Variational Satisfiability Solving

Efficiently Solving Lots of Related SAT Problems

Empirical Software Engineering, 2022




Jeffrey M. Young

INPUT | OUTPUT

Paul Bittner

universität ulm

Eric Walkingshaw

Thomas Thüm

universität ulm

SAT solvers are used everywhere

SAT solvers are used everywhere

Bug finding

Finding Bugs in an Alpha Microprocessor Using Satisfiability Solvers

Per Bjesse[1], Tim Leonard[2], and Abdel Mokkedem[2]

[1] Chalmers University of Technology, Sweden
bjesse@cs.chalmers.se
[2] Compaq Computer Corporation, USA
{tim.leonard,abdel.mokkedem}@compaq.com

Abstract. We describe the techniques we have used to search for bugs in the memory subsystem of a next-generation Alpha microprocessor. Our

Finding Bugs Efficiently with a SAT Solver

Julian Dolby
IBM T.J. Watson Research Center
P. O. Box 704
Yorktown Heights, NY 10598
dolby@us.ibm.com

Mandana Vaziri
IBM T.J. Watson Research Center
P. O. Box 704
Yorktown Heights, NY 10598
mvaziri@us.ibm.com

Frank Tip
IBM T.J. Watson Research Center
P. O. Box 704
Yorktown Heights, NY 10598
ftip@us.ibm.com

ABSTRACT

We present an approach for checking code against rich specifications, based on existing work that consists of encoding the program in a relational logic and using a constraint

concrete witnesses for bugs. However, testing approaches may miss problems due to incomplete coverage. Static analysis techniques over-approximate all program behaviors, and is capable of proving the absence of an error. However, static

SAT solvers are used everywhere

Bug finding Model checking

Bounded Model Checking

Armin Biere[1] Alessandro Cimatti[2] Edmund M. Clarke[3]
Ofer Strichman[3] Yunshan Zhu[4] *

[1] Institute of Computer Systems, ETH Zurich, 8092 Zurich, Switzerland.
Email: biere@inf.ethz.ch
[2] Istituto per la Ricerca Scientifica e Technologica (IRST)
via Sommarive 18, 38055 Povo (TN), Italy. Email: cimatti@irst.itc.it
[3] Computer Science Department, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh,
PA 15213, USA. Email: {emc,ofer
[4] ATG, Synopsys, Inc., 700 East Middlefield Road, M
Email: yunshan@synops

Abstract. Symbolic model checking with Binar
has been successfully used in the last decade for
systems such as sequential circuits and protocols
beginning of the 90's, it has been integrated in th
several major hardware companies. The main bo
BDDs may grow exponentially, and hence the an
stricts the size of circuits that can be verified effic
a technique called Bounded Model Checking (BM
SAT solver rather than BDD manipulation techni
1999, BMC has been well received by the industr
rors in complex systems that can not be handled b
therefore widely perceived as a complementary te

Interpolation and SAT-based Model Checking

K. L. McMillan

Cadence Berkeley Labs

Abstract. We consider a fully SAT-based method of unbounded sym-
bolic model checking based on computing Craig interpolants. In bench-
mark studies using a set of large industrial circuit verification instances,
sed symbolic model
-based model check-
g temporal properties of

SAT-Based Model Checking Without Unrolling

Aaron R. Bradley

Dept. of Electrical, Computer & Energy Engineering
University of Colorado at Boulder
Boulder, CO 80309
bradleya@colorado.edu

Abstract. A new form of SAT-based symbolic model checking is de-
scribed. Instead of unrolling the transition relation, it incrementally gen-

SAT solvers are used everywhere

Bug finding Model checking Bioinformatics

SAT in Bioinformatics:
Making the Case with Haplotype Inferen

Inês Lynce[1] and João Marques-Silva[2]

[1] IST/INESC-ID, Technical University of Lisbon, Portugal
ines@sat.inesc-id.pt
[2] School of Electronics and Computer Science, University of Southampton
jpms@ecs.soton.ac.uk

Abstract. Mutation in DNA is the principal cause fo
human beings, and Single Nucleotide Polymorphisr
most common mutations. Hence, a fundamental tas
map of haplotypes (which identify SNPs) in the hum
sociated with this effort, a key computational probl
of haplotype data from genotype data, since in prac
rather than haplotype data is usually obtained. Rec

Lynx: A Programmatic SAT Solver for the RNA-folding Problem

Vijay Ganesh, Charles W. O'Donnell,
Mate Soos†, Srinivas Devadas, Martin C. Rinard and Armando Solar-Lezama

Massachusetts Institute of Technology, †Security Research Labs
{vganesh, cwo, devadas, rinard, asolar} @csail.mit.edu, †mate@srlabs.de

Abstract. This paper introduces Lynx, an incremental programmatic SAT solver that allows non-expert users to easily introduce domain-specific code into modern Conflict-driven Clause-learning (CDCL) SAT solvers, thus enabling users to control the behavior of the solver in ways
that enables
lowing user-
arch, and to
ile the power
operties of

Conference Paper

SAT-based protein design

December 2009 · IEEE/ACM International Conference on Computer-Aided Design, Digest of Technical Papers

Source · IEEE Xplore

Conference: Computer-Aided Design - Digest of Technical Papers, 2009. ICCAD 2009. IEEE/ACM International Conference on

Authors:

N. Ollikainen

Ellen Sentovich
12.79

C. Coelho

PDF

Request full-text PDF

To read the full-text of this research, you can request a copy directly from the authors.

SAT solvers are used everywhere

Bug finding

Model checking

Bioinformatics

Scheduling

Software product-lines

Program Synthesis

Program Verification

Planning

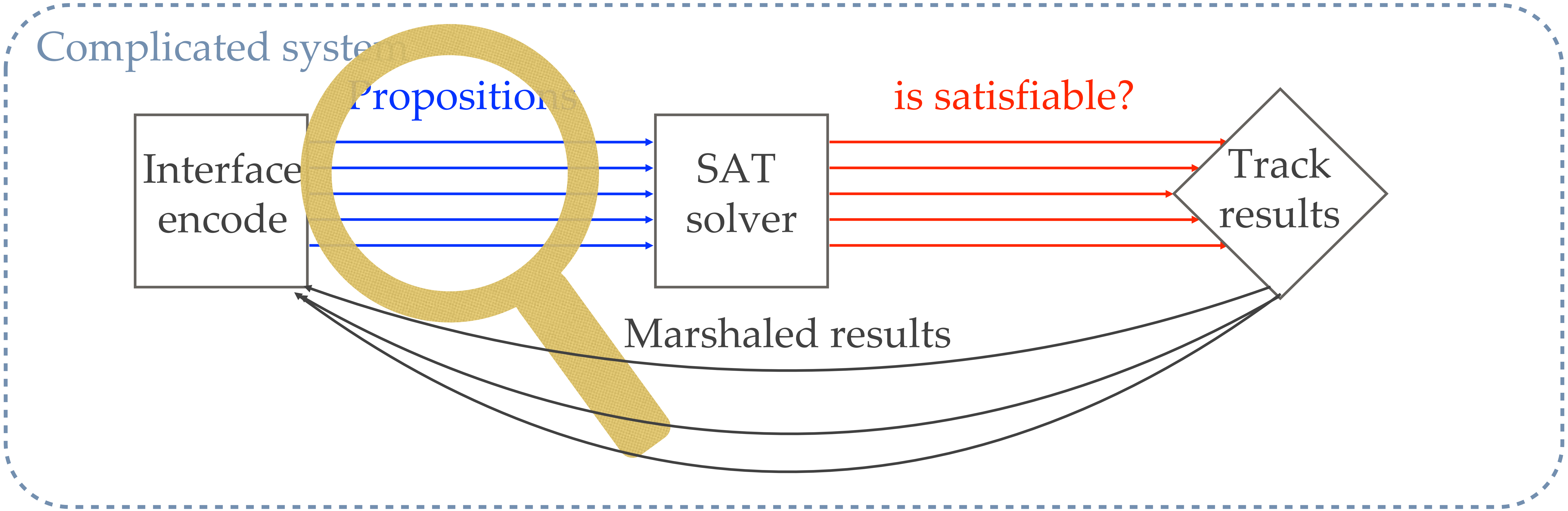
A satisfiability (SAT) solver is a tool that solves the Boolean satisfiability problem

Given a formula in propositional logic: $f \coloneqq (p \wedge q \Rightarrow r)$

Find an assignment such that *f* is *satisfied*:

$$p \rightarrow F$$
$$q \rightarrow T$$
$$r \rightarrow T$$

$$f = F \wedge T \Rightarrow T$$
$$f = F \Rightarrow T$$
$$f = T$$



This is great, however…

Propositions

$$\begin{cases} FM \land c_1 \land c_2 \land \cdots \land c_n \\ FM \land c_1 \land c_{10} \land \cdots \land c_n \\ FM \land c_{10} \land c_2 \land \cdots \land c_n \end{cases}$$

When we want to solve sets of related problems it is likely we are re-evaluating several clauses or terms!

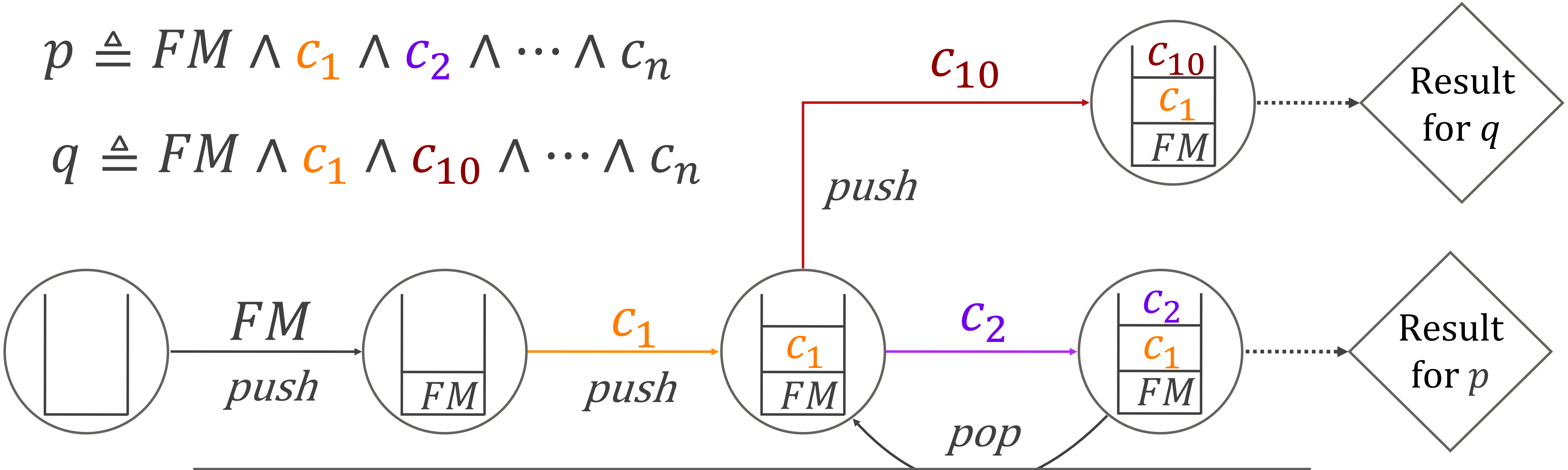
Incremental SAT: Add or remove clauses to the assertion stack

Push: Add a clause

Pop: Remove the last clause

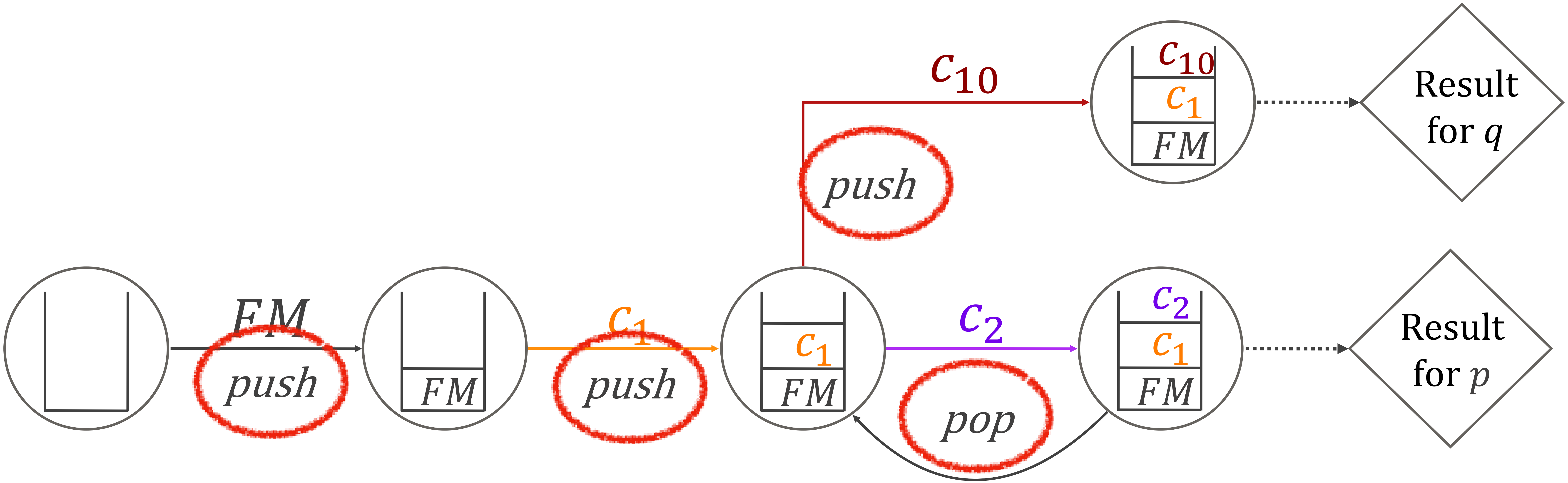
$$p \triangleq FM \wedge c_1 \wedge c_2 \wedge \cdots \wedge c_n$$

$$q \triangleq FM \wedge c_1 \wedge c_{10} \wedge \cdots \wedge c_n$$



Directly addresses redundancy and is efficient

Een and Sörensson, An Extensible SAT-Solver 2003



But the solver is not aware of what is shared!

Push/pop do not denote meaning to the user

Hand-written program for each domain!

What do we mean by variation?

CORE MODELS

330i Sedan

2.0-liter BMW TwinPower Turbo inline 4-cylinder, Rear-wheel drive

STARTING MSRP

$41,250

Build Your Own

choose red

choose blue

choose gray

Variants

All options have been chosen

Variational Artifact

That which can be configured to produce a variant

CORE MODELS

330i Sedan

2.0-liter BMW TwinPower Turbo inline 4-cylinder,
Rear-wheel drive

STARTING MSRP

$41,250

Build Your Own

choose red

choose blue

choose gray

Variants

All options have been chosen

Variational Artifact

That which can be configured to produce a variant

CORE MODELS

330i Sedan

2.0-liter BMW TwinPower Turbo inline 4-cylinder, Rear-wheel drive

STARTING MSRP

$41,250

Build Your Own

choose red

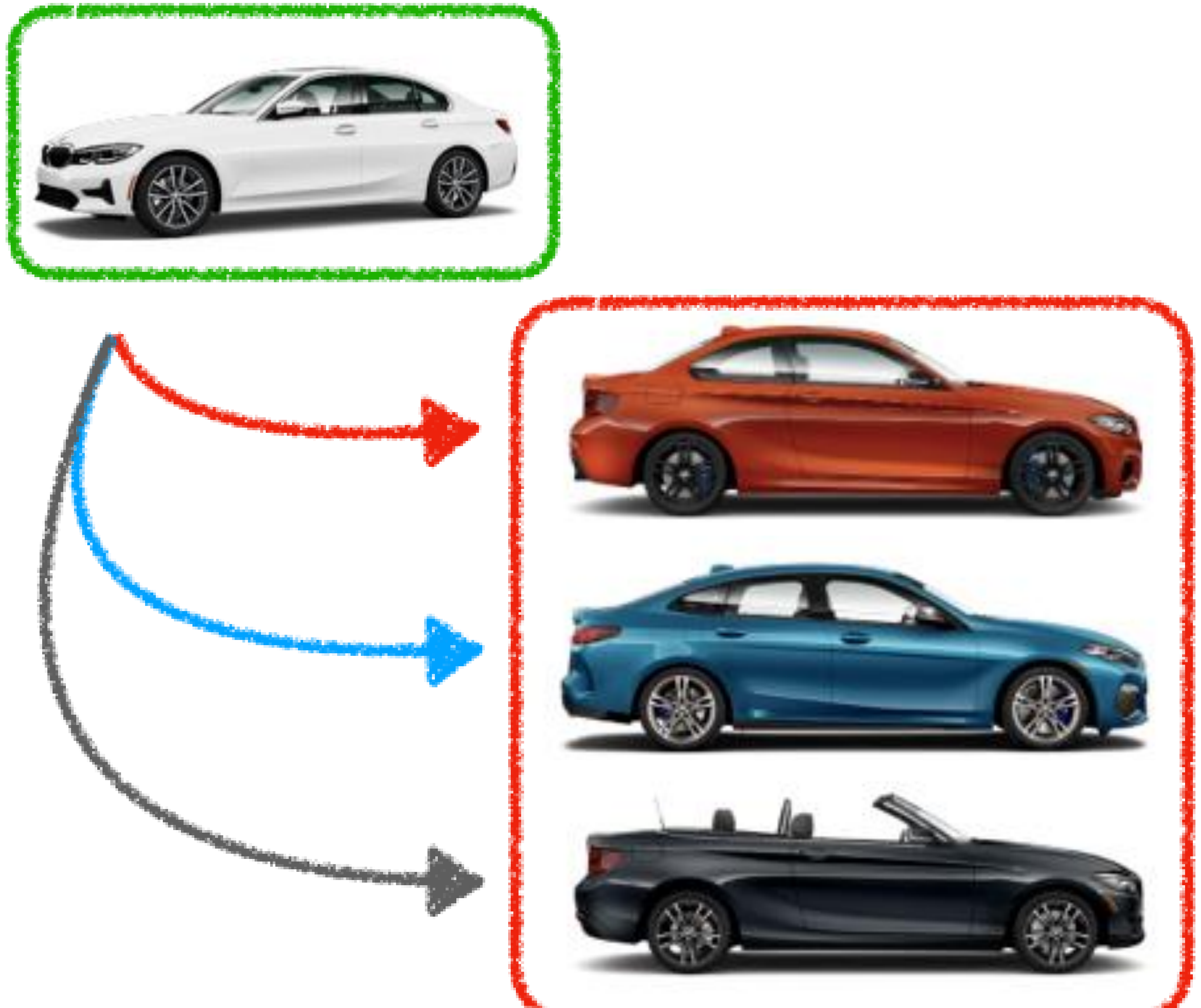
choose blue

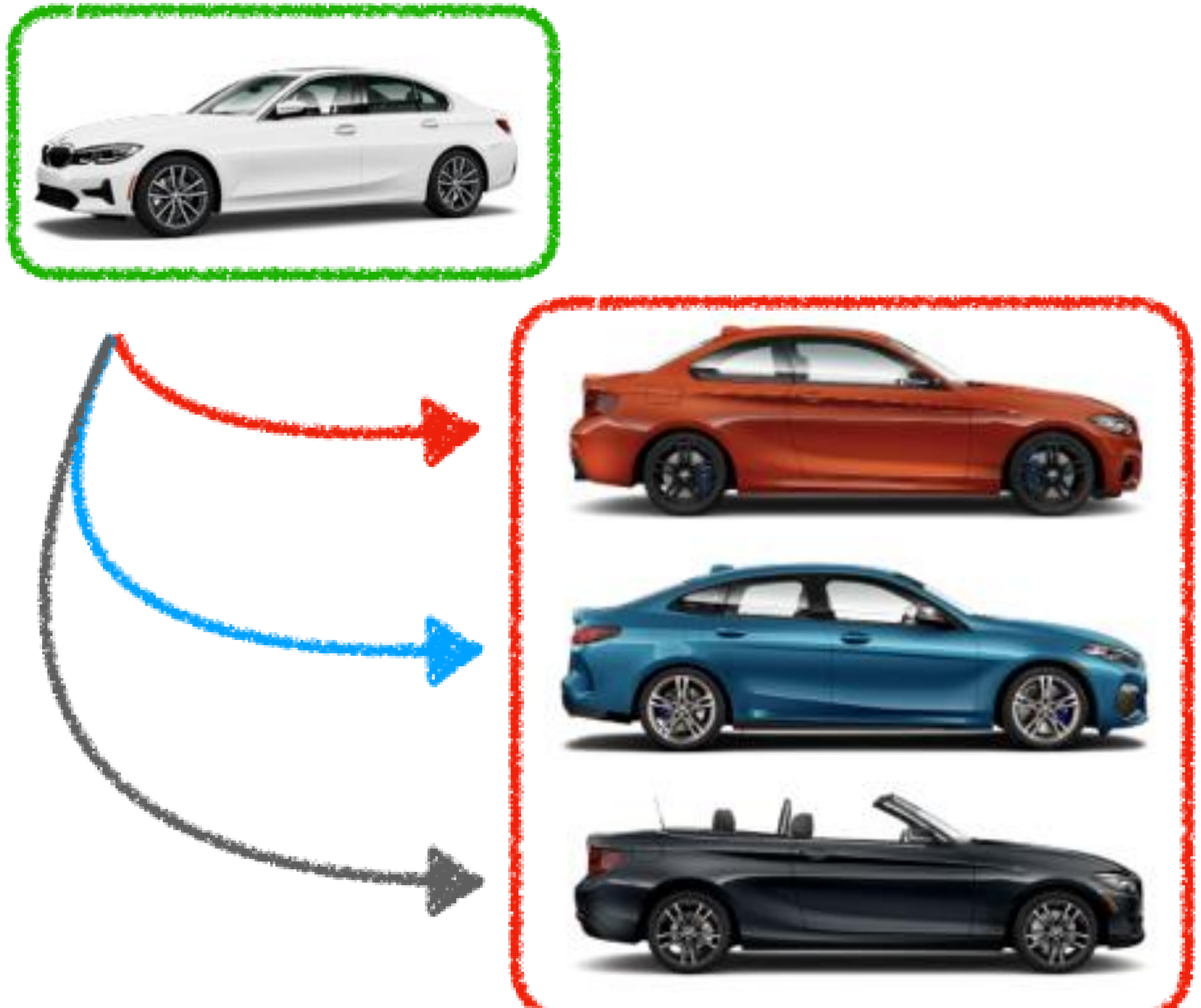
choose gray

Variants

All options have been chosen

A variation-aware system is a system that operates on variational artifacts, *not* on variants.





$$a \wedge b \wedge c \wedge e$$

$$a \wedge (b \vee \neg i) \wedge c$$

$$z \leftrightarrow (a \wedge b \wedge c \wedge e)$$

Modern SAT solvers operate on variants

SAT solver that operates on a variational artifact

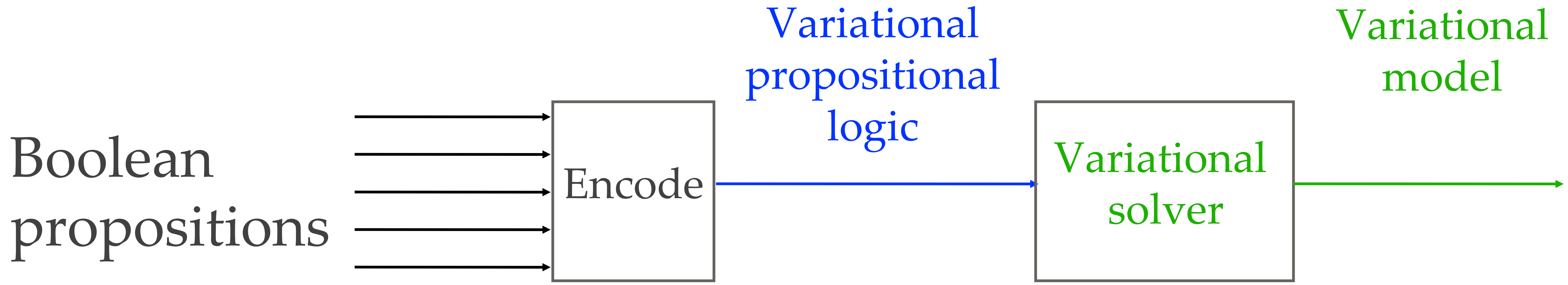
?

$a \wedge b \wedge c \wedge e$

$a \wedge (b \vee \neg i) \wedge c$

$z \leftrightarrow (a \wedge b \wedge c \wedge e)$

Build a satisfiability solver that is variation-aware



1. Define variational propositional logic

2. Develop a prototype variational SAT solver

3. Evaluate the solver on empirical data.
Show these ideas work in practice.

VPL = Propositional Logic + Choice Calculus

$$p \triangleq FM \land c_1 \land c_2 \land \cdots \land c_n$$

$$q \triangleq FM \land c_1 \land c_{10} \land \cdots \land c_n$$

New connective!
a Choice

$$p\_or\_q \triangleq FM \land c_1 \land \boxed{A\langle c_2, c_{10}\rangle} \land \cdots \land c_n$$

With choices, shared terms are statically explicit!

The Choice Calculus: A Representation for Software Variation, Erwig and Walkingshaw, TOSEM 2011

Define variational propositional logic

Plain

Dimension

$$p\_or\_q \triangleq FM \land c_1 \land A\langle c_2, c_{10}\rangle \land \cdots \land c_n$$

Left Alternative

Right Alternative

Configurations

$A = True$

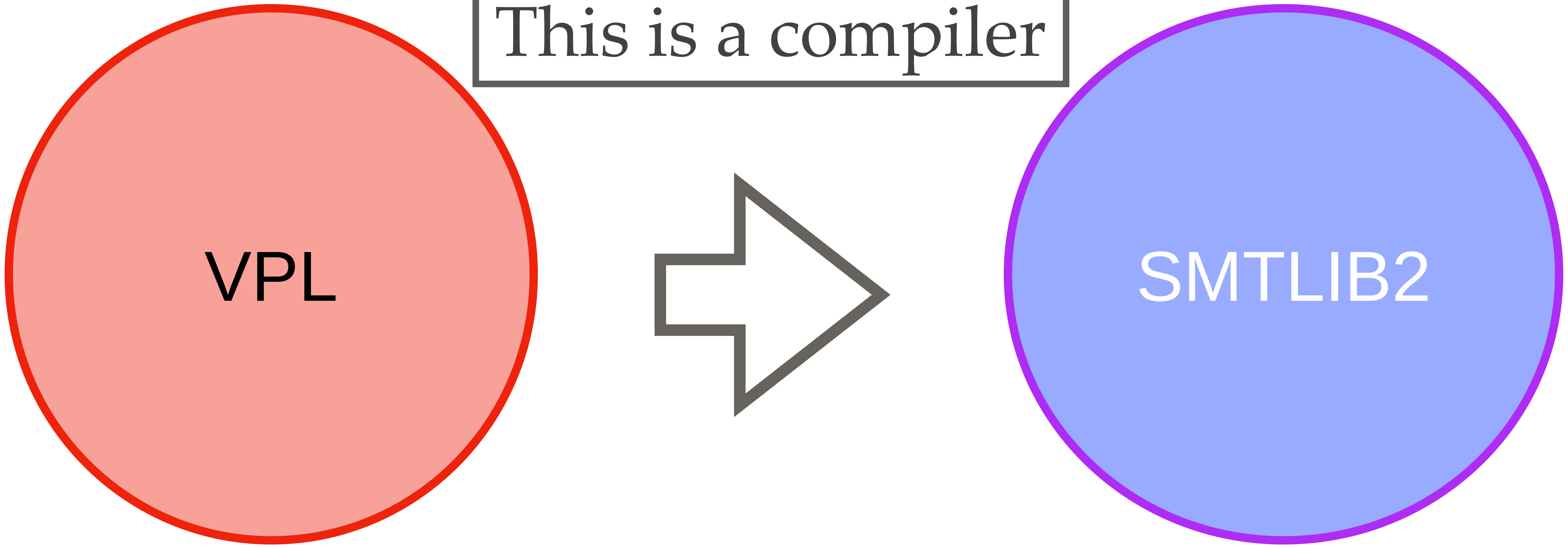
$A = False$

Variants

$$FM \land c_1 \land c_2 \land \cdots \land c_n$$

$$FM \land c_1 \land c_{10} \land \cdots \land c_n$$

This is a compiler



$$FM \wedge (c_1 \vee A\langle c_2, c_{10} \rangle \vee \neg(c_i \wedge c_j))$$

```
(declare-const e1 Int)
(declare-const e2 Int)
(declare-const a1 (Array Int Int))
(declare-const a2 (Array Int Int))
(assert (= (store a2        3 A⟨1202,2718⟩) a2))
(assert (= (store A⟨a1,a2⟩ 3 1729        ) a1))
(assert (= (select A⟨a1,a2⟩ 3) e1))
(assert (= (select a2       3) e2))
(check-sat)
(get-model)
```

Goal:
Performance at least as good as a hand-written incremental program.

Given a VPL formula: $$FM \wedge (c_1 \vee A\langle c_2, c_{10}\rangle \vee \neg(c_i \wedge c_j))$$

Strategy: Define a intermediate language, solve in 3 modes

Evaluation — Send terms to the solver if safe to do so

Accumulation — Cache and combine plain terms for future use

Choice Removal — Manipulate the configuration to remove choices

$$FM \wedge (c_1 \vee A\langle c_2, c_{10} \rangle \vee \neg(c_i \wedge c_j))$$

Evaluation

$$(c_1 \vee A\langle c_2, c_{10} \rangle \vee \neg(c_i \wedge c_j))$$

Accumulation

$$\ldots \vee A\langle c_2, c_{10} \rangle \vee \cdots$$

Choice Removal

$$FM \wedge (c_1 \vee A\langle c_2, c_{10}\rangle \vee \neg(c_i \wedge c_j))$$

$Model_{A=True}$

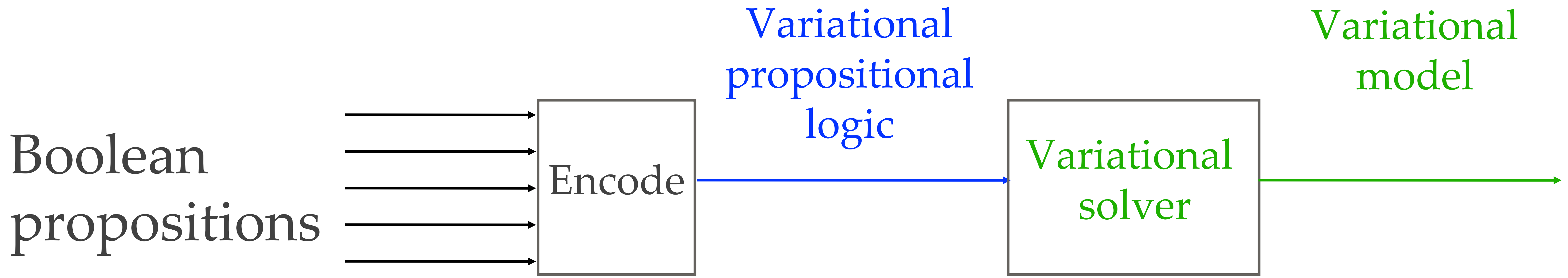
$FM \rightarrow T$
$c_1 \rightarrow T$
$c_2 \rightarrow T$
$\vdots$

$Model_{A=False}$

$FM \rightarrow T$
$c_1 \rightarrow F$
$c_{10} \rightarrow T$
$\vdots$

$Variational\ Model$

$\_Sat \rightarrow A \vee \neg A$
$FM \rightarrow A \vee \neg A$
$c_1 \rightarrow A$
$c_{10} \rightarrow \neg A$
$c_2 \rightarrow A$
$\vdots$



1. Define variational propositional logic ✓

2. Develop a prototype variational SAT solver ✓

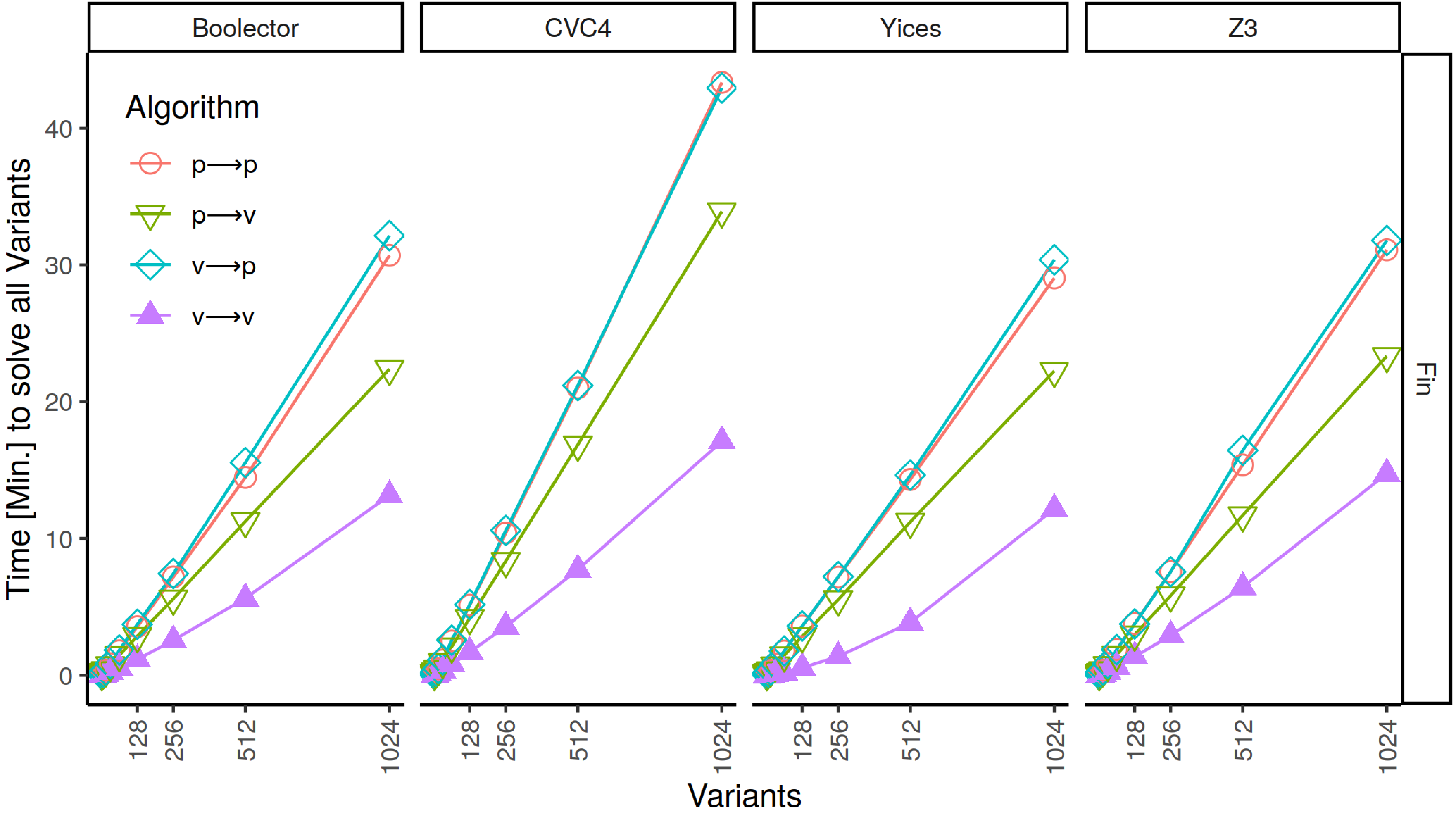
3. Empirically evaluate the solver

RQ1: Performance by variants

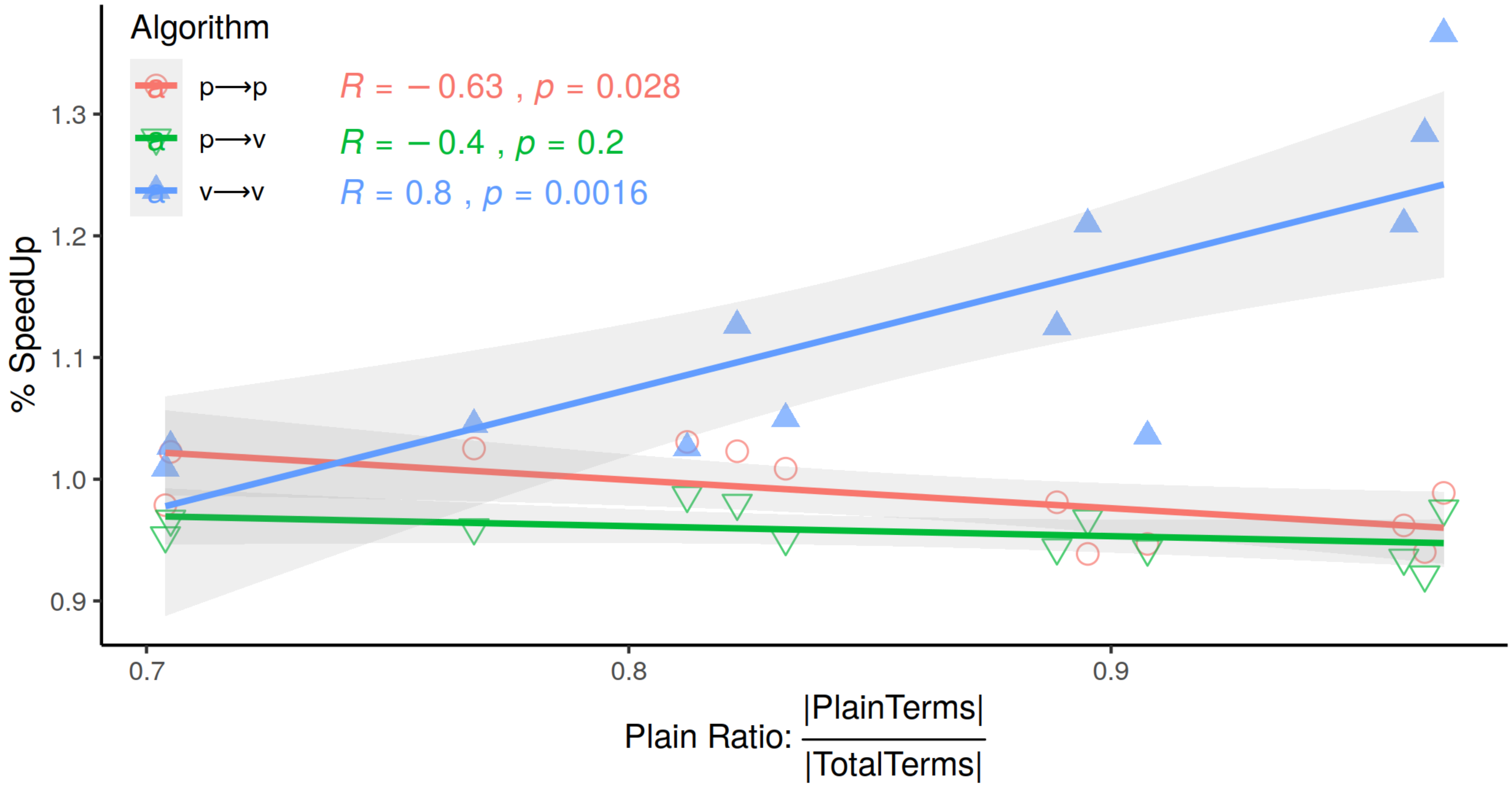
	Automotive02	FinancialServices01
# unique dimensions choices	4	10
# plain terms	26,088	1,441
# choices	4,212	3,809
Max. Obs. Speedup	**2.6-3.5x**	**2.2-2.5x**

*Datasets translated into VPL from Nieke et al, Anomaly Analyses for Feature-Model Evaluation

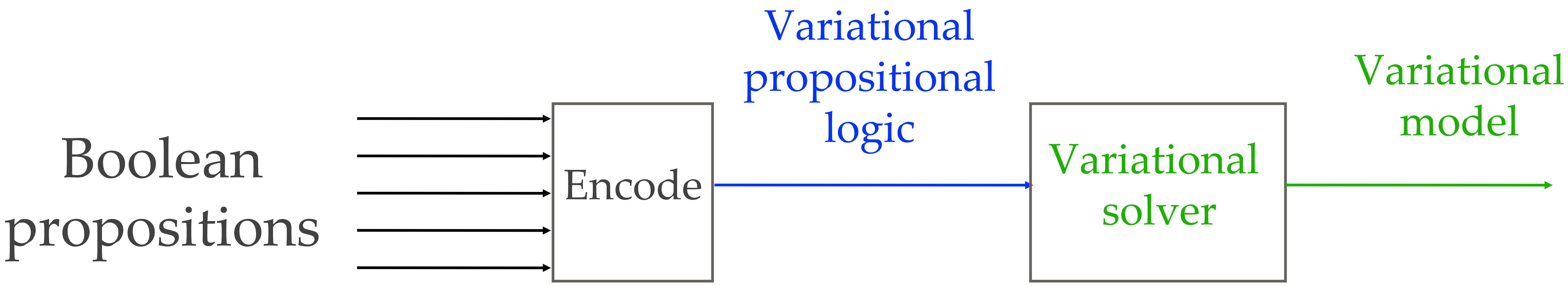
RQ1: Performance by variants



RQ2: Impact of sharing on performance



Plain ratio of a VPL formula is positively correlated to performance



Frequent need to solve many related satisfiability problems.

Our solution: variational satisfiability solving.

Must know all points of variation before solving.

Sharing impacts performance.