Propositional Interpolation & Abstract Interpretation

Vijay D'Silva

Propositional Interpolation

A

B

$P \wedge (P \Rightarrow Q)$

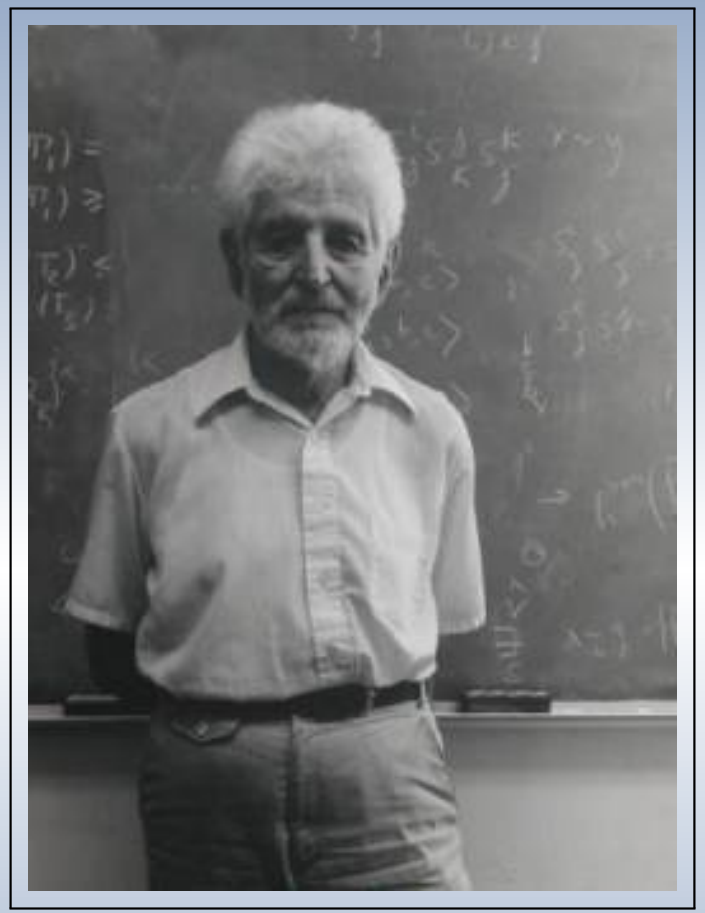
$R \Rightarrow Q$

$$A \quad \Rightarrow \quad I \quad \Rightarrow \quad B$$

$$P \wedge (P \Rightarrow Q) \qquad\qquad R \Rightarrow Q$$

$$A \quad \Rightarrow \quad I \quad \Rightarrow \quad B$$

$$P \wedge (P \Rightarrow Q) \quad \Rightarrow \quad Q \quad \Rightarrow \quad R \Rightarrow Q$$



William Craig

THE JOURNAL OF SYMBOLIC LOGIC
VOLUME 22, Number 3, Sept. 1957

LINEAR REASONING.
A NEW FORM OF THE HERBRAND-GENTZEN THEOREM.

WILLIAM CRAIG

1. Introduction. In Herbrand's Theorem [2] or Gentzen's Extended Hauptsatz [1], a certain relationship is asserted to hold between the structures of A and A′, whenever A *implies* A′ (i.e., A ⊃ A′ is valid) and moreover A is a conjunction and A′ an alternation of first-order formulas in prenex normal form. Unfortunately, the relationship is described in a roundabout way, by relating A and A′ to a quantifier-free tautology. One purpose of this paper is to provide a description which in certain respects is more direct. Roughly speaking, ascent to A ⊃ A′ from a quantifier-free level will be replaced by movement from A to A′ on the quantificational level. Each movement will be closely related to the ascent it replaces.

Theorem 1 (Craig 1957)

Let A and B be closed, first order formulae.

If $A \Rightarrow B$ is valid, there exists a formula I such that

$$A \Rightarrow I \quad \textit{and} \quad I \Rightarrow B \quad \textit{and} \quad \mathrm{Var}(I) \subseteq \mathrm{Var}(A) \cap \mathrm{Var}(B).$$

This talk: A and B are propositional formulae.
Observe that $A \Rightarrow B$ iff $A \wedge \neg B$ is unsatisfiable.

Abstract Interpretation

ABSTRACT INTERPRETATION : A UNIFIED LATTICE MODEL FOR STATIC ANALYSIS OF PROGRAMS BY CONSTRUCTION OR APPROXIMATION OF FIXPOINTS

Patrick Cousot* and Radhia Cousot**

Laboratoire d'Informatique, U.S.M.G., BP. 53
38041 Grenoble cedex, France

1. *Introduction*

A program denotes computations in some universe of objects. Abstract interpretation of programs consists in using that denotation to describe computations in another universe of abstract objects, so that the results of abstract execution give some informations on the actual computations. An intuitive example (which we borrow from Sintzoff [72]) is the rule of signs. The text -1515 * 17 may be understood to denote computations on the abstract universe {(+), (-), (±)} where the semantics of arithmetic operators is defined by the rule of signs. The abstract execution -1515 * 17 ==> -(+) * (+) ==> (-) * (+) ==> (-), proves that -1515 * 17 is a negative number. Abstract interpretation is concerned by a particular underlying structure of the usual universe of computations (the sign, in our example). It gives a summary of some facets of the actual executions of a program. In general this summary is simple to obtain but inaccurate (e.g. -1515 + 17 ==> -(+) + (+) ==> (-) + (+) ==> (±)). Despite its fundamentally incomplete results abstract interpretation allows the programmer or the compiler to answer questions which do not need full knowledge of program executions or which tolerate an imprecise answer, (e.g. partial correctness proofs of programs ignoring the termination problems, type checking, program optimizations which are not carried in the absence of certainty about their feasibility, ...).

Abstract program properties are modeled by a complete semilattice, Birkhoff[61]. Elementary program constructs are locally interpreted by order preserving functions which are used to associate a system of recursive equations with a program. The program global properties are then defined as one of the extreme fixpoints of that system, Tarski[55]. The abstraction process is defined in section 6. It is shown that the program properties obtained by an abstract interpretation of a program are consistent with those obtained by a more refined interpretation of that program. In particular, an abstract interpretation may be shown to be consistent with the formal semantics of the language. Levels of abstraction are formalized by showing that consistent abstract interpretations form a lattice (section 7). Section 8 gives a constructive definition of abstract properties of programs based on constructive definitions of fixpoints. It shows that various classical algorithms such as Kildall [73], Wegbreit[75] compute program properties as limits of finite Kleene[52]'s sequences. Section 9 introduces finite fixpoint approximation methods to be used when Kleene's sequences are infinite, Cousot[76]. They are shown to be consistent with the abstraction process. Practical examples illustrate the various sections. The conclusion points out that abstract interpretation of programs is a unified approach to apparently unrelated program analysis techniques.

SYSTEMATIC DESIGN OF PROGRAM ANALYSIS FRAMEWORKS

Patrick Cousot* and Radhia Cousot**

Laboratoire d'Informatique,U.S.M.G., BP.53X
38041 Grenoble cedex, France

1. INTRODUCTION and SUMMARY

Semantic analysis of programs is essential in optimizing compilers and program verification systems. It encompasses data flow analysis, data type determination, generation of approximate invariant assertions, etc.

Several recent papers (among others Cousot & Cousot[77a], Graham & Wegman[76], Kam & Ullman[76], Kildall[73], Rosen[78], Tarjan[76], Wegbreit[75]) have introduced abstract approaches to program analysis which are tantamount to the use of a *program analysis framework* (A,t,γ) where A is a lattice of (approximate) assertions, t is an (approximate) predicate transformer and γ is an often implicit function specifying the meaning of the elements of A. This paper is devoted to the systematic and correct design of program analysis frameworks with respect to a formal semantics.

Preliminary definitions are given in Section 2 concerning the merge over all paths and (least) fixpoint program-wide analysis methods. In Section 3 we briefly define the (forward and backward) deductive semantics of programs which is later used as a formal basis in order to prove the correctness of the approximate program analysis frameworks. Section 4 very shortly recall the main elements of the lattice theoretic approach to approximate semantic analysis of programs.

In Section 6 we study and examplify various methods which can be used in order to define a space of approximate assertions or equivalently an approximation function. They include the characterization of the least Moore family containing an arbitrary set of assertions, the construction of the least closure operator greater than or equal to an arbitrary approximation function, the definition of closure operators by composition, the definition of a space of approximate assertions by means of a complete join congruence relation or by means of a family of principal ideals.

Section 7 is dedicated to the design of the approximate predicate transformer induced by a space of approximate assertions. First we look for a reasonable definition of the correctness of approximate predicate transformers and show that a local correctness condition can be given which has to be verified for every type of elementary statement. This local correctness condition ensures that the (merge over all paths or fixpoint) global analysis of any program is correct. Since isotony is not required for approximate predicate transformers to be correct it is shown that non-isotone program analysis frameworks are manageable although it is later argued that the isotony hypothesis is natural. We next show that among all possible approximate predicate transformers which can be used with a given space of approximate assertions there exists a best one which provides the maximum information relative to a program-wide analysis method. The best approximate predicate transformer

Concrete Domain

Concrete Semantics

Abstract Domain and Semantics

Optimal Abstractions

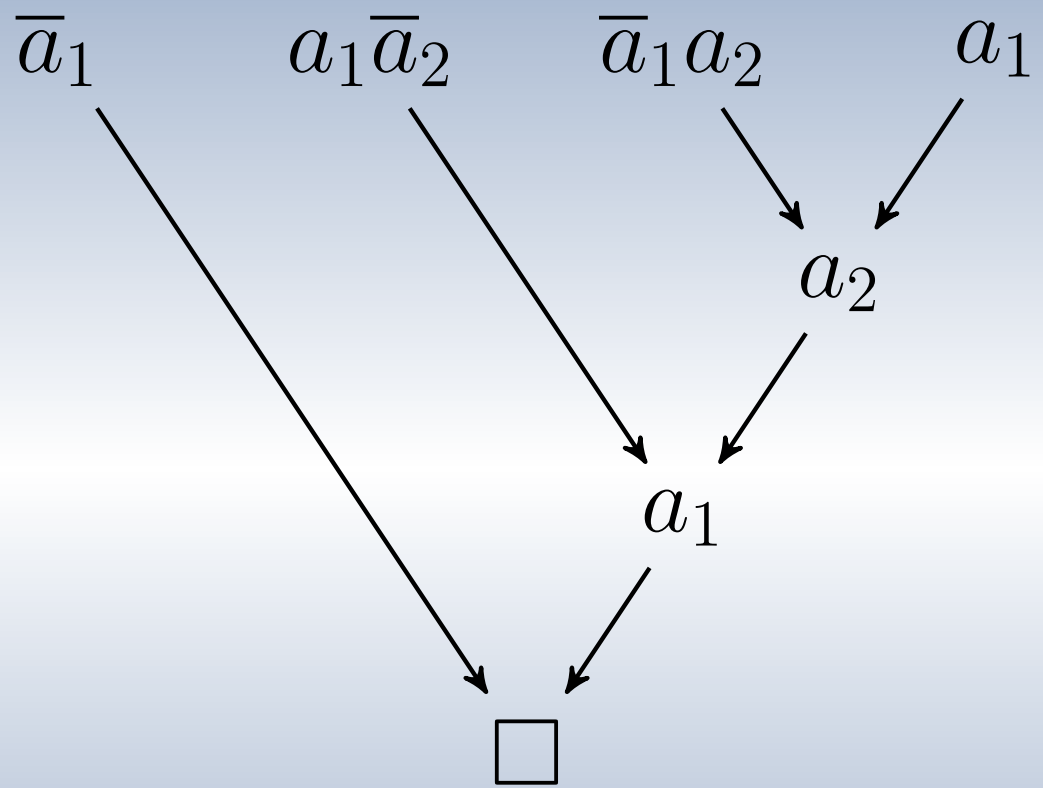
Concrete Domain	$\{\langle$Coloured Clause, Formula$\rangle\}$
Concrete Semantics	Resolution + Interpolation
Abstract Domain and Semantics	Restricted Colouring
Optimal Abstractions	Existing Algorithms

Interpolation Algorithms: Very Short Introduction

Resolution

Var	Propositional variables: $a_1, a_2, a_3, \ldots$
Literal	Variable or its negation: $a, \overline{a}, \neg a$
Clause	Disjunction or set of literals: $\{x_1, x_2, x_5\}$
CNF Formula	Conjunction or set of clauses: $\{\{a\}, \{\overline{a}, b\}\}$

$$\frac{x \vee C \qquad D \vee \overline{x}}{C \vee D} \quad [\text{Resolution}]$$


$\overline{a}_1$
$a_1\overline{a}_2$
$\overline{a}_1 a_2$
a_1
a_2
a_1
$\square$

Constructing Craig Interpolation Formulas

Guoxiang Huang

Mathematics Department, University of Hawaii at Manoa
Honolulu, HI 96822 (E-mail: huang@math.hawaii.edu)

Abstract. A Craig interpolant of two inconsistent theories is a formula which is true in one and false in the other. This paper gives an efficient method for constructing a Craig interpolant from a refutation proof which involves binary resolution, paramodulation, and factoring. This method can solve the machine learning problem of discovering a first order concept from given examples. It can also be used to find sentences which distinguish pairs of nonisomorphic finite structures.

1 Background and Introduction

Let Σ and Π be two inconsistent first order theories. Then by Craig's Interpolation Theorem, there is a sentence θ, called a *Craig interpolant*, such that θ is true in Σ and false in Π and every nonlogical symbol occurring in θ occurs in both Σ and Π. Craig interpolants can be used to solve the problem of learning a first order concept by letting Σ and Π be the lists of positive and negative examples of the concept to be learned.

The standard nonconstructive model-theoretic proof of Craig's Theorem is in [3]. Lyndon showed how to construct an interpolant from a special form of natural deduction (see [1]). We show how to construct an interpolant from a refutation proof which uses binary resolution, factoring and paramodulation. In our examples, we use OTTER (the standard text on OTTER is [4]) to generate such proofs.

Daniele Mundici

Arch. math. Logik **23** (1983), 27–36

A LOWER BOUND FOR THE COMPLEXITY OF CRAIG'S INTERPOLANTS IN SENTENTIAL LOGIC*

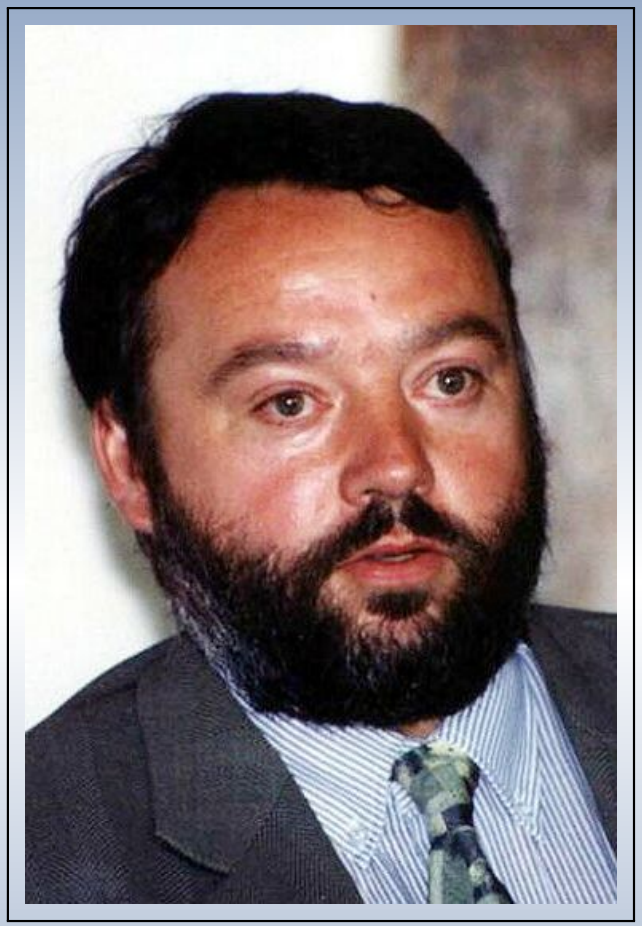
Daniele Mundici

Abstract

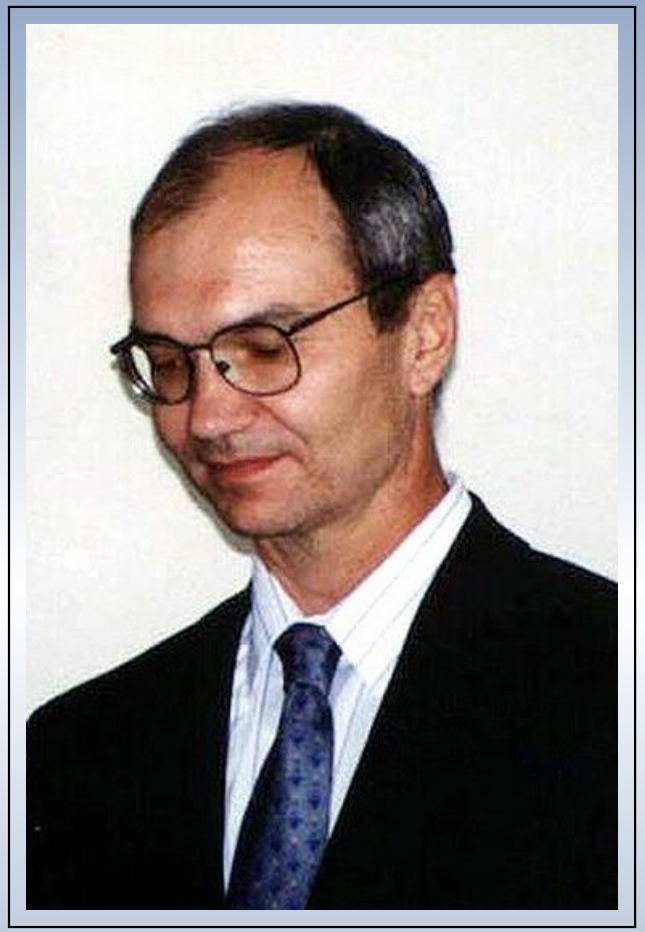
For any sentence α (in sentential logic) let d_α be the delay complexity of the boolean function f_α represented by α. We prove that for infinitely many d (and starting with some $d<620$) there exist valid implications $\alpha\to\beta$ with $d_\alpha, d_\beta \leqq d$ such that any Craig's interpolant χ has its delay complexity d_χ greater than $d+(1/3)\cdot\log(d/2)$. This is the first (non-trivial) known lower bound on the complexity of Craig's interpolants in sentential logic, whose general study may well have an impact on the central problems of computation theory.

Theorem 2 (Mundici 1983)

A polynomial upper-bound on the circuit-size of propositional interpolants implies that $\mathsf{NP} \cap \mathsf{coNP}$ *has polynomial-size circuits.*



Jan Krajíček



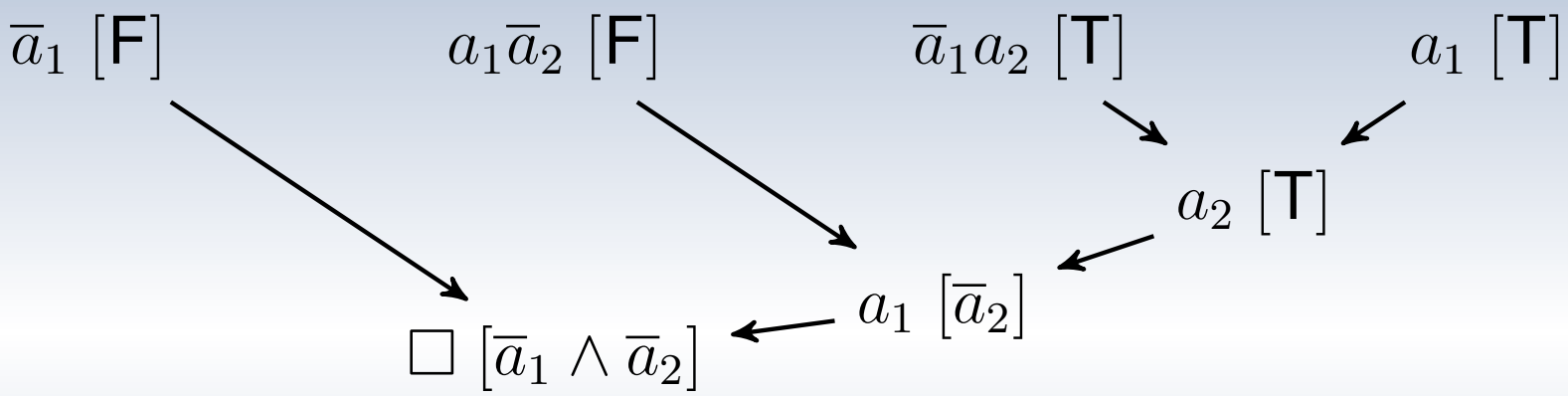
Pavel Pudlák

Bounding interpolant-size is hard. Easier to study the relationship between proof size and interpolant size.

Theorem 3 (Krajíček 1997, Pudlák 1997)

If $A \wedge B$ is unsatisfiable and has a resolution refutation of size n, then there is an interpolant I with circuit-size at most $3n$.

$$A \stackrel{\text{def}}{=} (\overline{a}_1) \wedge (a_1 \vee \overline{a}_2) \qquad B \stackrel{\text{def}}{=} (\overline{a}_1 \vee a_2) \wedge (a_1)$$



$$\frac{\langle x \vee C, I_1 \rangle \qquad \langle D \vee \overline{x}, I_2 \rangle}{\langle C \vee D, I \rangle}$$

$$\begin{array}{cc} I_1 \vee I_2 & \text{if } x \in \mathrm{Var}(A) \setminus \mathrm{Var}(B) \\ (x \vee I_1) \wedge (I_2 \vee \overline{x}) & \text{if } x \in \mathrm{Var}(A) \cap \mathrm{Var}(B) \\ I_1 \wedge I_2 & \text{if } x \in \mathrm{Var}(B) \setminus \mathrm{Var}(A) \end{array}$$

Interpolation and SAT-based Model Checking

K. L. McMillan

Cadence Berkeley Labs

Abstract. We consider a fully SAT-based method of unbounded symbolic model checking based on computing Craig interpolants. In benchmark studies using a set of large industrial circuit verification instances, this method is greatly more efficient than BDD-based symbolic model checking, and compares favorably to some recent SAT-based model checking methods on positive instances.

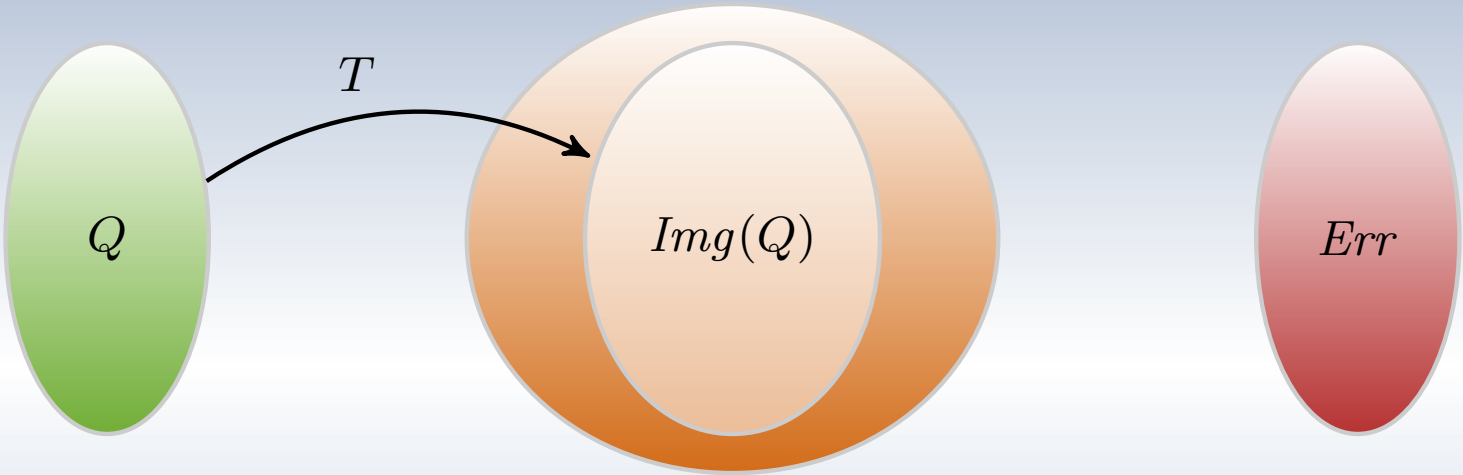
1 Introduction

Symbolic model checking [8, 9] is a method of verifying temporal properties of finite (and sometimes infinite) state systems that relies on a symbolic representation of sets, typically as Binary Decision Diagrams [7] (BDD's). By contrast, bounded model checking [4] can falsify temporal properties by posing the existence of a counterexample of k steps or fewer as a Boolean satisfiability (SAT) problem. Using a modern SAT solver, this method is efficient in producing counterexamples [10, 6]. However, it cannot verify properties unless an upper bound is known on the depth of the state space, which is not generally the case.

This paper presents a purely SAT-based method of *unbounded* model checking. It exploits a SAT solver's ability to produce refutations. In bounded model

If $Q(x_0) \wedge T(x_0, x_1) \wedge Err(x_1)$ is UNSAT, there exists $I(x_1)$:

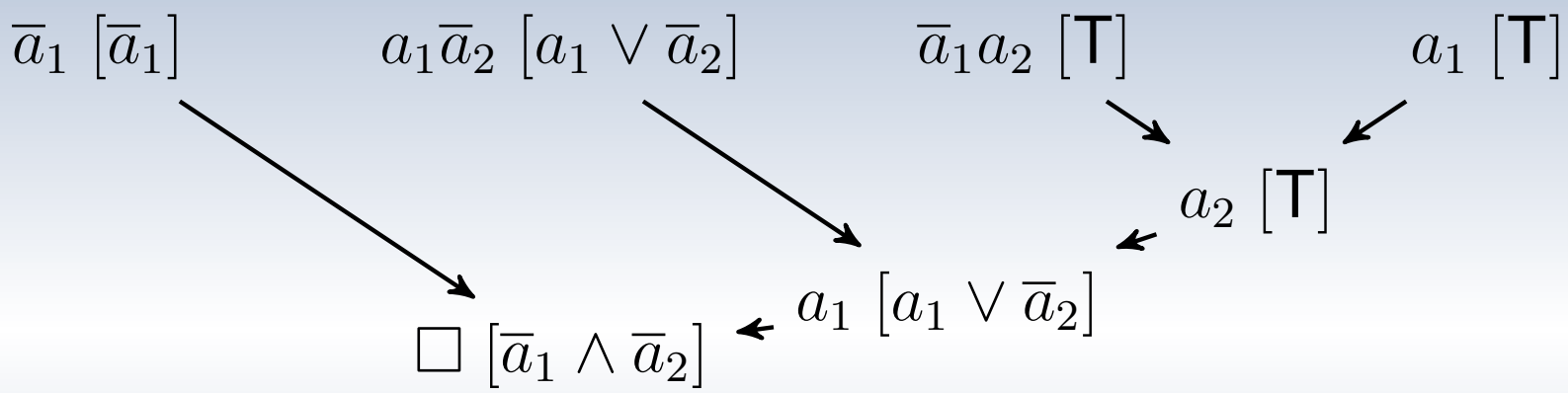
$Q(x_0) \wedge T(x_0, x_1) \Rightarrow I(x_1)$ and $I(x_1) \wedge Err(x_1)$ is UNSAT.



Theorem 4 (McMillan 2003)

There is a purely SAT-based procedure for deciding error reachability in finite-state systems.

$$A \stackrel{\text{def}}{=} (\overline{a}_1) \wedge (a_1 \vee \overline{a}_2) \qquad B \stackrel{\text{def}}{=} (\overline{a}_1 \vee a_2) \wedge (a_1)$$



$$\frac{\langle x \vee C, I_1 \rangle \qquad \langle D \vee \overline{x}, I_2 \rangle}{\langle C \vee D, I \rangle}$$

$$\begin{array}{ll} I_1 \vee I_2 & \text{if } x \in \mathrm{Var}(A) \setminus \mathrm{Var}(B) \\ I_1 \wedge I_2 & \text{if } x \in \mathrm{Var}(A) \cap \mathrm{Var}(B) \\ I_1 \wedge I_2 & \text{if } x \in \mathrm{Var}(B) \setminus \mathrm{Var}(A) \end{array}$$

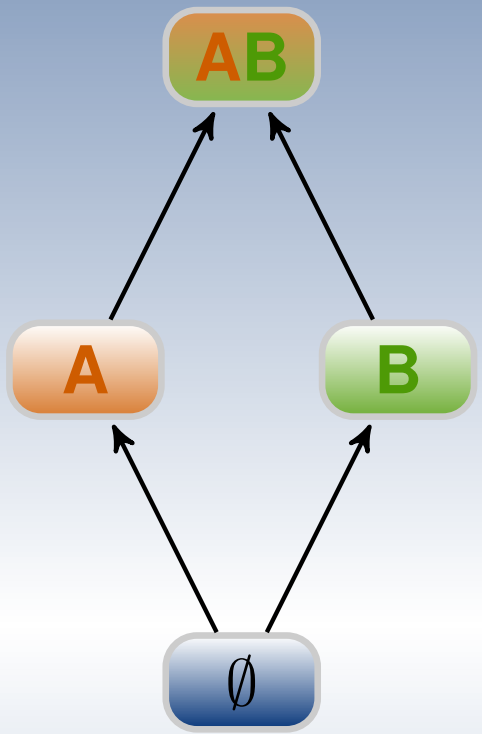
Questions

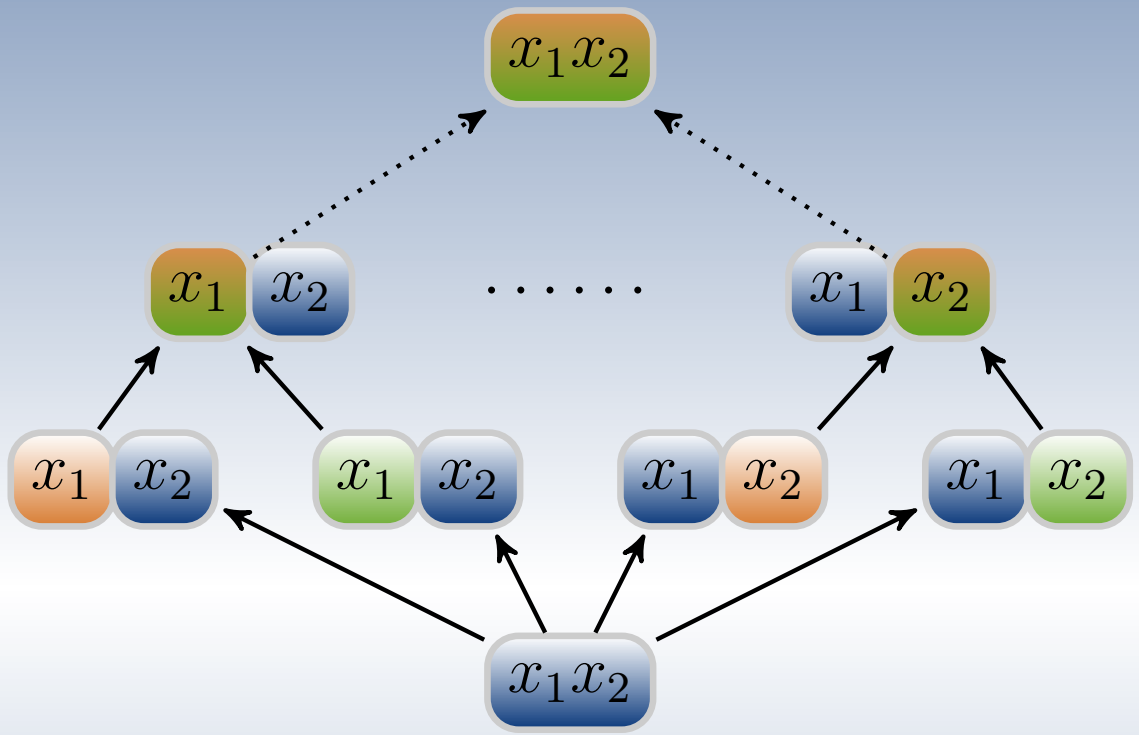
Do more interpolation algorithms exist?

What are the properties of the interpolants obtained?

What are quality metrics for interpolation algorithms?

“Abstract Interpolation”





Concrete Domain

Colours : $\mathcal{S} \stackrel{\text{def}}{=} \{\emptyset, \mathbf{A}, \mathbf{B}, \mathbf{AB}\}$

Coloured clauses: $C \to \mathcal{S}$, a lattice under point-wise order.

Coloured CNF: Set of coloured clauses.

Concrete Semantics: Coloured Interpolation

Base Case

$$C|_{\mathbf{A}} \stackrel{\text{def}}{=} \{x \in C | Colour(x) \sqsubseteq \mathbf{A}\}$$

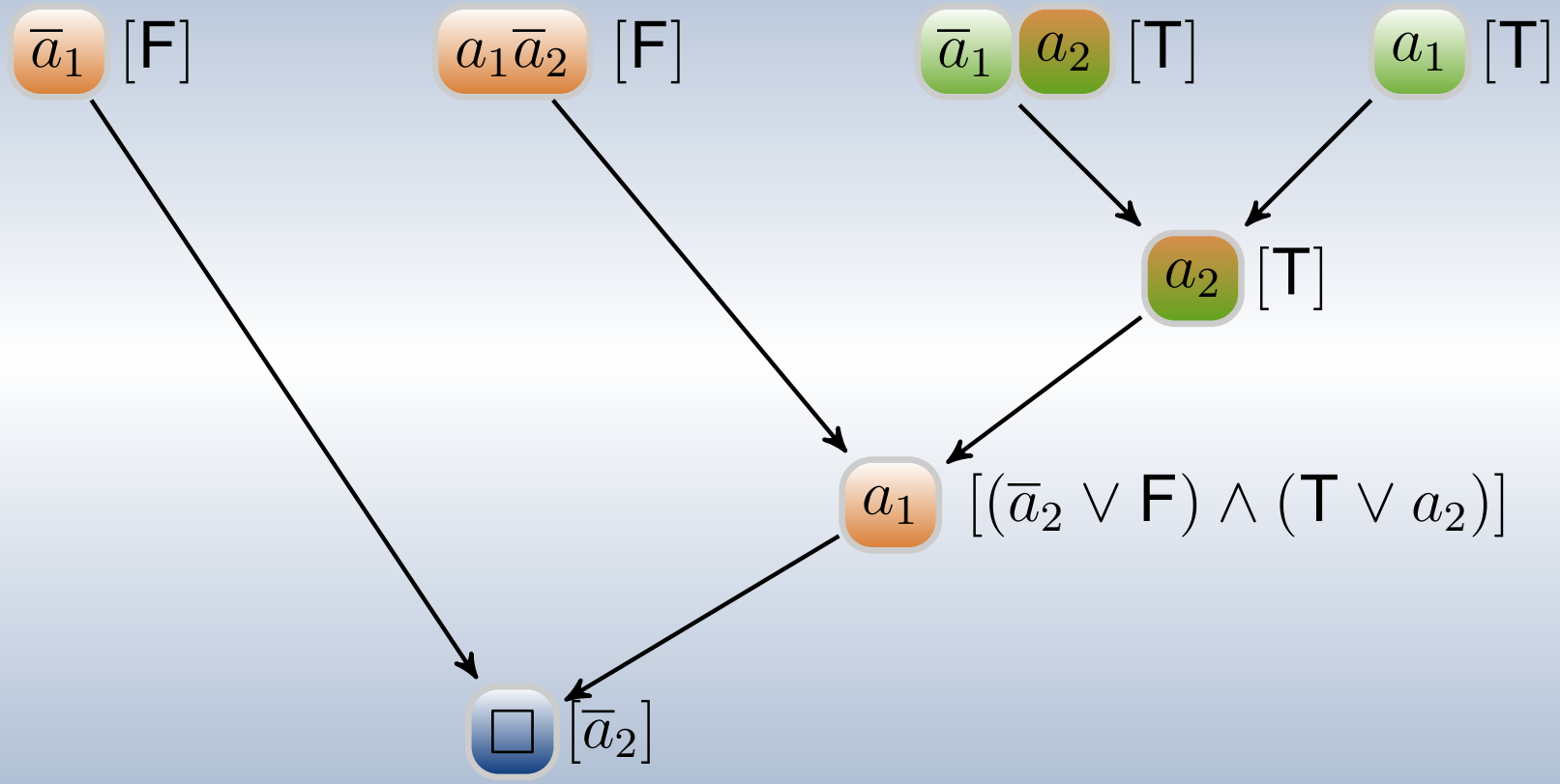
$$\frac{C}{\langle C, C|_{\mathbf{B}} \rangle} \; [C \in A] \qquad \frac{C}{\langle C, C|_{\mathbf{A}} \rangle} \; [C \in B]$$

Interpolation & Resolution

$$\frac{\langle x \vee C, I_1 \rangle \qquad \langle D \vee \overline{x}, I_2 \rangle}{\langle C \sqcup D, I \rangle}$$

$$\begin{array}{cll} I_1 \vee I_2 & \text{if } Colour(x) \sqcup Colour(\overline{x}) = & \mathbf{A} \\ (x \vee I_1) \wedge (I_2 \vee \overline{x}) & \text{if } Colour(x) \sqcup Colour(\overline{x}) = & \mathbf{AB} \\ I_1 \wedge I_2 & \text{if } Colour(x) \sqcup Colour(\overline{x}) = & \mathbf{B} \end{array}$$

$$A \stackrel{\text{def}}{=} (\overline{a}_1) \wedge (a_1 \vee \overline{a}_2) \qquad B \stackrel{\text{def}}{=} (\overline{a}_1 \vee a_2) \wedge (a_1)$$

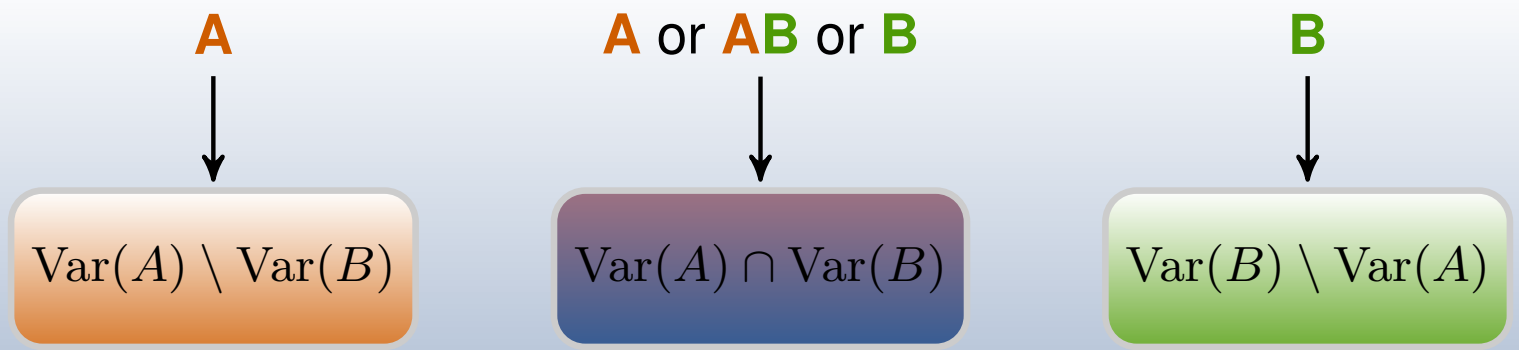


This interpolant cannot be derived with the previously shown algorithms.

Locality Preservation

A colouring of a CNF pair $\langle A, B \rangle$ is *locality preserving* if

- Every literal in the formula has a non-empty colour.
- A literal occurring only in A is coloured **A** and vice-versa.



Theorem 5

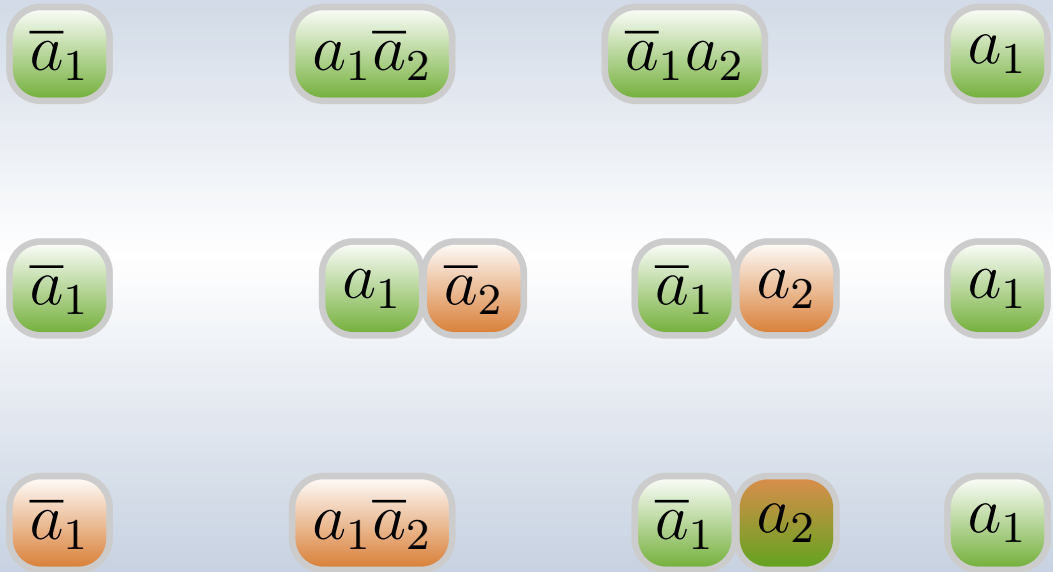
If $\langle A, B \rangle$ is assigned a locality preserving colouring, and $\langle \Box, I \rangle$ is derived with the interpolating resolution rule, then I is an interpolant for A and B.

Proofs for existing algorithms are all different. Adapt inductive invariant from [Yorsh and Musuvathi, CADE 2005].

Locality preserving colourings define interpolation algorithms.

Existing algorithms do not propagate colours. Why?

A colouring is *partitioning* if all occurrences of a variable have the same colour.



Abstraction: Every colouring is contained in a partitioning one. Different partitions define different abstract domains.

Theorem 6

There is a unique, coarsest partition admitting three locality preserving colourings.

McMillan

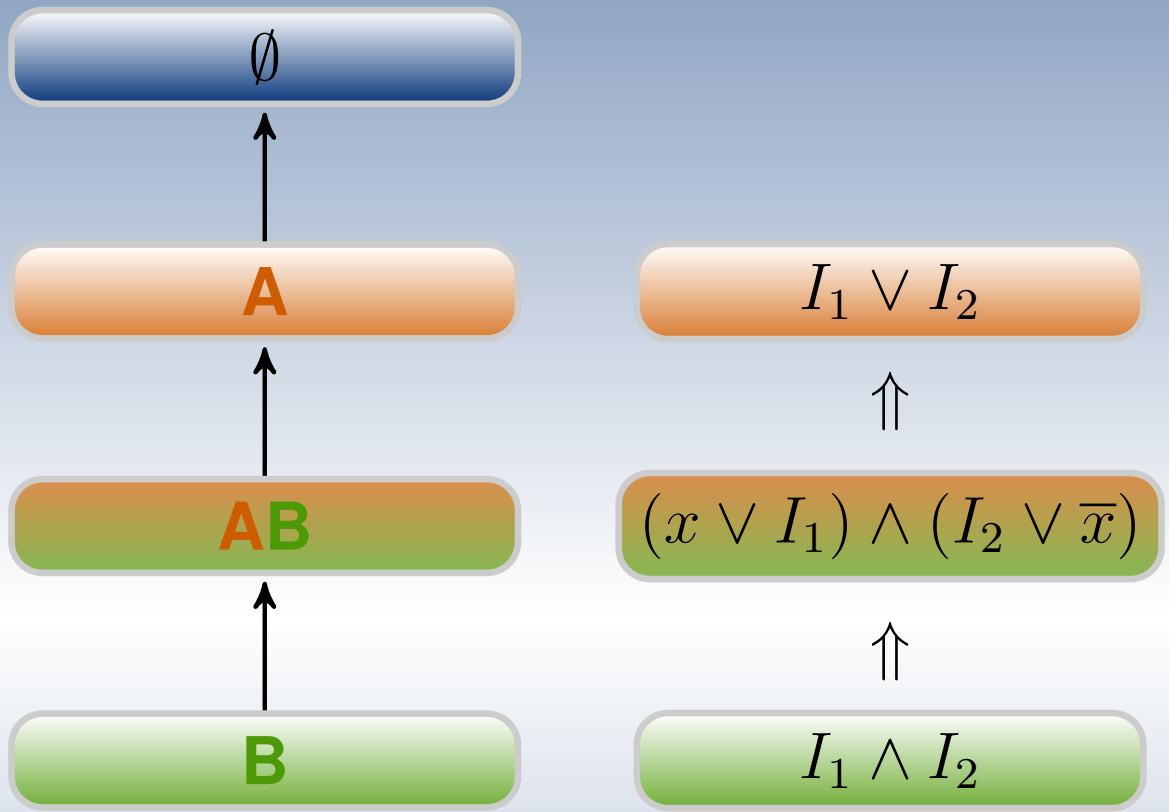
$\mathrm{Var}(A) \setminus \mathrm{Var}(B)$ | $\mathrm{Var}(A) \cap \mathrm{Var}(B)$ | $\mathrm{Var}(B) \setminus \mathrm{Var}(A)$

Huang-Krajíček-Pudlák

$\mathrm{Var}(A) \setminus \mathrm{Var}(B)$ | $\mathrm{Var}(A) \cap \mathrm{Var}(B)$ | $\mathrm{Var}(B) \setminus \mathrm{Var}(A)$

Dual-McMillan

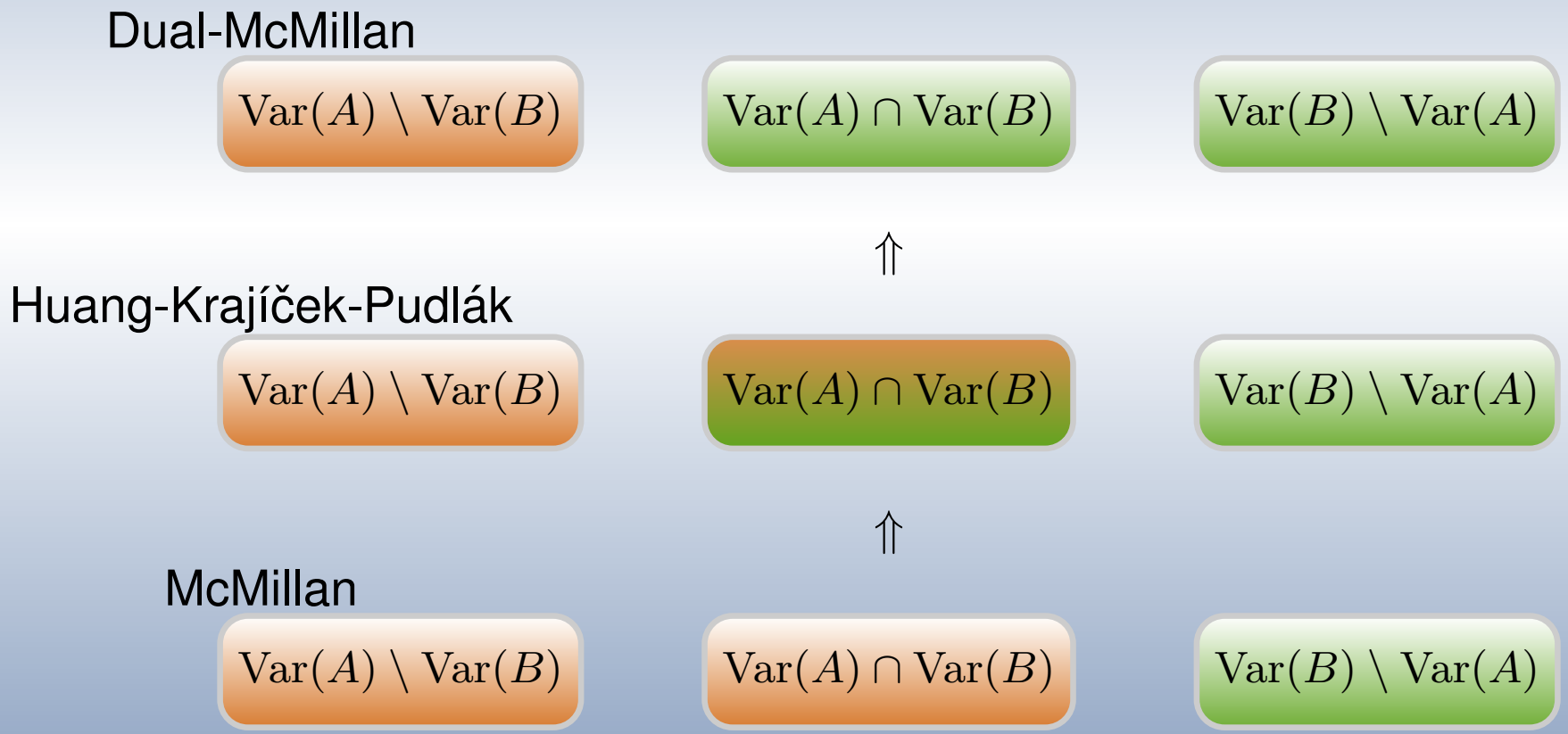
$\mathrm{Var}(A) \setminus \mathrm{Var}(B)$ | $\mathrm{Var}(A) \cap \mathrm{Var}(B)$ | $\mathrm{Var}(B) \setminus \mathrm{Var}(A)$



Order colours by strength of the formulae obtained.
Lift point-wise to an order on coloured clauses.

Theorem 7

The set of locality-preserving colourings form a complete lattice with respect to the strength order.



What else is in the paper?

- Dual of a colour and interpolation algorithm.
- Proofs of the optimality for partitions and strength use order-theoretic arguments.
- Smallest and largest set of variables appearing syntactically in an interpolant (underlying colourings are not partitioning).
- Standard resolution is an abstraction of interpolating-resolution. In fact, it is an α-complete abstraction.

Summary

- Different interpolation algorithms exist.
- Algorithms are defined by colouring functions.
- Study properties of algorithms by studying colourings.
- Existing algorithms are two of three residing in the coarsest abstraction.
- Huang-Krajíček-Pudlák algorithm is the most abstract with respect to colouring order.
- McMillan's algorithm produces strongest interpolants.
- Every algorithm has a dual.

The End

Properties of Colourings

- Locality preserving colourings are a join-semi-lattice.
- Locality is not preserved under meet.
- Every colour has a *dual*. The dual of **A** is **B**, and vice-versa and **AB** and $\emptyset$ are self-duals. Duality extends point-wise to labelling functions.
- If I is the interpolant for $\langle A, B \rangle$ obtained with an colouring, the dual colouring yields $\neg J$ where J is the interpolant for $\langle B, A \rangle$.

Huang-Krajíček-Pudlák Method

$$\frac{\langle x \vee C, I_1 \rangle \qquad \langle D \vee \overline{x}, I_2 \rangle}{\langle C \vee D, I \rangle}$$

$$\begin{array}{cl} I_1 \vee I_2 & \text{if } x \in \mathrm{Var}(A) \setminus \mathrm{Var}(B) \\ (x \vee I_1) \wedge (I_2 \vee \overline{x}) & \text{if } x \in \mathrm{Var}(A) \cap \mathrm{Var}(B) \\ I_1 \wedge I_2 & \text{if } x \in \mathrm{Var}(B) \setminus \mathrm{Var}(A) \end{array}$$

McMillan's Method

$$\frac{\langle x \vee C, I_1 \rangle \qquad \langle D \vee \overline{x}, I_2 \rangle}{\langle C \vee D, I \rangle}$$

$$\begin{array}{ll} I_1 \vee I_2 & \text{if } x \in \mathrm{Var}(A) \setminus \mathrm{Var}(B) \\ I_1 \wedge I_2 & \text{if } x \in \mathrm{Var}(A) \cap \mathrm{Var}(B) \\ I_1 \wedge I_2 & \text{if } x \in \mathrm{Var}(B) \setminus \mathrm{Var}(A) \end{array}$$