Automatic Generation of Concurrent Provers

Raul H. C. Lopes

Departamento de Informática – UFES, Caixa Postal 01-9011/290670 – Brazil
raulh@inf.ufes.br

1. Introduction

A framework is outlined that combines automatic generation of proof search strategies for theorem provers with concurrent proof search. The framework consists of two programs: $\mathcal{P}_2$, that implements an algorithm that generates prioritized logic programs, representing proof search strategies; and $\mathcal{P}_2$-*frame*, a proof search engine that can use strategies generated by $\mathcal{P}_2$ to drive concurrent provers.

$\mathcal{P}_2$ has been applied to generate provers for Intuitionistic Propositional Calculus(**IPC**), modal logics T, S4, and S5, and for classical second-order logic (see, for example, [4].) Figure 1 shows a few difficult theorems proved by a concurrent prover, using a proof search strategy generated by $\mathcal{P}_2$. Particularly interesting are the automatic proofs obtained for the problems 1 to 4 (taken from [1]) and for Cantor's theorem that the power set of a set S is larger than S (problem 15.)

1. $\vdash \forall U \forall V((U < 0) \supset \neg(U = abs(V))) \supset$
   $\exists A(\forall Y(\neg A(abs(Y))) \wedge A(-(2)))$
2. $\vdash \forall X(\exists U(X = (2 * U)) \leftrightarrow \neg\exists V((X + 1) = (2 * V))) \supset$
   $\exists A \forall X(A(X) \leftrightarrow \neg A(X + 1))$
3. $\vdash \forall P \forall Y((\forall A((A(0) \wedge \forall X(A(X) \supset A(X + 1))) \supset A(Y)) \wedge$
   $(P(0) \wedge \forall X(P(X) \supset P(X + 1)))) \supset P(Y))$
4. $\forall X \forall Y \forall Z((F_2(X, Y) = F_2(Z, Z)) \supset (X = Y)) \vdash$
   $\forall P \forall U \forall V((\forall A((A(F_2(0, 0)) \wedge$
   $\forall X \forall Y(A(F_2(X, Y)) \supset A(F_2(F_1(X), F_1(Y))))) \supset A(F_2(U, V))) \wedge\ P(U)) \supset P(V))$
5. $\vdash \neg\exists G \forall F \exists J(G(J) = F)$

Fig. 1. A problem set for $\mathcal{P}_2$

2. Concurrent proof strategies

$\mathcal{P}_2$ can generate proof search strategies, that are composed of assertions (called *methods*) about the uses of the inference rules of a given logic taken from examples of proofs. It assumes that proofs can be performed in a goal-oriented fashion, and that they can be decomposed in *proof steps*, containing a *goal* and a designation of an inference rule applied to it. Its main components are: a set of randomized algorithms for generating *methods* from proof steps, and algorithms for ordering *methods*, for matching methods with proof goals, and for establishing redundancy of *methods*.

A method is generated from a step by *schematization* of its goal into a meta-goal (uniform replacement of variables by meta-variables), and from other methods by *lifting* (random replacement of one meta-variable by a new meta-variable), and *thinning* (random dropping of a formula of the meta-goal.) A method is *confirmed* by a step when its *meta-goal* unifies with the step's goal and they both designate the same inference rule. A method is *contradicted* by a step when its *meta-goal* unifies with the step's goal and they designate different inferences rule. Methods with negative conditions can be automatically generated and they restrict the range of steps that one method can match (see [4]).

Methods are ordered with respect to the set of input steps. A method is given highest priority if it has no contradiction in the given examples. There can be several methods in that condition and they concurrently compete to be applied in a proof. Immediately after them are assigned to highest priority methods whose contradictions are all confirmed by methods in the preceding level. This idea is used by $\mathcal{P}_2$ to define partial order on methods. A *synchronous single pool* parallel branch-and-bound algorithm ([3]) is used to drive the proof search: methods with the same priority are scheduled for concurrent matches with a given goal. A successful match produces a new branch in the search tree.

3. Conclusion

The $\mathcal{P}_2$-*frame* has two strong points: it can generate proof search strategies for arbitrary logics; and the proof search strategies it produces can drive proofs either with traditional depth-first or breadth-first search procedure (as described in [4]), or with parallel algorithms, based, for example, on branch-and-bound. This last option is particularly attractive in higher-order proving, where the the undecidability of unification can lead provers into endless loops. Concurrent matches were important in several steps of the proofs obtained for theorems $1, 2$, and 4 of fig. 1.

References

1. W.W. Bledsoe and Guohui Feng, *SET-VAR*, Journal of Automated Reasoning **11** (1993), 293–314.
2. Harald Ganzinger (ed.), *Proceedings of the 16th International Conference on Automated Deduction*, Springer-Verlag, 1999, LNAI, 1632.
3. Bernard Gendron and Teodor Gabriel Crainic, *Parallel branch-and-bound algorithms: survey and synthesis*, Operations Research **42** (1994), no. 6, 1042–1066.
4. Raul H.C. Lopes, *Automatic generation of proof search strategies for second-order logic*, In Ganzinger [2], LNAI, 1632, pp. 414–428.