks and Meta-Languages: Theory and Practice (LFMTP 2013)*. ACM Press, 2013.

GL13. S. Graham-Lengrand. Psyche: a proof-search engine based on sequent calculus with an LCF-style architecture. In D. Galmiche and D. Larchey-Wendling, editors, *Proc. of the 22nd Int. Conf. on Automated Reasoning with Analytic Tableaux and Related Methods (Tableaux'13)*, volume 8123 of *LNCS*, pages 149–156. Springer-Verlag, 2013.

GRS04. H. Ganzinger, H. RueB, and N. Shankar. Modularity and refinement in inference systems. Technical Report SRI-CSL-04-02, SRI, 2004.

Sha05. N. Shankar. Inference systems for logical algorithms. In *Proc. of the 25th Int. Conf. on Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'05)*, volume 3821 of *LNCS*, pages 60–78. Springer-Verlag, 2005.