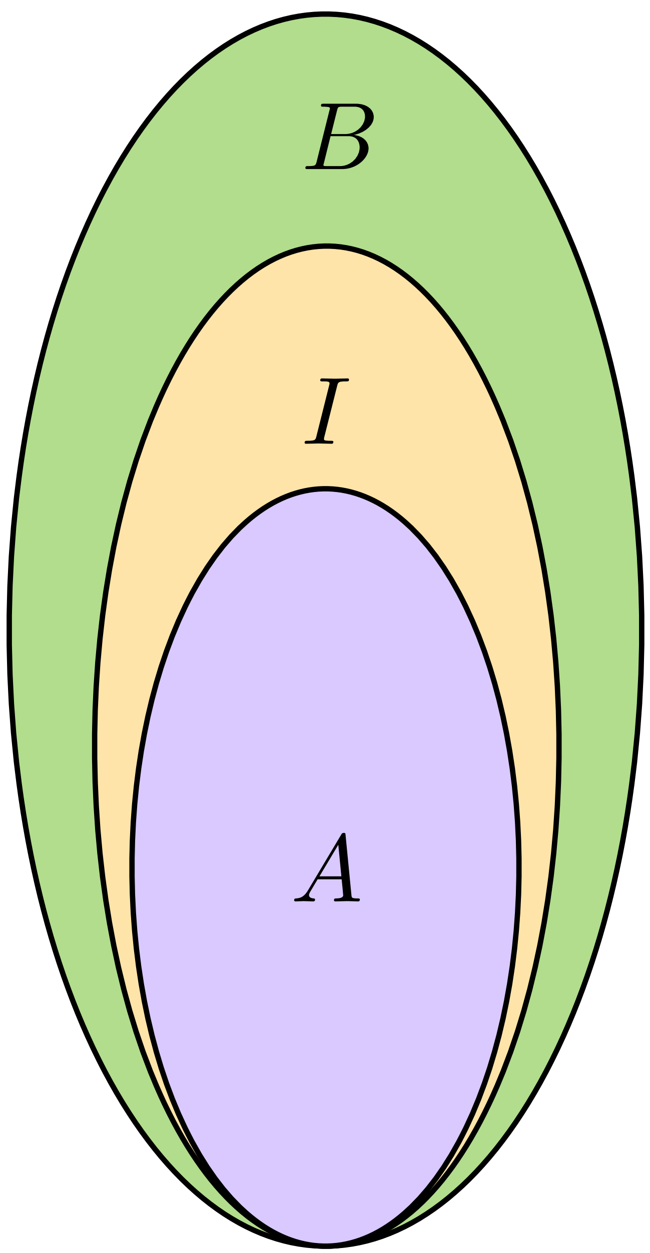
Interpolation: Theory and Applications

Vijay D'Silva
Google Inc., San Francisco

UC Berkeley 2016

1	A Brief History of Interpolation
2	Verification with Interpolants
3	Interpolant Construction
4	Further Reading and Research

Craig Interpolants

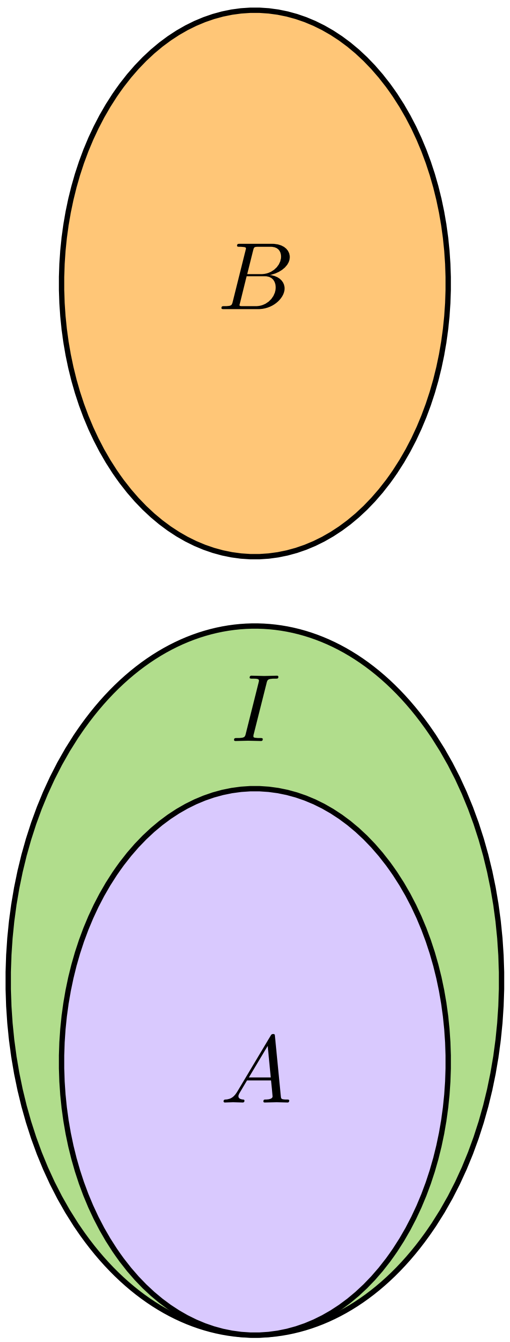


For two formulae A and B such that A implies B, a *Craig interpolant* is a formula I such that

1. A implies I, and
2. I implies B, and
3. the non-logical symbols in I occur in A and in B.

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

Reverse Interpolants or "Interpolants"



For a contradiction $A \wedge B$, a *reverse interpolant* is a formula I such that

1. A implies I, and
2. $I \wedge B$ is a contradiction, and
3. the non-logical symbols in I occur in A and in B.

Note: In classical logic, $A \implies B$ is valid exactly if $A \wedge \neg B$ is a contradiction.

In the verification literature, "interpolant" usually means "reverse interpolant."

International Business Machines Corporation
2050 Rt 52
Hopewell Junction, NY 12533
845-892-5262

October 7, 2008

Dear Andreas,

I would like to congratulate Cadence Research Labs on their 15th Anniversary. In these 15 years, Cadence Research Labs has worked at several frontiers of Electronic Design Automation. They focus on hard problems that when solved significantly push the state of the art forward. They found novel solutions to system, synthesis and formal verification problems.

Formal verification is the process of exhaustively validating that a logic entity behaves correctly. In contrast to testing-based approaches, which may expose flaws though generally cannot yield a proof of correctness, the exhaustiveness of formal verification ensures that no flaw will be left unexposed. Formal verification is thus a critical technology in many domains, being essential to safety-critical applications and

Model checking algorithms are widely used for verifying hardware and software models. CRL has pioneered numerous fundamental ideas and algorithms to this field, including "interpolation" as a satisfiability-based proof method which is often dramatically faster and more scalable than prior proof techniques. CBL researchers invented numerous novel methods to automatically reduce the domain of a verification problem through "abstracting" it based upon unsatisfiability proofs. These techniques have substantially increased the scalability of formal verification of complex hardware designs.

Model checking algorithms are widely used for verifying hardware and software models. CRL has pioneered numerous fundamental ideas and algorithms to this field, including "interpolation" as a satisfiability-based proof method which is often dramatically faster and more scalable than prior proof techniques. CBL researchers invented numerous novel methods to automatically reduce the domain of a verification problem through "abstracting" it based upon unsatisfiability proofs. These techniques have substantially increased the scalability of formal verification of complex hardware designs.

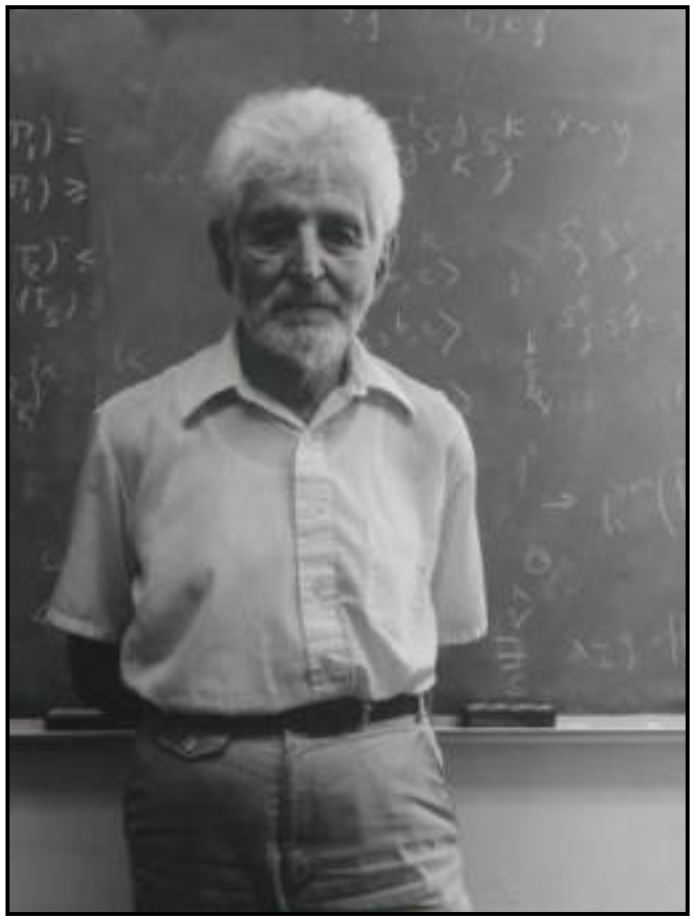
CRL researchers have not only used logic optimizations to speed up formal verification algorithms, but are now also applying them to sequential optimization. Sequential synthesis has long been a holy grail in logic optimization. A large part of the design space remains untapped unless one can reliably and effectively optimize and verify in the sequential domain. Recent progress from CRL shows that there is some promise we can tap into this some time in the not too distant future.

Leon

Leon Stok
Director,
Electronic Design Automation
IBM Corporation

1	A Brief History of Interpolation
2	Verification with Interpolants
3	Interpolant Construction
4	Further Reading and Research

Craig’s Interpolation Lemma (1957)



William Craig in 1988

http://sophos.berkeley.edu/interpolations/

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 $\supset$ 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

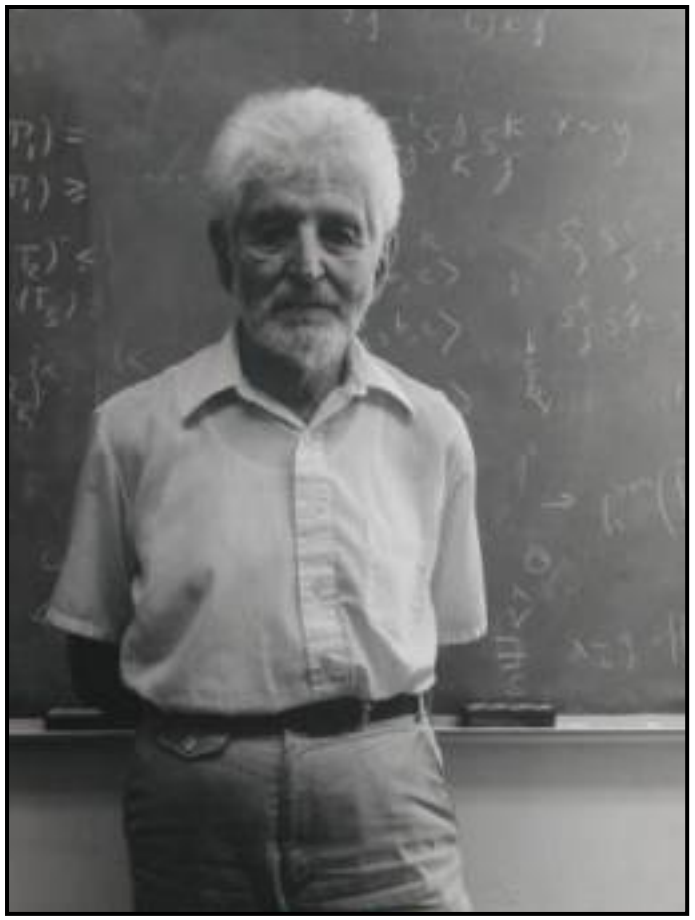
THE JOURNAL OF SYMBOLIC LOGIC
Volume 22, Number 3, Sept, 1957

THREE USES OF THE HERBRAND-GENTZEN THEOREM IN RELATING MODEL THEORY AND PROOF THEORY

WILLIAM CRAIG

1. Introduction. One task of metamathematics is to relate suggestive but nonelementary modeltheoretic concepts to more elementary prooftheoretic concepts, thereby opening up modeltheoretic problems to prooftheoretic methods of attack. Herbrand’s Theorem (see [8] or also [9],

Craig's Interpolation Lemma (1957)



William Craig in 1988
http://sophos.berkeley.edu/interpolations/

Lemma. First-order logic has the interpolation property. That is, if $A \implies B$ is valid, then a Craig interpolant exists.

A High-Level View of the Proof

"The intuitive idea for Craig's proof of the Interpolation Theorem rests on the completeness theorem for FOL, in the form of the equivalence of validity with provability in a suitable system of axioms and rules of inference. By "suitable" here is meant one in which there is a notion of a direct proof for which if Phi implies Psi is provable then there is a direct proof of Psi from Phi. One would expect that in such a proof, the relation symbols of Phi that are in Psi would not disappear in the middle. Such systems were devised by Herbrand (1930) and Gentzen (1934); Hilbert-style systems enlarged by the axioms and rules of the epsilon-calculus can also serve this purpose.

Feferman, *Harmonious Logic: Craig's Interpolation Theorem and Its Descendants,* 2008

Another Perspective on the Interpolation Theorem

"Important results have many things to say. At first sight, the Interpolation Theorem of Craig (1957) seems a rather technical result for connoisseurs inside logical meta-theory. But over the past decades, its broader importance has become clear from many angles. In this paper, I discuss my own current favourite views of interpolation: no attempt is made at being fair or representative. First, I discuss the ***entanglement of inference and vocabulary*** *that is crucial to interpolation. Next, I move to the role of interpolants in facilitating* ***generalized inference across different models****. Then I raise the perhaps surprising issue of 'what is the right formulation of Craig's Theorem?', high-lighting the existence of non-trivial options in formulating meta-theorems.*

...

Finally, I discuss the 'end of history'. Craig's Theorem is about the last significant property of first-order logic that has come to light. Is there something deeper going on here, and if so, can we prove it?"

-- Johan van Benthem, *The Many Faces of Interpolation,* 2008

Interpolation In Mathematical Logic

- Simpler proofs of known properties: Beth definability, Robinson’s theorem.
- Interpolant structure: Lyndon Interpolation theorems (1959).
- Preservation under homomorphisms (connections to finite-model theory).

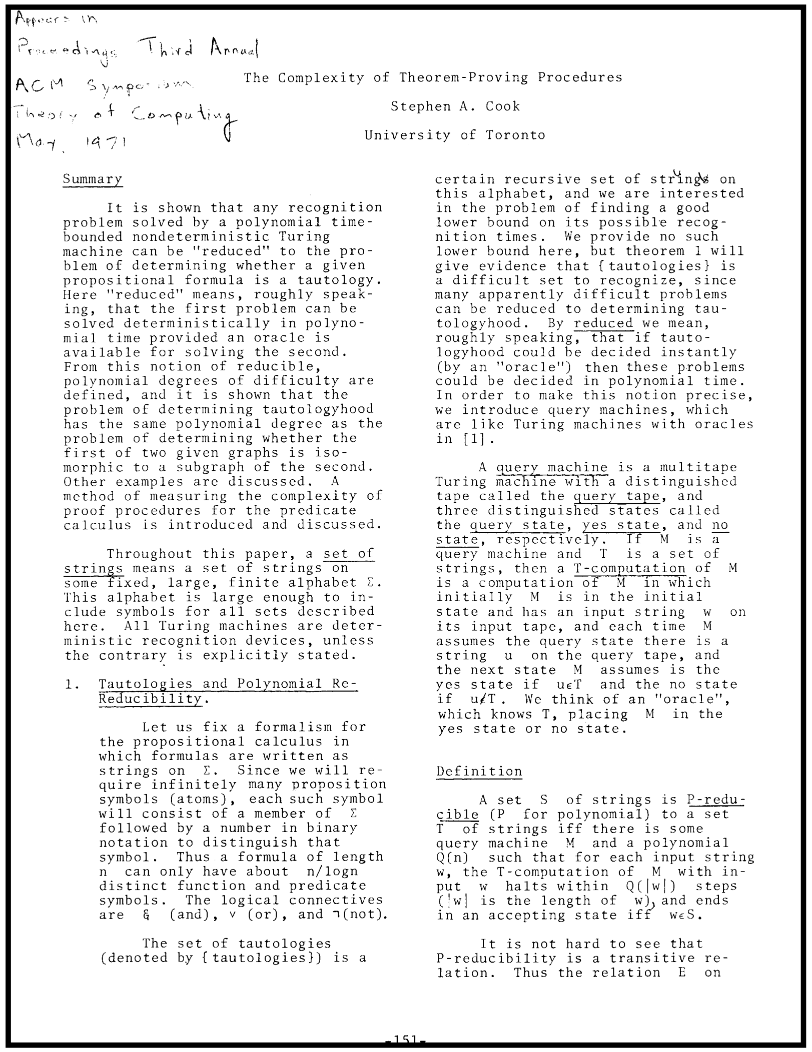
- Many-sorted and Infinitary logics: Feferman ’68, ’74, Lopez-Escobar ’65, Barwise ’69, Stern ’75, Otto ’00.
- Model theoretic characterizations: Makowsky ’85 for a survey.
- Amalgamation and algebraic characterization

- Guarded fragment: Hoogland, Marx, Otto ’00
- Modal and fixed point logics: Ten Cate ’05,
- Uniform interpolation: Pitt ’92, Visser ’96, d’Agostino, Hollenberg ’00

Interpolation In Complexity and Proof Theory

1957	1960	1970	1980	1990	2000	2010

1971	1971, Cook. The Complexity of Theorem Proving Procedures



Appears in Proceedings Third Annual ACM Symposium Theory of Computing May, 1971

The Complexity of Theorem-Proving Procedures

Stephen A. Cook

University of Toronto

Summary

It is shown that any recognition problem solved by a polynomial time-bounded nondeterministic Turing machine can be "reduced" to the problem of determining whether a given propositional formula is a tautology. Here "reduced" means, roughly speaking, that the first problem can be solved deterministically in polynomial time provided an oracle is available for solving the second. From this notion of reducible, polynomial degrees of difficulty are defined, and it is shown that the problem of determining tautologyhood has the same polynomial degree as the problem of determining whether the first of two given graphs is isomorphic to a subgraph of the second. Other examples are discussed. A method of measuring the complexity of proof procedures for the predicate calculus is introduced and discussed.

Throughout this paper, a set of strings means a set of strings on some fixed, large, finite alphabet Σ. This alphabet is large enough to include symbols for all sets described here. All Turing machines are deterministic recognition devices, unless the contrary is explicitly stated.

1. Tautologies and Polynomial Re-Reducibility.

Let us fix a formalism for the propositional calculus in which formulas are written as strings on Σ. Since we will require infinitely many proposition symbols (atoms), each such symbol will consist of a member of Σ followed by a number in binary notation to distinguish that symbol. Thus a formula of length n can only have about n/logn distinct function and predicate symbols. The logical connectives are & (and), ∨ (or), and ¬(not).

The set of tautologies (denoted by {tautologies}) is a certain recursive set of strings on this alphabet, and we are interested in the problem of finding a good lower bound on its possible recognition times. We provide no such lower bound here, but theorem 1 will give evidence that {tautologies} is a difficult set to recognize, since many apparently difficult problems can be reduced to determining tautologyhood. By reduced we mean, roughly speaking, that if tautologyhood could be decided instantly (by an "oracle") then these problems could be decided in polynomial time. In order to make this notion precise, we introduce query machines, which are like Turing machines with oracles in [1].

A query machine is a multitape Turing machine with a distinguished tape called the query tape, and three distinguished states called the query state, yes state, and no state, respectively. If M is a query machine and T is a set of strings, then a T-computation of M is a computation of M in which initially M is in the initial state and has an input string w on its input tape, and each time M assumes the query state there is a string u on the query tape, and the next state M assumes is the yes state if u∈T and the no state if u∉T. We think of an "oracle", which knows T, placing M in the yes state or no state.

Definition

A set S of strings is P-reducible (P for polynomial) to a set T of strings iff there is some query machine M and a polynomial Q(n) such that for each input string w, the T-computation of M with input w halts within Q(|w|) steps (|w| is the length of w), and ends in an accepting state iff w∈S.

It is not hard to see that P-reducibility is a transitive relation. Thus the relation E on

Proof Content. For any language $A \in$ NP and $n \in \mathbb{N}$, one can construct in polynomial time a formula

$$F_n(x_1, \ldots, x_n, y_1, \ldots, y_{p(n)})$$

in propositional logic such that for all $x \in \{0,1\}^n$

$$x \in A \iff \exists y. F_n(x, y) = true$$

Wednesday, March 30, 16

Interpolation and Complexity Theory

1971	1971, Cook. The Complexity of Theorem Proving Procedures
1982	Mundici, NP and Craig's Interpolation Theorem (pub. 1984)
1983	Mundici, A Lower bound for the complexity of Craig's Interpolants in Sentential Logic

Theorem. (Mundici, 1982) At least one of the following is true.

1. P $=$ NP.
2. NP $\neq$ coNP.
3. For F and G in propositional logic, such that $F \implies G$, an interpolant is not computable in time polynomial in the size of F and G.

Interpolation and (Proof) Complexity Theory

1957	1960	1970	1980	1990	2000	2010

1997	Jan Krajíček, Interpolation theorems, lower bounds for proof systems, and independence results for bounded arithmetic.
	Pudlák, Lower Bounds for Resolution and Cutting Plane Proofs and Monotone Computations

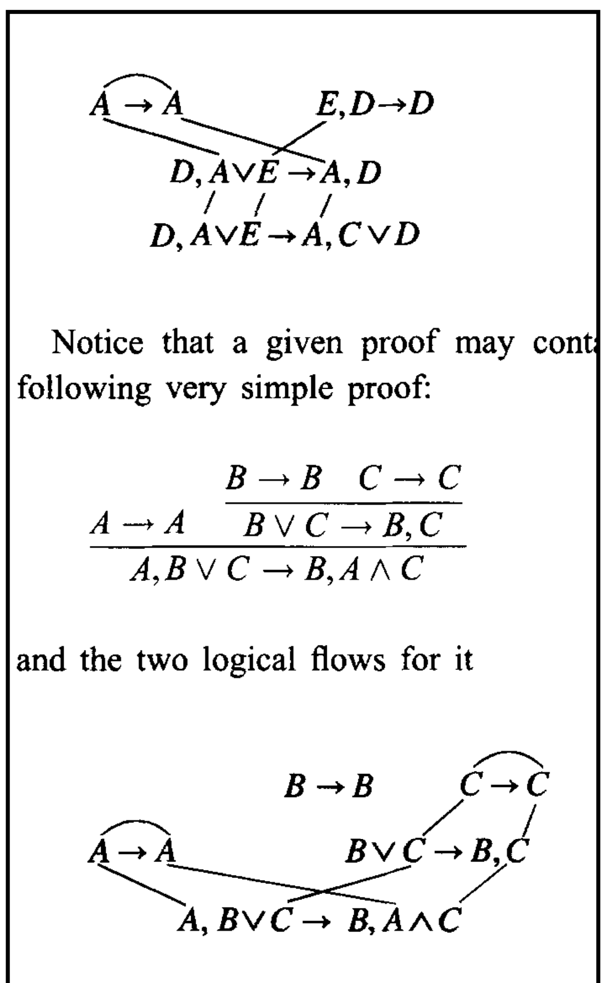
A proof system $\vdash$ has feasible interpolation if, whenever there is a short refutation of $A \wedge B$, the interpolant is computable in polynomial time in the size of the proof.

Lemma If there is a resolution refutation of size n for a formula $A \wedge B$, there is an interpolant of circuit size $3n$ that is computable in time n.

Interpolation and (Proof) Complexity Theory

1957	1960	1970	1980	1990	2000	2010

1997	Carbone, Interpolants, Cut Elimination and Flow Graphs for the Propositional Calculus



$A \to A \qquad E, D \to D$

$D, A \vee E \to A, D$

$D, A \vee E \to A, C \vee D$

Notice that a given proof may cont
following very simple proof:

$$\dfrac{A \to A \qquad \dfrac{B \to B \quad C \to C}{B \vee C \to B, C}}{A, B \vee C \to B, A \wedge C}$$

and the two logical flows for it

$B \to B \qquad C \to C$

$A \to A \qquad B \vee C \to B, C$

$A, B \vee C \to B, A \wedge C$

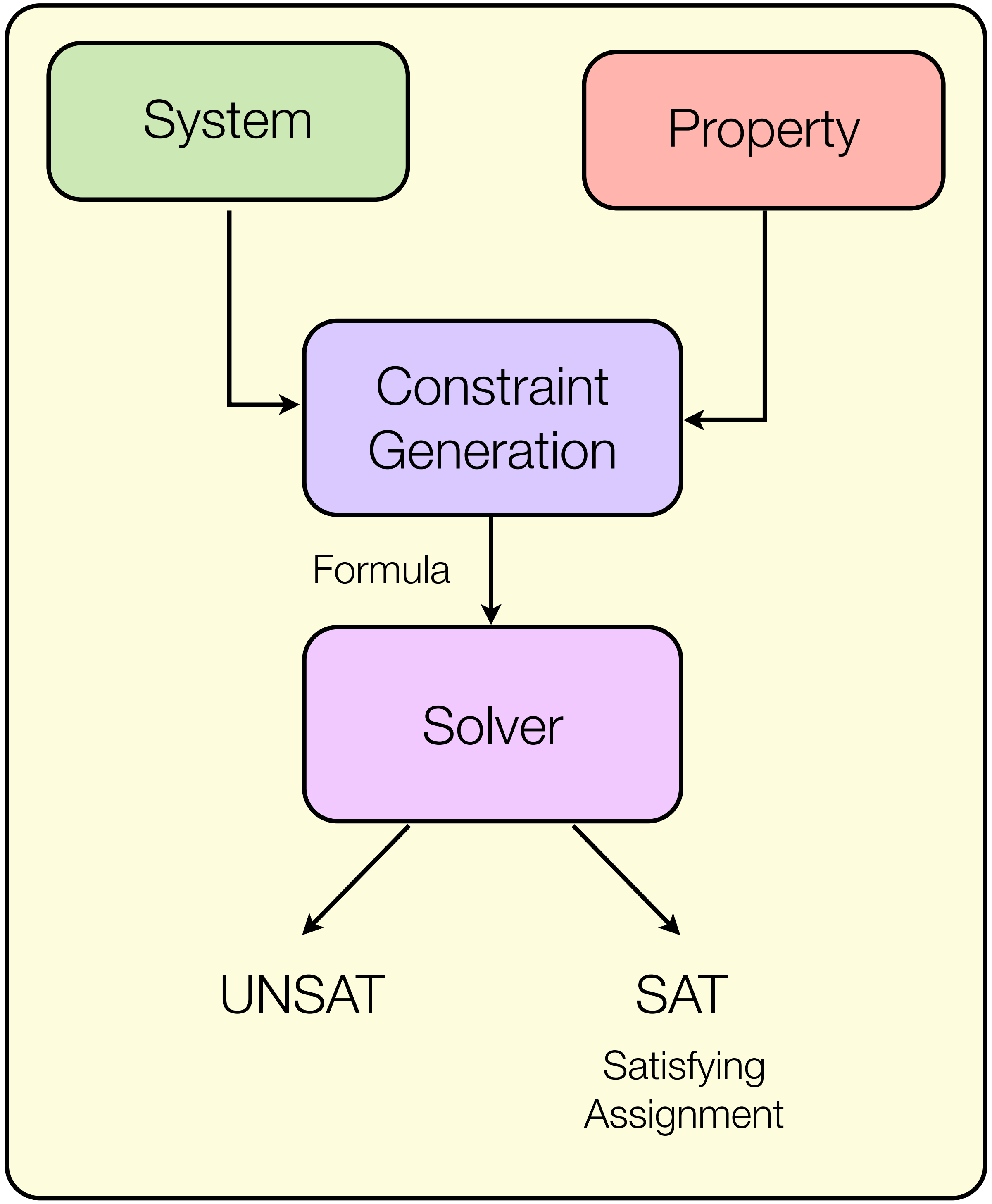
Combinatorial description of how information flows in a proof. Interpolants eliminate certain flows and preserve others. The “relevant” information is preserved.

Interpolants in Automated Reasoning

1995	Huang, Constructing Craig Interpolation Formulas. (OTTER)
2001	Amir, McIlraith, Partition-Based Logical Reasoning.
2003	McMillan, Interpolation and SAT-Based Model Checking.
2004	Henziger, Jhala,Majumdar,McMillan, Abstractions from Proofs
2005	McMillan, An Interpolating Theorem Prover

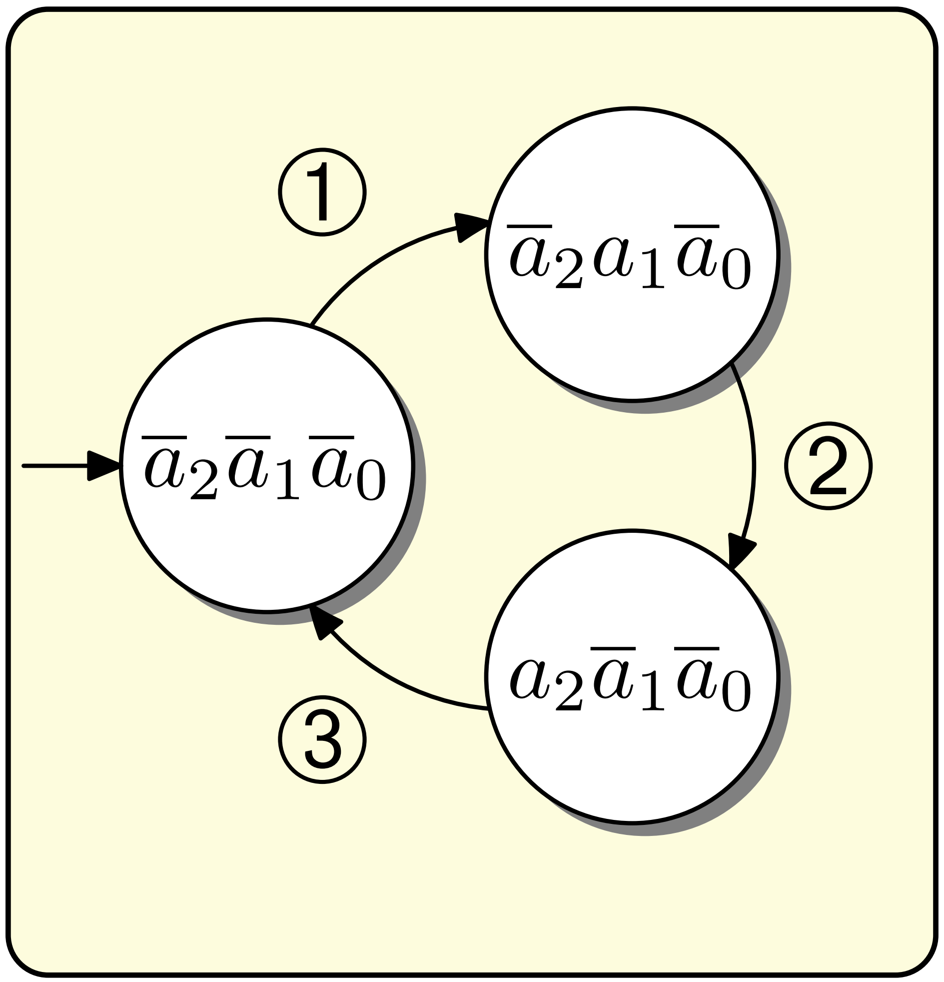
1	A Brief History of Interpolation
2	Verification with Interpolants
3	Interpolant Construction
4	Further Reading and Research

Systems Analysis with SAT/SMT Solvers



- Bounded Model Checking and Symbolic Execution generate formulae encoding bounded executions.
- Can we generate invariants?
- Can we explore deeper executions without running out of memory?
- Can we avoid exploring redundant system behaviours?

A Simple Binary Counter



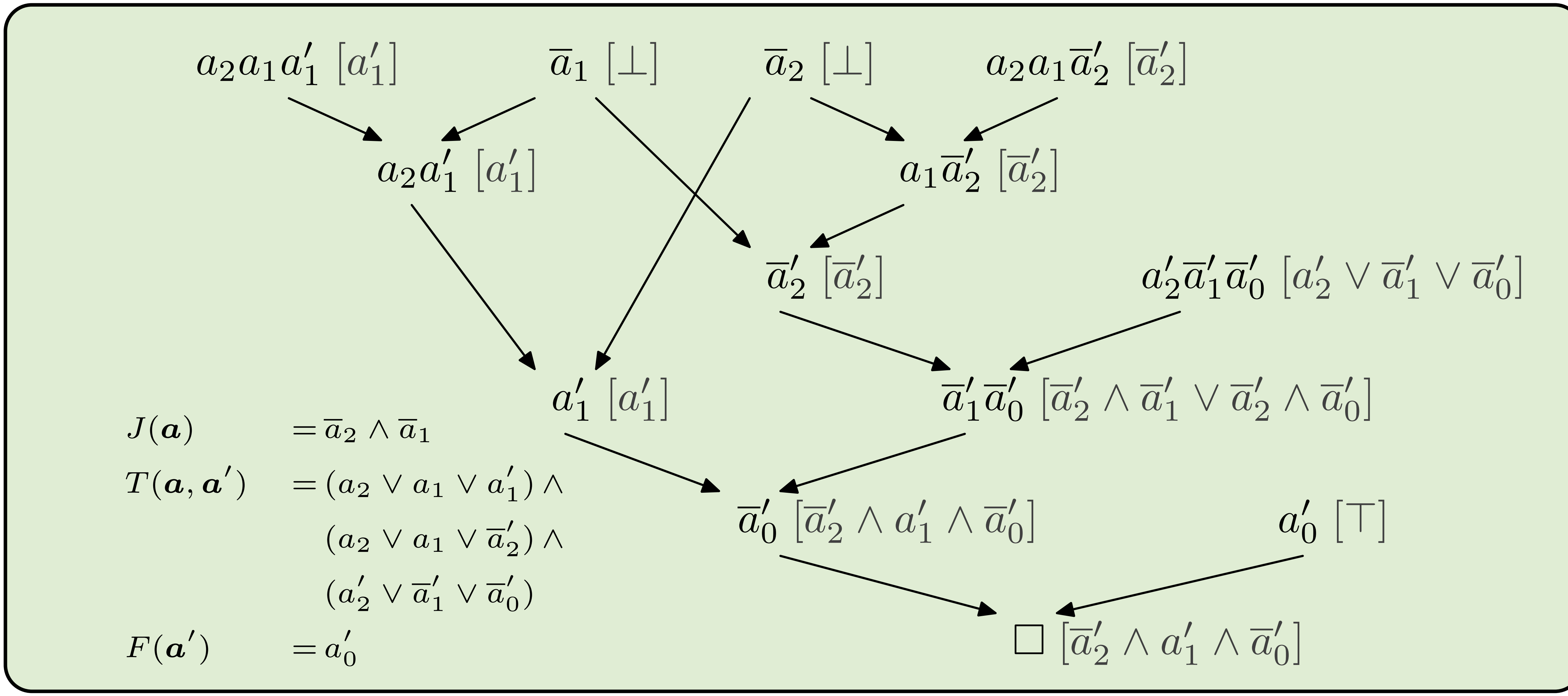
Initial State

$$
\begin{aligned}
J(\boldsymbol{a}) &\stackrel{\text{def}}{=} (\overline{a}_2 \wedge \overline{a}_1 \wedge \overline{a}_0) \\
T(\boldsymbol{a}, \boldsymbol{a}') &\stackrel{\text{def}}{=} (\overline{a}_2 \wedge \overline{a}_1 \Rightarrow \overline{a}'_2 \wedge a'_1) \wedge && \textcircled{1} \\
&\quad\; (\overline{a}_2 \wedge a_1 \Rightarrow a'_2 \wedge \overline{a}'_1) \wedge && \textcircled{2} \\
&\quad\; (a_2 \wedge \overline{a}_1 \Rightarrow \overline{a}'_2 \wedge \overline{a}'_1) \wedge && \textcircled{3} \\
&\quad\; (a_2 \wedge a_1 \Rightarrow a'_2 \wedge a'_1) \wedge && \textcircled{4} \\
&\quad\; (\overline{a}'_2 \wedge \overline{a}'_1 \vee \overline{a}'_2 \wedge a'_1 \vee a'_2 \wedge \overline{a}'_1) \Rightarrow \overline{a}'_0 \\
F(\boldsymbol{a}) &\stackrel{\text{def}}{=} a_0 \wedge (\overline{a}_1 \vee \overline{a_2})
\end{aligned}
$$

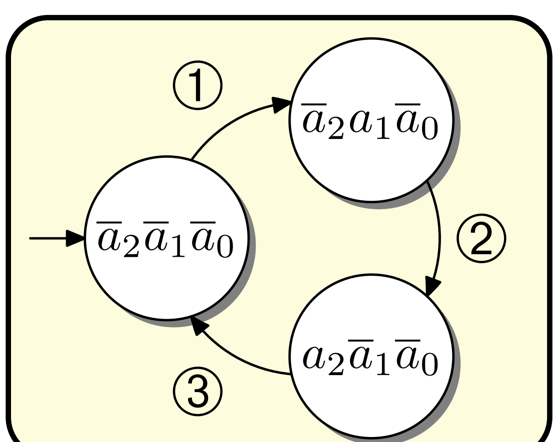
Transition Relation

Reachability Condition

A Refutation and Its Interpolant

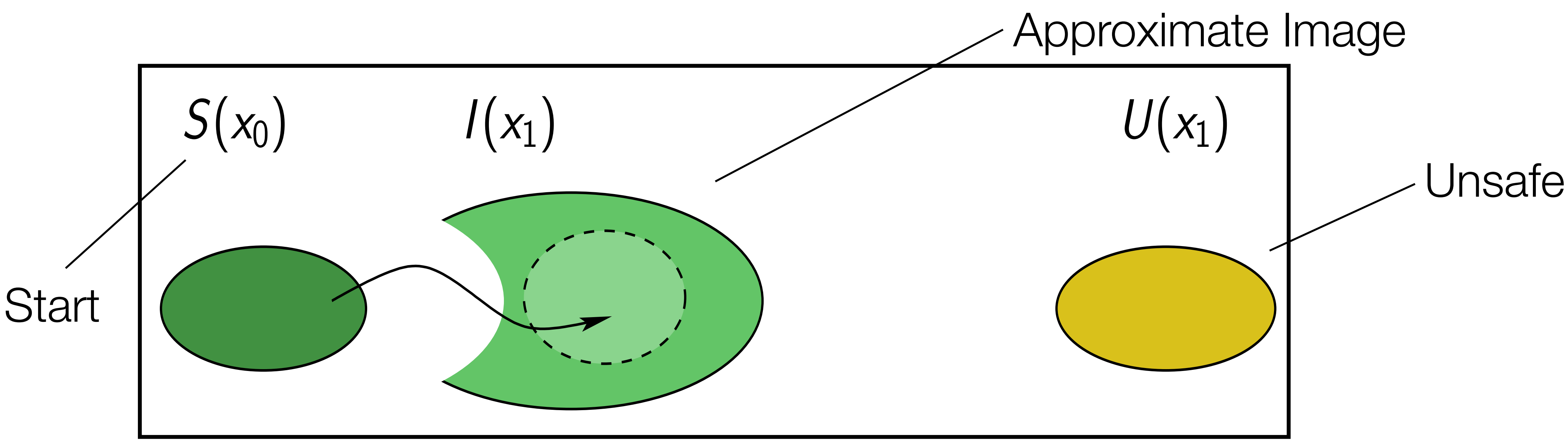


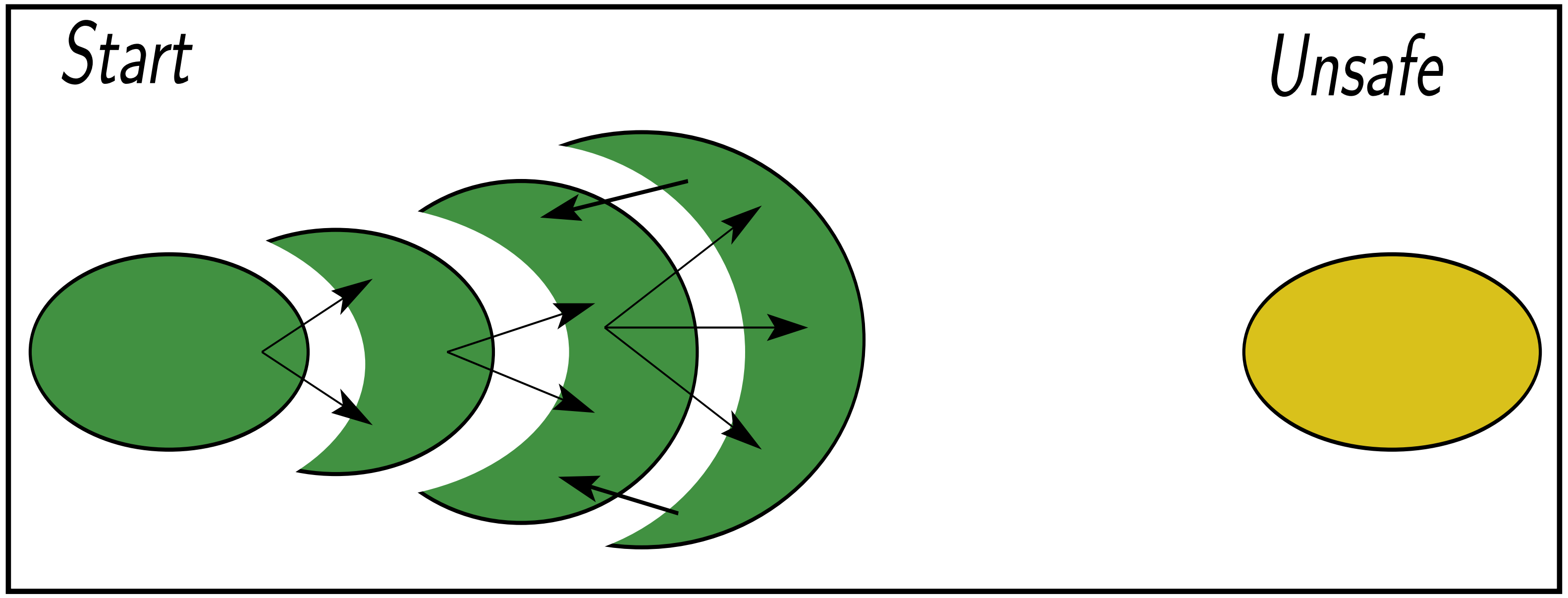
$$
\begin{aligned}
J(\boldsymbol{a}) &= \overline{a}_2 \wedge \overline{a}_1 \\
T(\boldsymbol{a}, \boldsymbol{a}') &= (a_2 \vee a_1 \vee a_1') \wedge \\
&\quad (a_2 \vee a_1 \vee \overline{a}_2') \wedge \\
&\quad (a_2' \vee \overline{a}_1' \vee \overline{a}_0') \\
F(\boldsymbol{a}') &= a_0'
\end{aligned}
$$



The interpolant, in this case is the image. In general, interpolants for an appropriately constructed formula are overapproximations of images.

Reachability with Interpolants

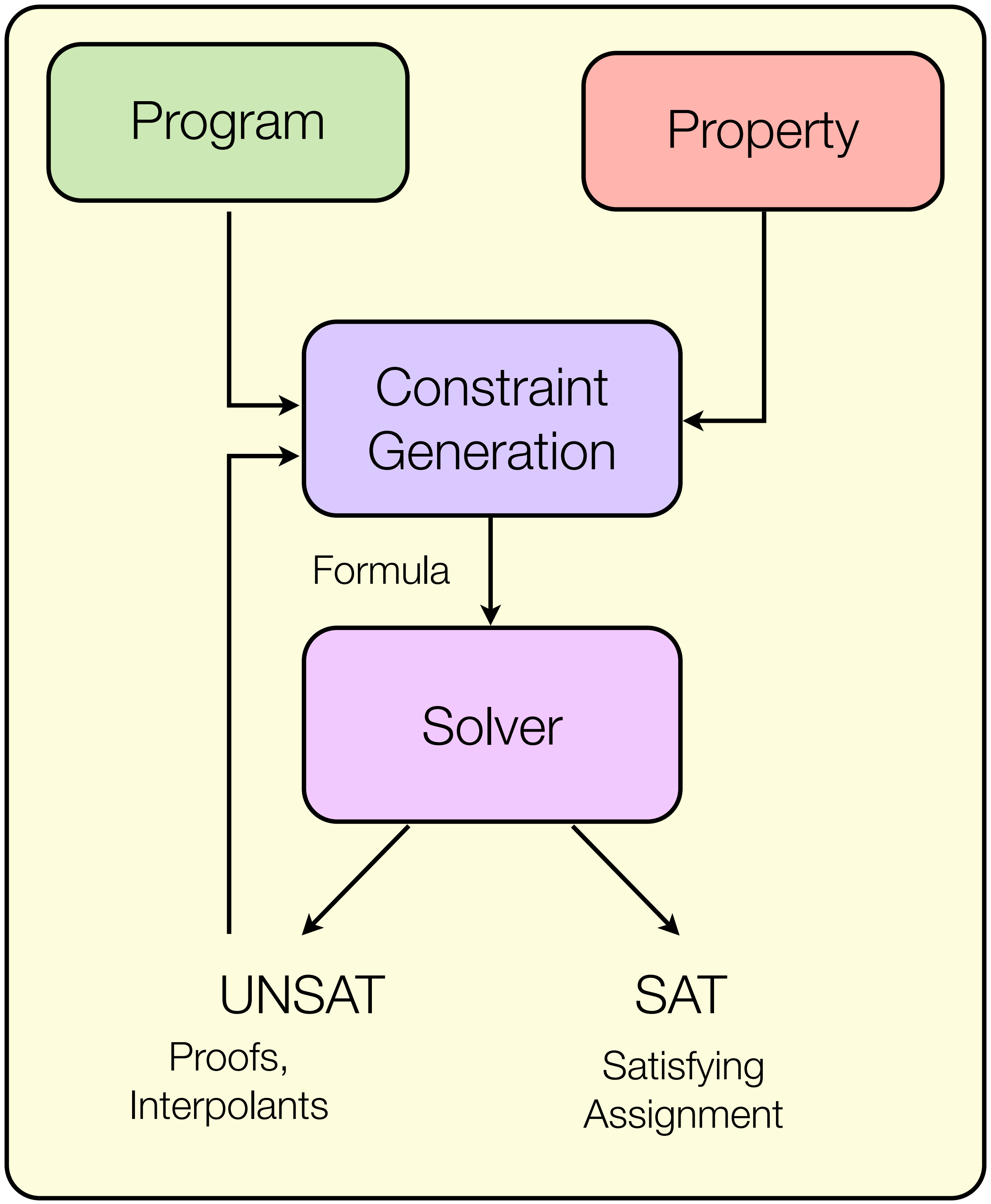




Algorithm: Compute images till a fixed point is reached.

Image of Qx: $\exists x : Q(x) \wedge T(x, x')$

Interpolation Slogans



- A poor person's quantifier elimination
- A separator between two regions of a search space
- A summary of why a bounded-property holds occurs
- An approximate image operator
- A relevance heuristic that articulates the core reason for a proof

Bounded Execution as a Formula

```
   int x = i;
int y = j;
while (foo()) {
// Code that does not
// modify 'x' or 'y'.
    x = y + 1;
    y = x + 1;
}
if (i = j && x <= 10)
 assert(y <= 10);
```

```
int x0= i;
int y0 = j;
x1 = y0 + 1
y1 = x0 + 1;
x2 = y1 + 1
y2 = x1 + 1;

x3 = y2 + 1
y3 = x2 + 1;
if (i = j &&
    x3 <= 10) {
  if (y3 > 10)
      Err:// ERROR
REACHED
}
```

Bounded Execution and Interpolants

```
int x0= i;
int y0 = j;
x1 = y0 + 1
y1 = x0 + 1;
x2 = y1 + 1
y2 = x1 + 1;
```

A

```
x3 = y2 + 1
y3 = x2 + 1;
if (i = j &&
    x3 <= 10) {
  if (y3 > 10)
      Err:// ERROR
REACHED
}
```

B

$$x_2 = i + 2 \wedge y_2 = j + 2$$

- Symbolic representation of the states reachable after two iterations
- Image computation for program statements typically requires quantifier elimination in that theory.

Another Interpolant

```
   int x = i;
int y = j;
while (foo()) {
// Code that does not
// modify 'x' or 'y'.
    x = y + 1;
    y = x + 1;
}
if (i = j && x <= 10)
 assert(y <= 10);
```

$$i = j \implies x_2 \leq y_2$$

- Potential loop invariant
- Invariant computation often requires fixed point computation, quantifier elimination, or even both.

A Space of Interpolants

```
int x0= i;
int y0 = j;
x1 = y0 + 1
y1 = x0 + 1;
x2 = y1 + 1
y2 = x1 + 1;
x3 = y2 + 1
y3 = x2 + 1;
if (i = j &&
    x3 <= 10) {
  if (y3 > 10)
      Err:// ERROR
}
```

A

B

$$i = j \implies x_2 \leq y_2$$

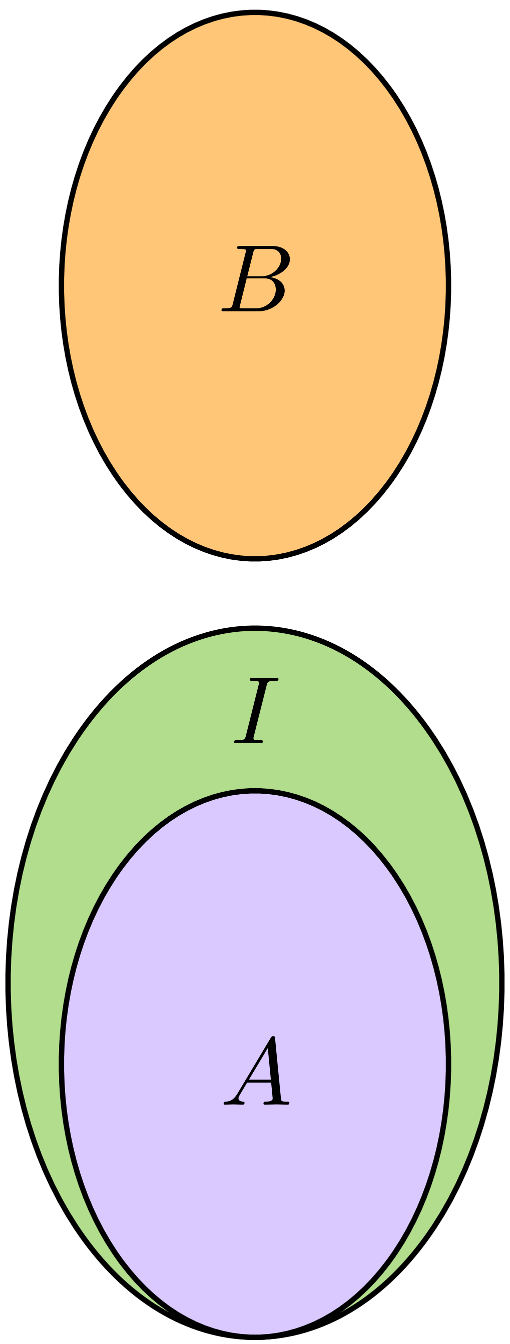
$$i = j \implies y_2 \leq x_2$$

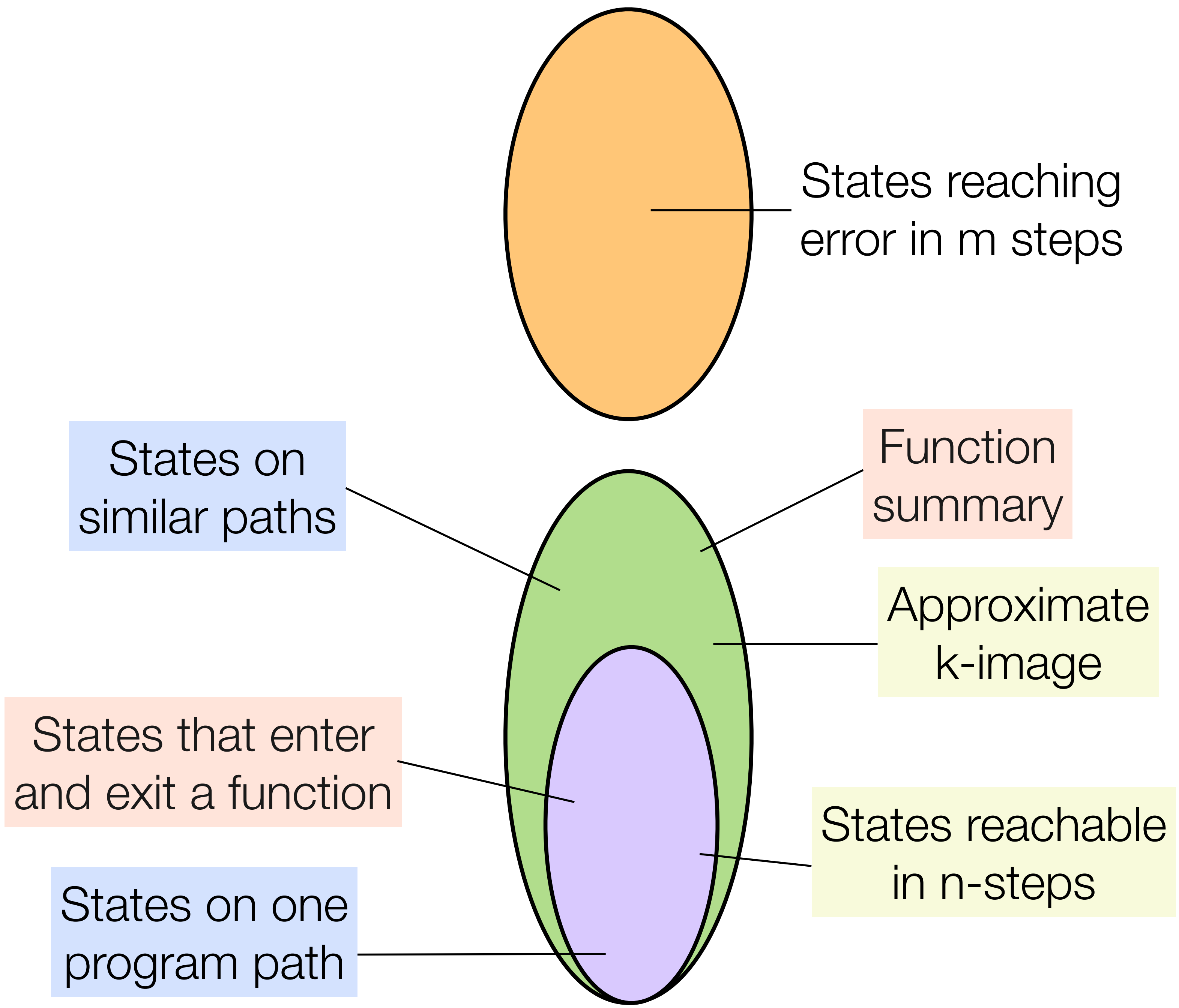
$$x_2 = i + 2 \wedge y_2 = j + 2$$

$$i = j \implies x_2 = y_2$$

- Multiple interpolants exist.
- They differ in size, logical strength, symbols, etc.
- The ideal one depends on the problem

Approximation of States with Interpolants

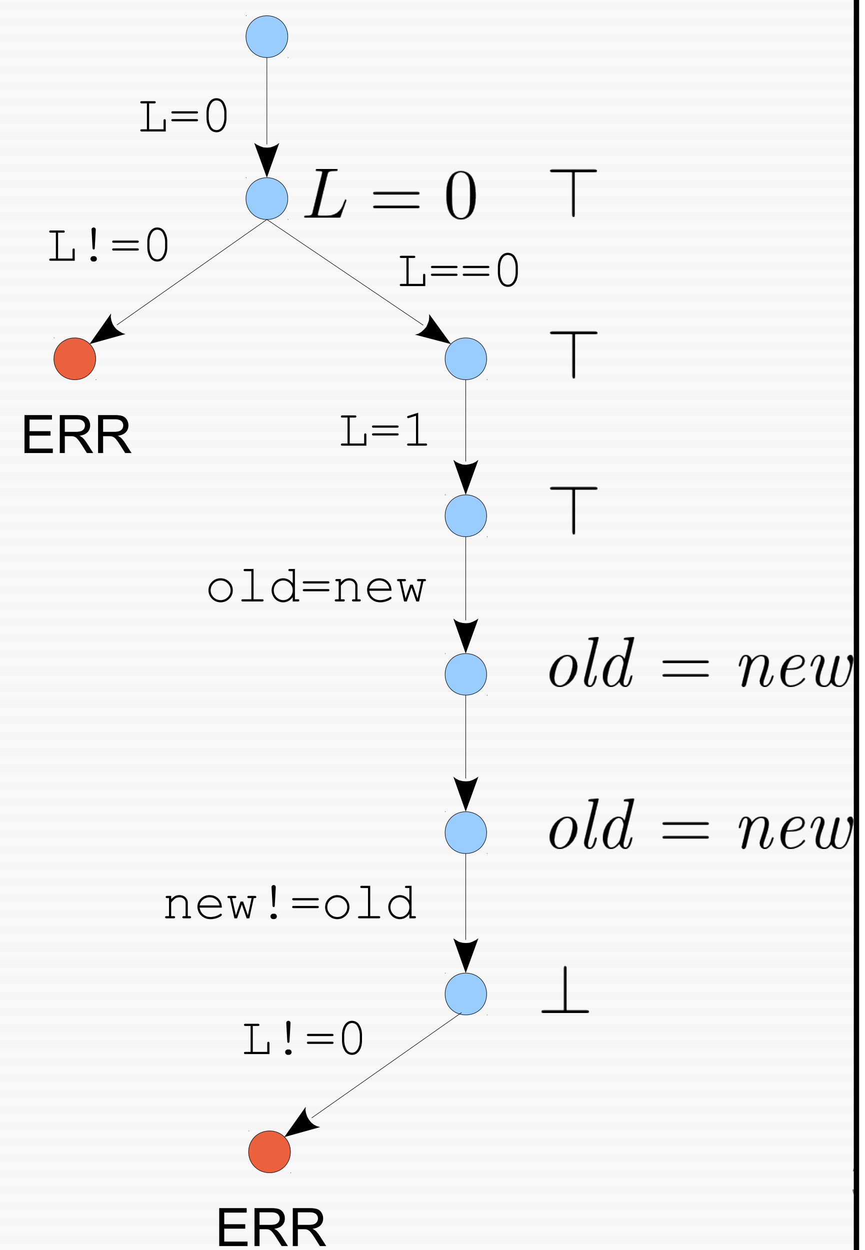




The challenge is to encode these constraints and respect the vocabulary condition

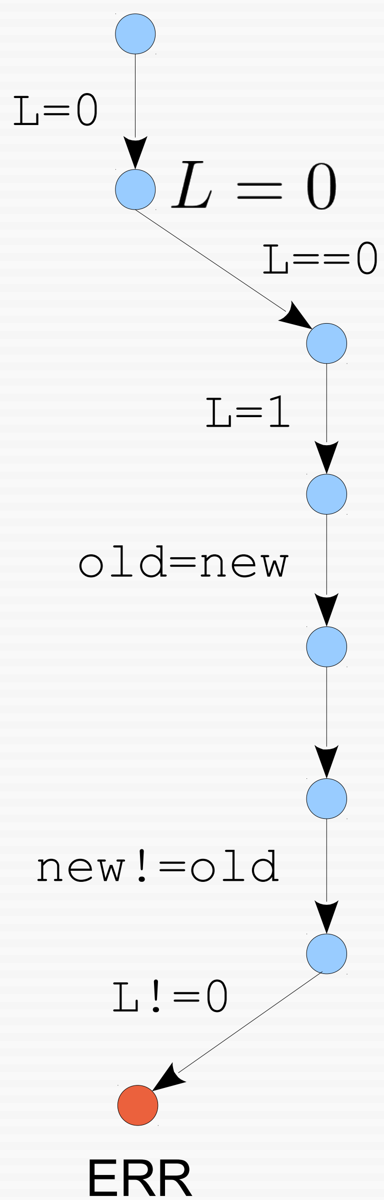
Abstract Reachability Tree Construction

```
L = 0;
do {
   assert(L==0);   } lock()
   L = 1;
   old = new;
   if (*){
      L = 0;       } unlock()
      new++;
   }
} while (new!=old);
```



Example: McMillan 2006, Graphic Ruemmer ’14

Sequence Interpolants from Reachability Tree

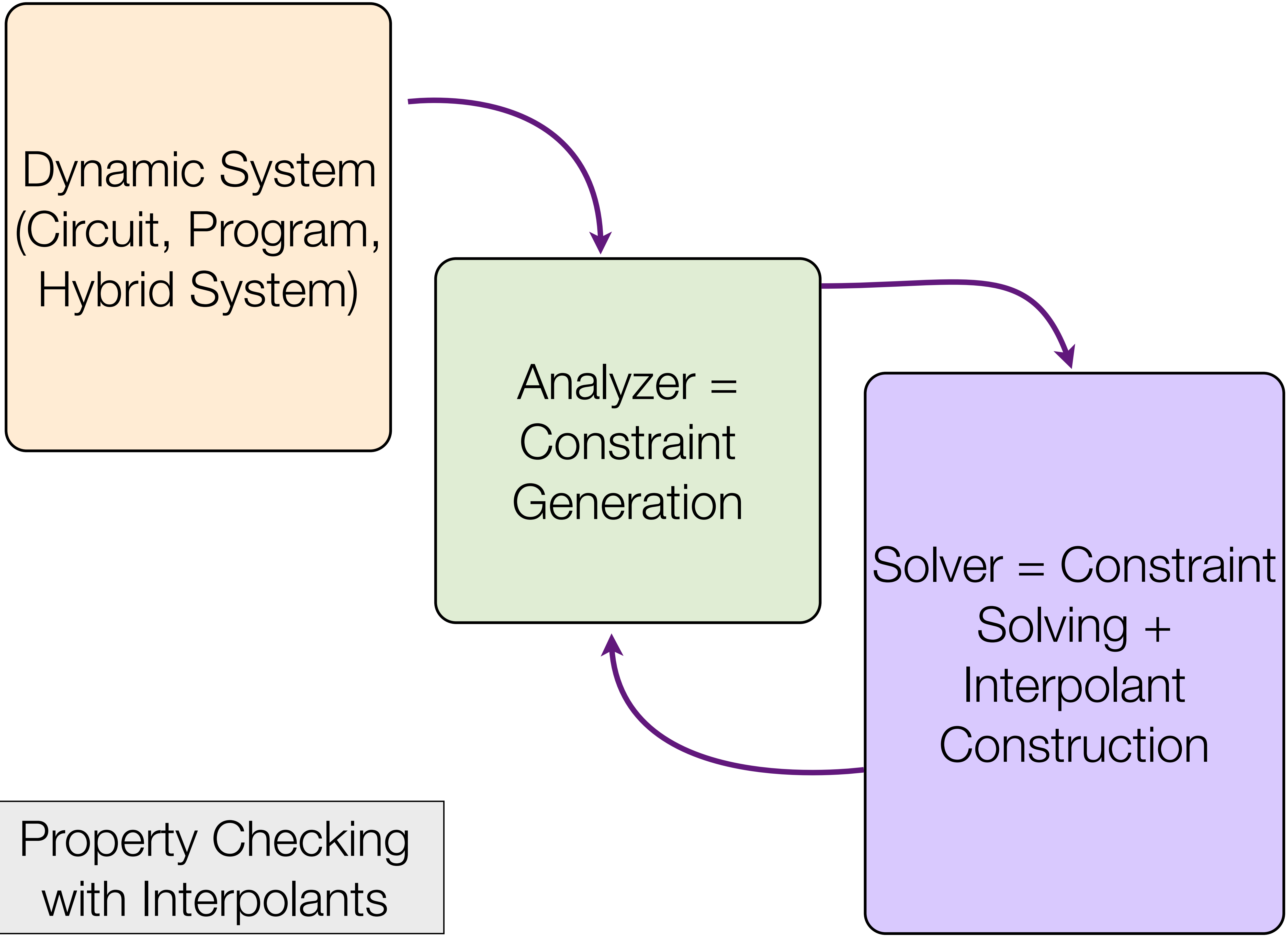


		$I_0 :$	$\top$
$T_1 :$	$L_0 = 0$	$I_1 :$	$\top$
$T_2 :$	$L_0 = 0$	$I_2 :$	$\top$
$T_3 :$	$L_1 = 1$	$I_3 :$	$\top$
$T_4 :$	$old_1 = new_0$	$I_4 :$	$old_1 = new_0$
$T_5 :$	$\top$	$I_5 :$	$old_1 = new_0$
$T_6 :$	$new_0 \neq old_1$	$I_6 :$	$\bot$
$T_7 :$	$L_1 \neq 0$	$I_7 :$	$\bot$

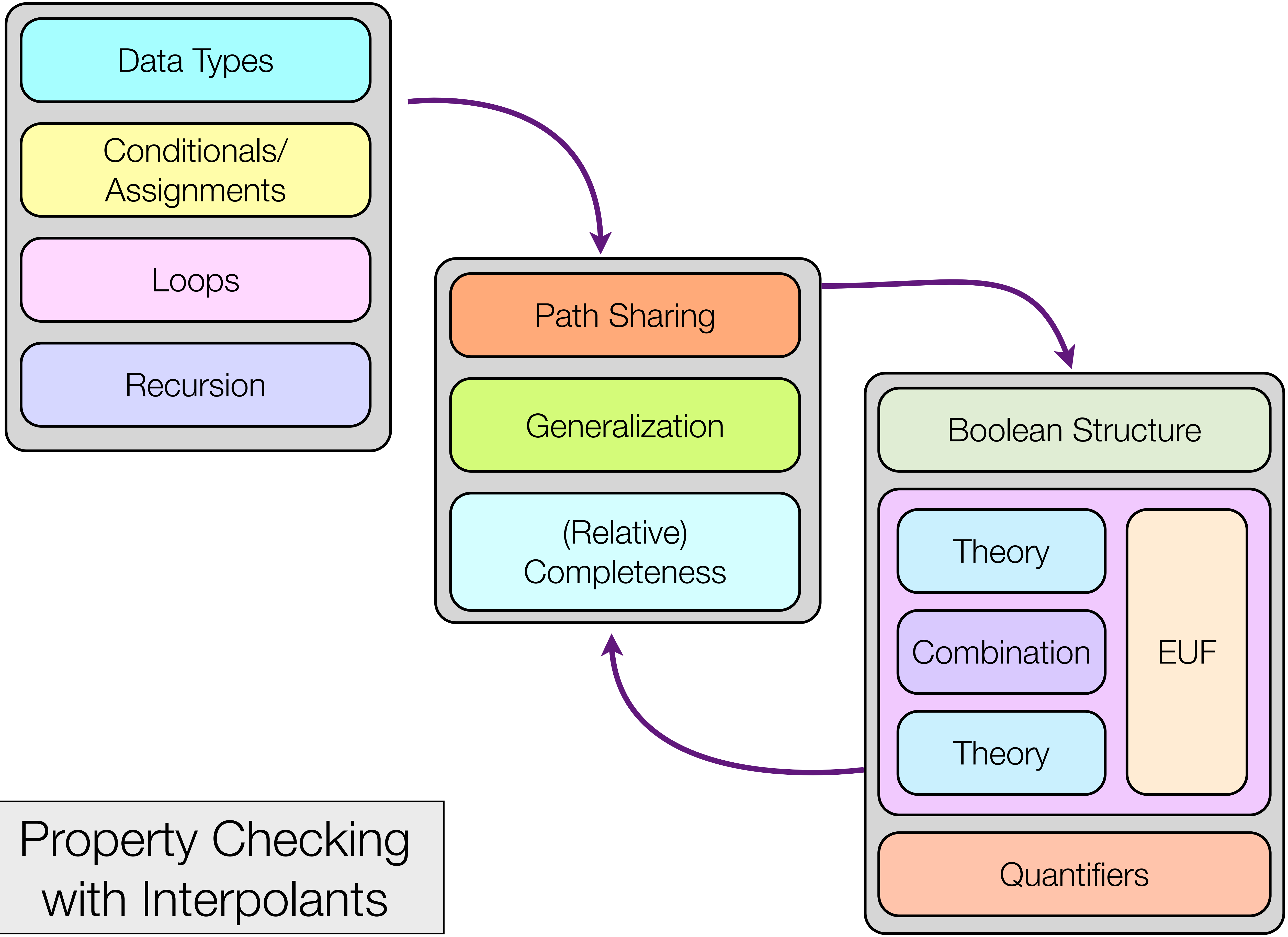
Example: McMillan 2006, Graphic Ruemmer '14

Abstract Reachability Construction

- Interpolants Decorate Positions on the Reachability Tree
- They denote state that are reached at those points
- A covering check is used to determine if all states at some location have been visited
- More complicated than predicate abstraction or fixed point computation due to non-monotonicity of interpolant construction

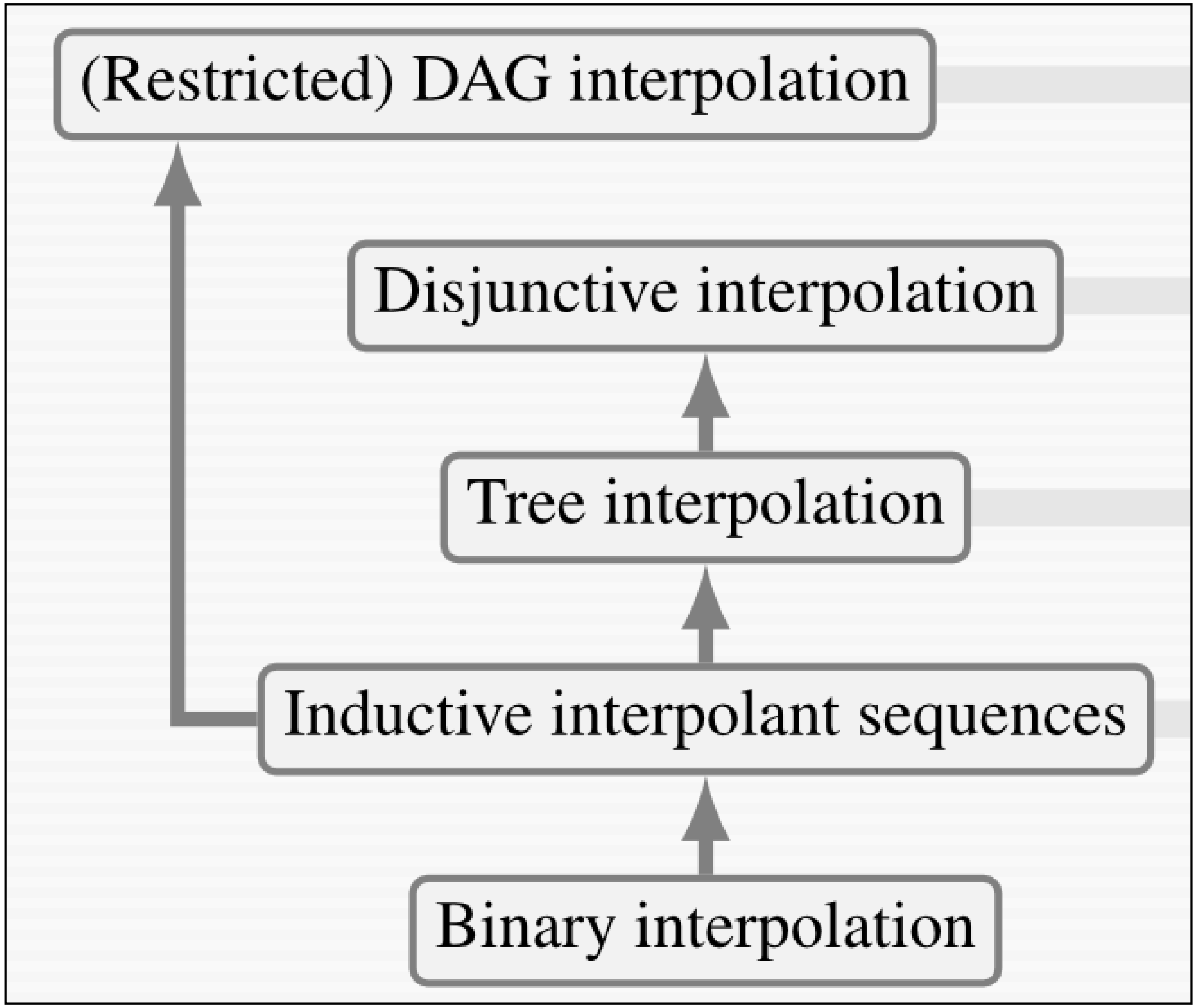


Property Checking
with Interpolants



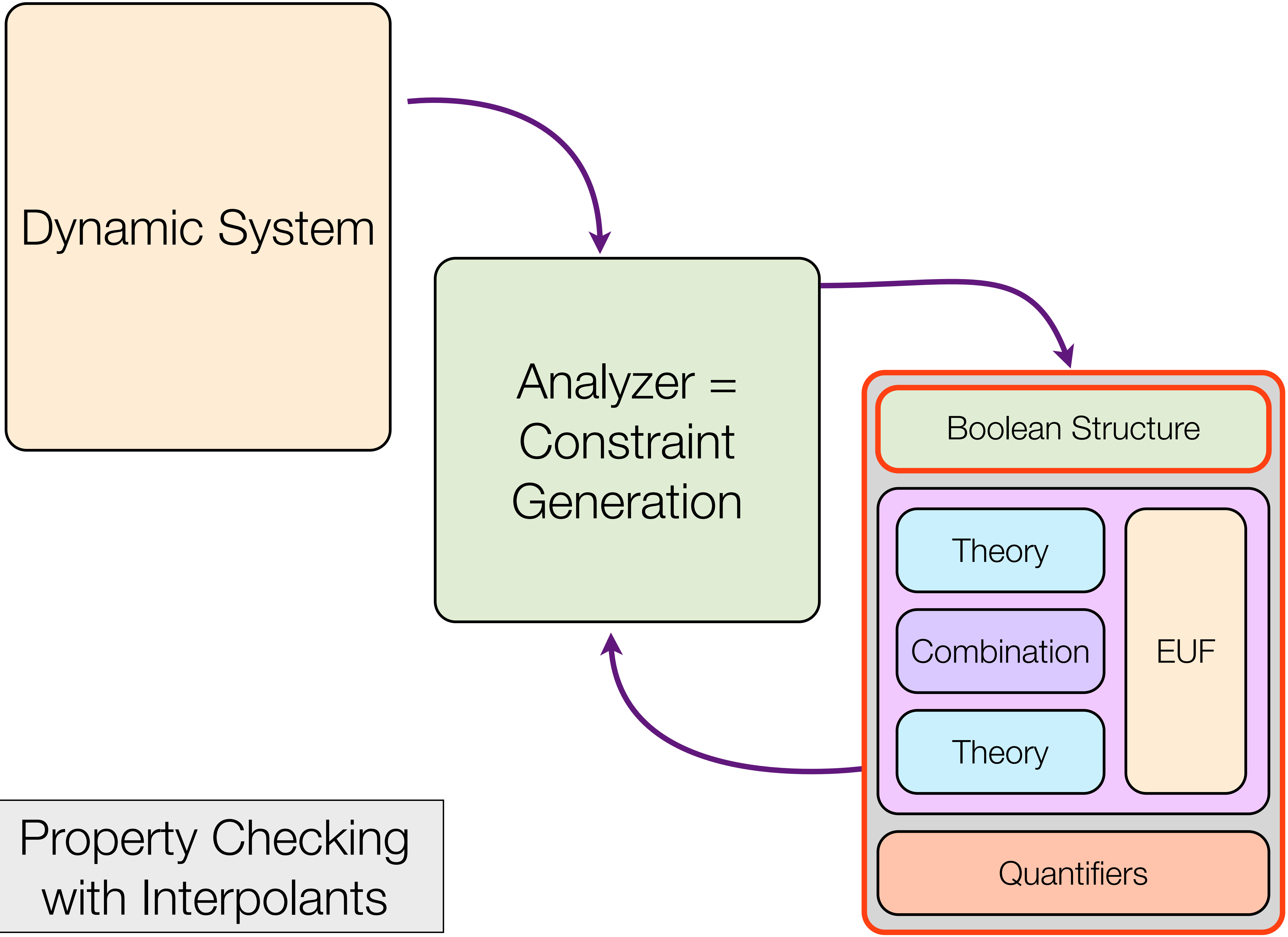
Property Checking with Interpolants

Generalizations of Classical Interpolants

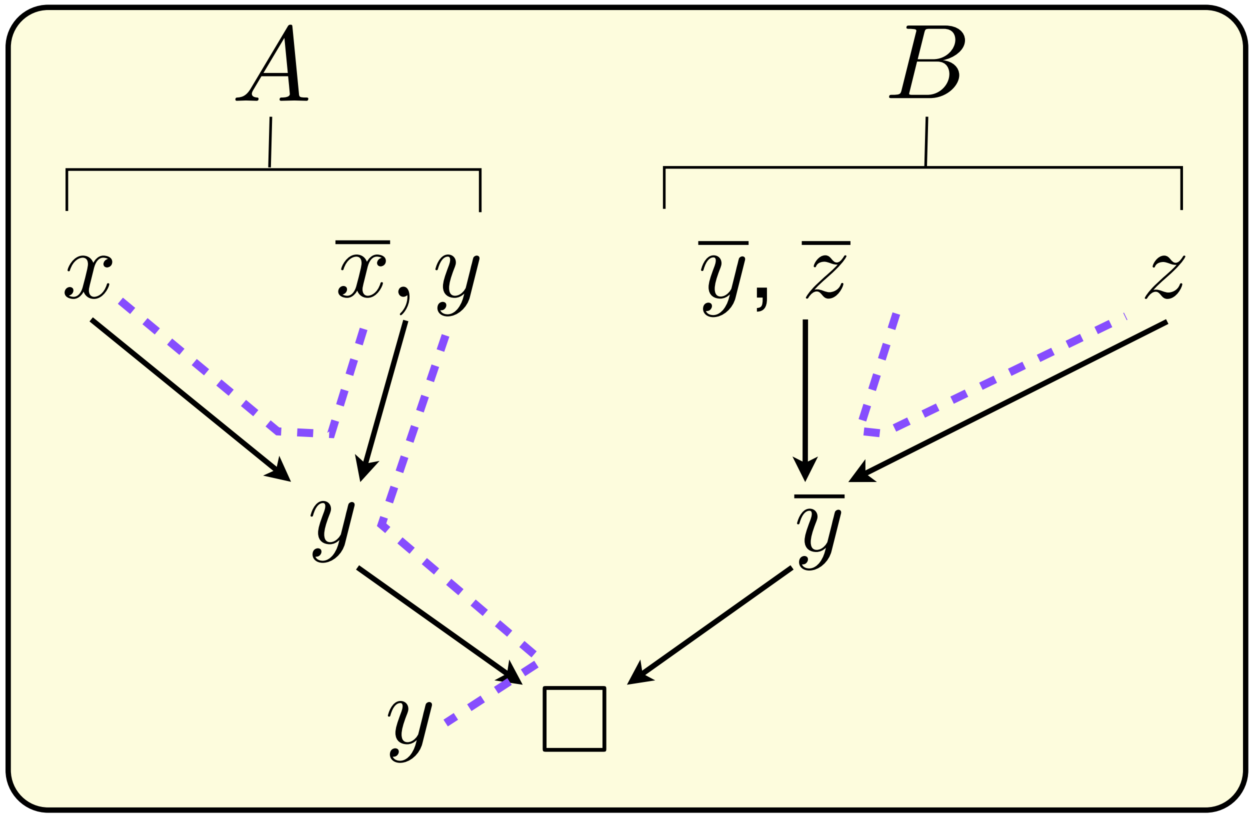


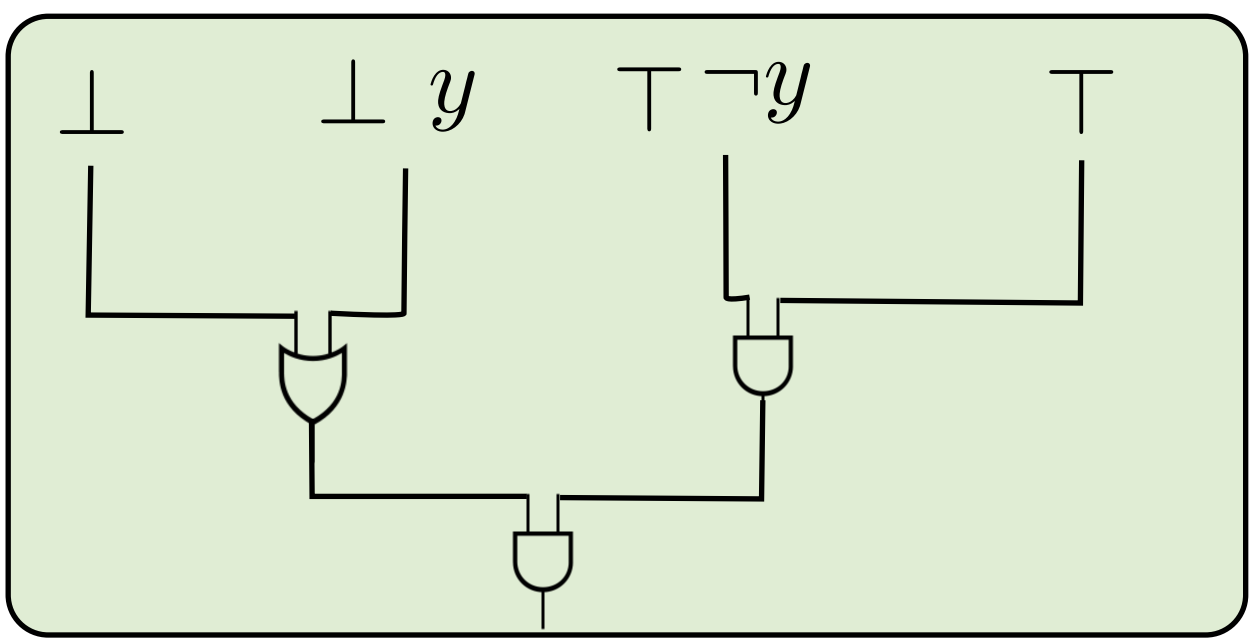
1	A Brief History of Interpolation
2	Verification with Interpolants
3	Interpolant Construction
4	Further Reading and Research

Property Checking with Interpolants



Flows in Resolution Proofs





- Literals flow around in the proof
- Literals with opposite polarity cancel each other
- Propositional interpolant construction can be viewed as controlling initial inputs and gating the flows using a circuit
- Want to let shared variables flow through the A-part and restrict flow from the B-part.

Interpolating Proof Rules

Split rules based on vocabulary

A-Hyp $\quad \dfrac{}{C \quad [\{\ell \mid var(\ell) \in var(B)\}]} \quad (C \in A)$

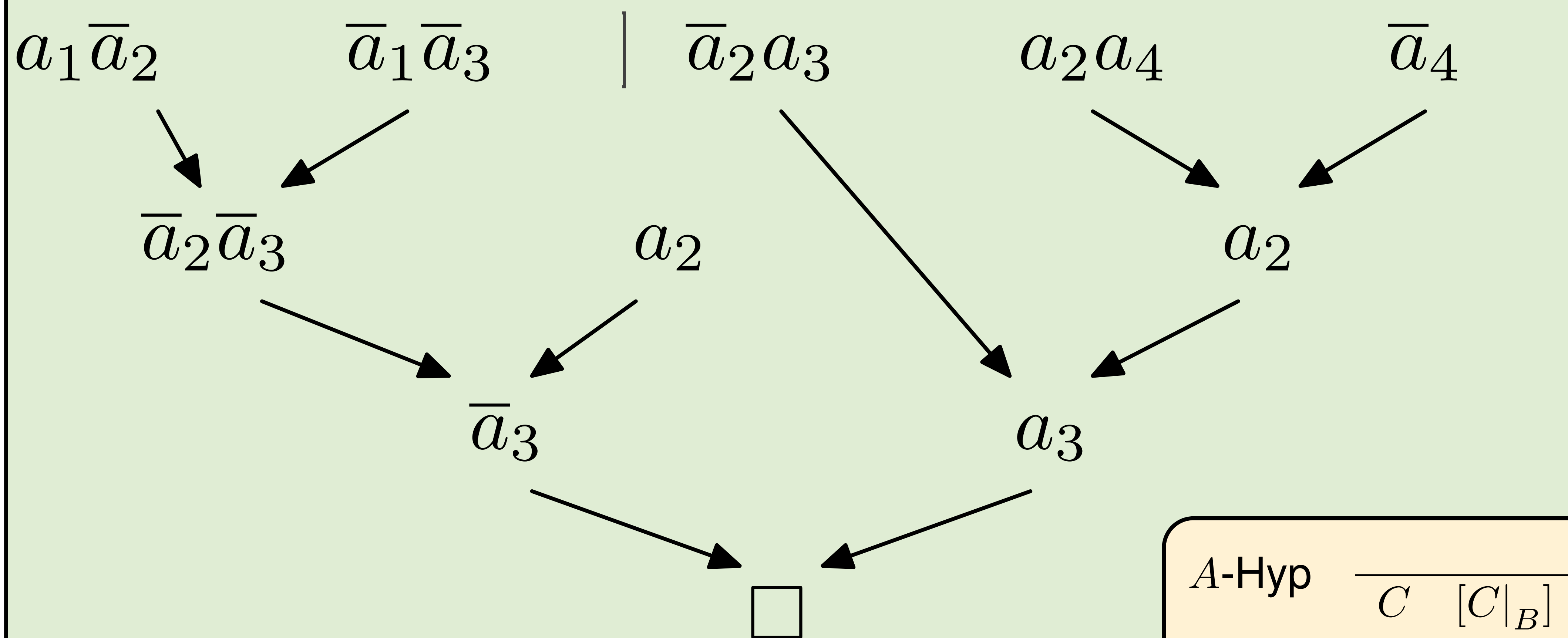
B-Hyp $\quad \dfrac{}{C \quad [\top]} \quad (C \in B)$

A-Res $\quad \dfrac{C \vee x\ [I_1] \qquad \overline{x} \vee D\ [I_2]}{C \vee D \quad [I_1 \vee I_2]} \quad (x \in var(A) \setminus var(B))$

B-Res $\quad \dfrac{C \vee x\ \boxed{[I_1]} \qquad \overline{x} \vee D\ \boxed{[I_2]}}{C \vee D \quad \boxed{[I_1 \wedge I_2]}} \quad (x \in var(B))$

Annotate formulae with *Partial Interpolants*

Applying Interpolating Proof Rules



A-Hyp $\quad \dfrac{}{C \quad [C|_B]}$

B-Hyp $\quad \dfrac{}{C \quad [\top]}$

A-Res $\quad \dfrac{C \vee x\ [I_1] \qquad \overline{x} \vee D\ [I_2]}{C \vee D \quad [I_1 \vee I_2]}$

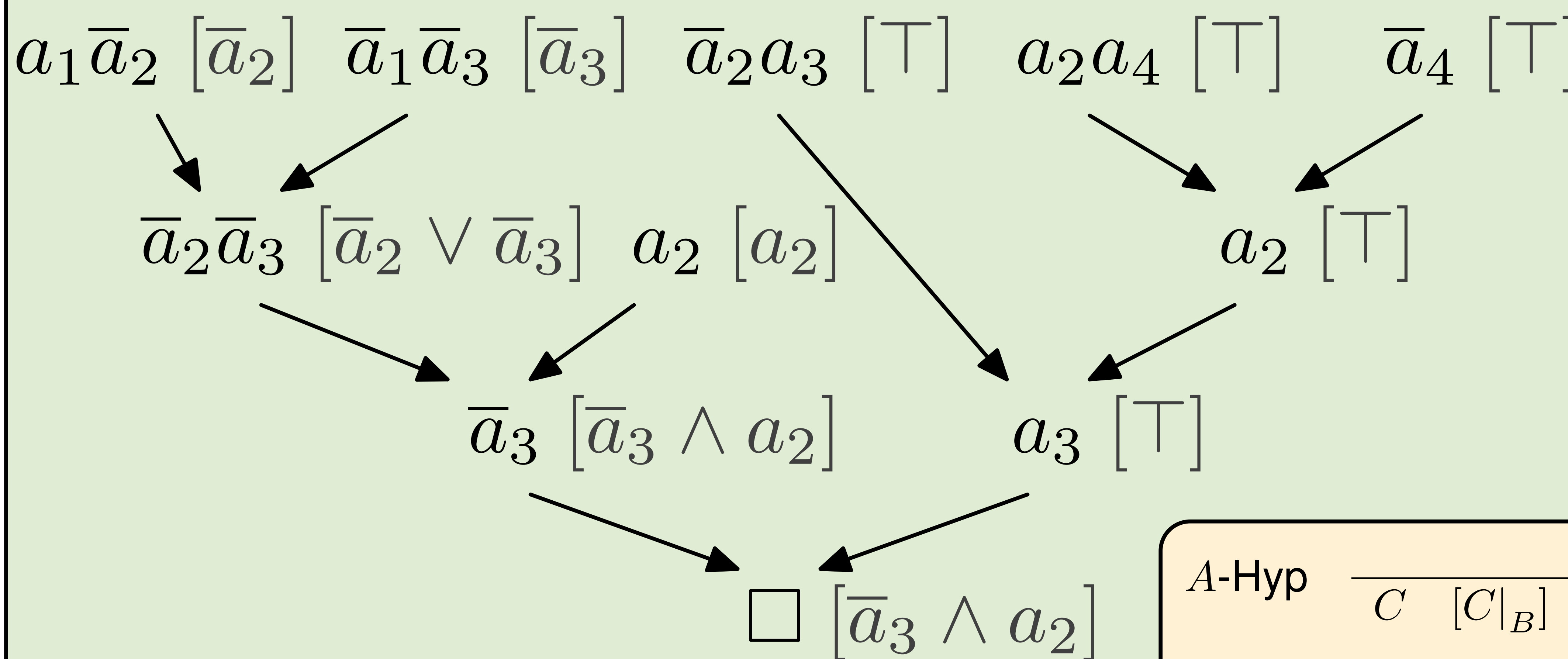
B-Res $\quad \dfrac{C \vee x\ [I_1] \qquad \overline{x} \vee D\ [I_2]}{C \vee D \quad [I_1 \wedge I_2]}$

$$A = (a_1 \vee \overline{a}_2) \wedge (\overline{a}_1 \vee \overline{a}_3) \wedge a_2$$

$$B = (\overline{a}_2 \vee a_3) \wedge (a_2 \vee a_4) \wedge \overline{a}_4$$

$$I =$$

Applying Interpolating Proof Rules



$$A = (a_1 \vee \overline{a}_2) \wedge (\overline{a}_1 \vee \overline{a}_3) \wedge a_2$$

$$B = (\overline{a}_2 \vee a_3) \wedge (a_2 \vee a_4) \wedge \overline{a}_4$$

$$I = \overline{a}_3 \wedge a_2$$

A-Hyp $$\frac{}{C \quad [C|_B]}$$

B-Hyp $$\frac{}{C \quad [\top]}$$

A-Res $$\frac{C \vee x\ [I_1] \qquad \overline{x} \vee D\ [I_2]}{C \vee D \quad [I_1 \vee I_2]}$$

B-Res $$\frac{C \vee x\ [I_1] \qquad \overline{x} \vee D\ [I_2]}{C \vee D \quad [I_1 \wedge I_2]}$$

McMillan's Interpolation System

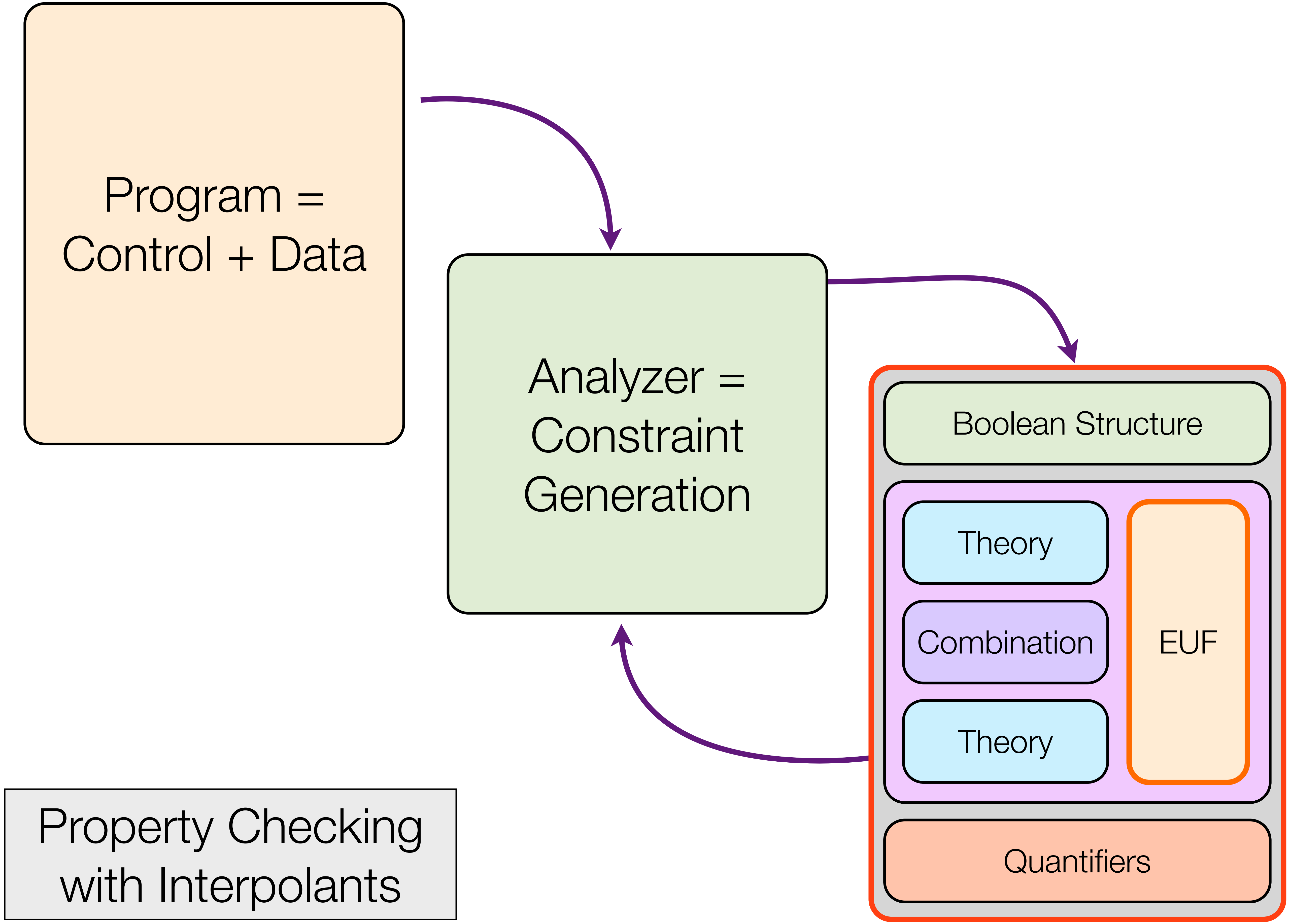
A-Hyp $\dfrac{}{C \quad [\{\ell \mid var(\ell) \in var(B)\}]}$ $(C \in A)$

B-Hyp $\dfrac{}{C \quad [\top]}$ $(C \in B)$

A-Res $\dfrac{C \vee x \; [I_1] \qquad \overline{x} \vee D \; [I_2]}{C \vee D \quad [I_1 \vee I_2]}$ $(x \in var(A) \setminus var(B))$

B-Res $\dfrac{C \vee x \; [I_1] \qquad \overline{x} \vee D \; [I_2]}{C \vee D \quad [I_1 \wedge I_2]}$ $(x \in var(B))$

Theorem. The partial interplant labelling the empty clause is an interplant for A and B.


Program =
Control + Data
Analyzer =
Constraint
Generation
Boolean Structure
Theory
Combination
Theory
EUF
Quantifiers
Property Checking
with Interpolants

Equality Proofs

$$\begin{tikzcd}
f(u,y) = z \arrow[dddr] & u = x \arrow[dr] & & v = y \arrow[dl] & f(x,v) \neq z \arrow[ddl] \\
& & \boxed{f(u,y) = f(x,v)} \arrow[dr] & & \\
& & & f(u,y) \neq z \arrow[dll] & \\
& \square & & &
\end{tikzcd}$$

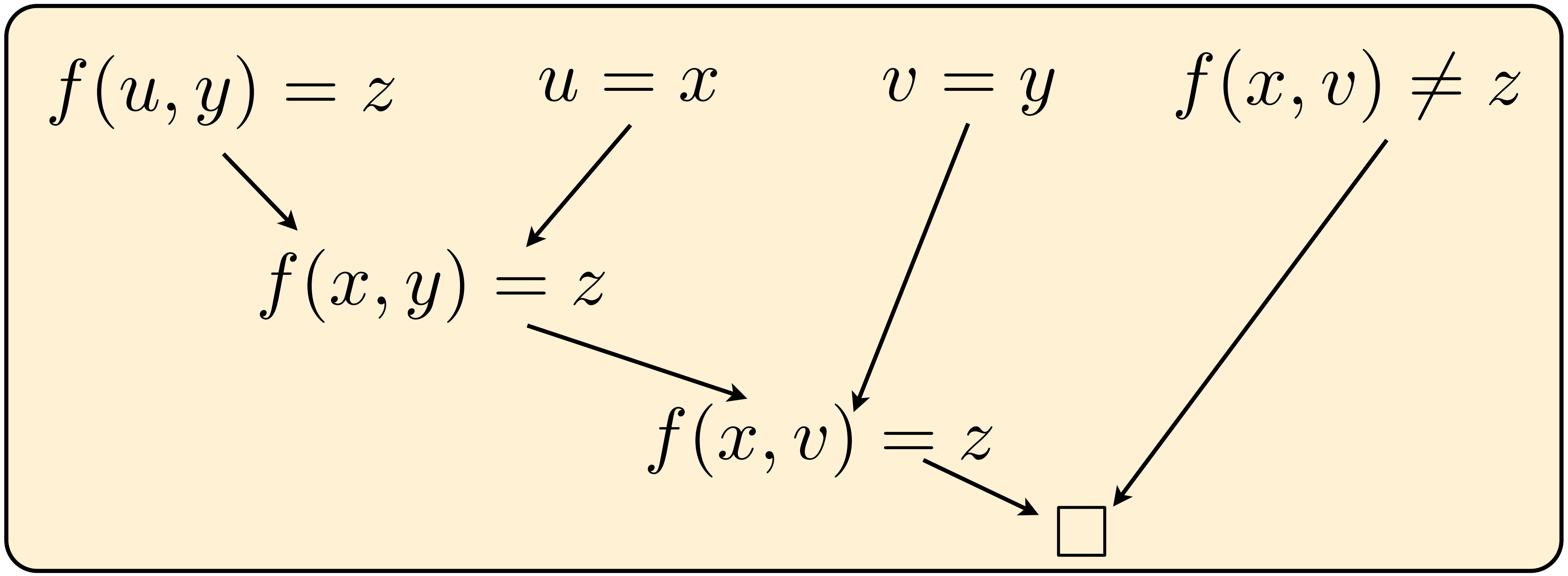
$$A = u = x \wedge f(u,y) = z$$

$$B = v = y \wedge f(x,v) \neq z$$

$$I = f(x,y) = z$$

- Deduced *literals* may not be in A or in B
- New *terms* may use non-shared symbols
- Interpolant may be over terms not in the proof

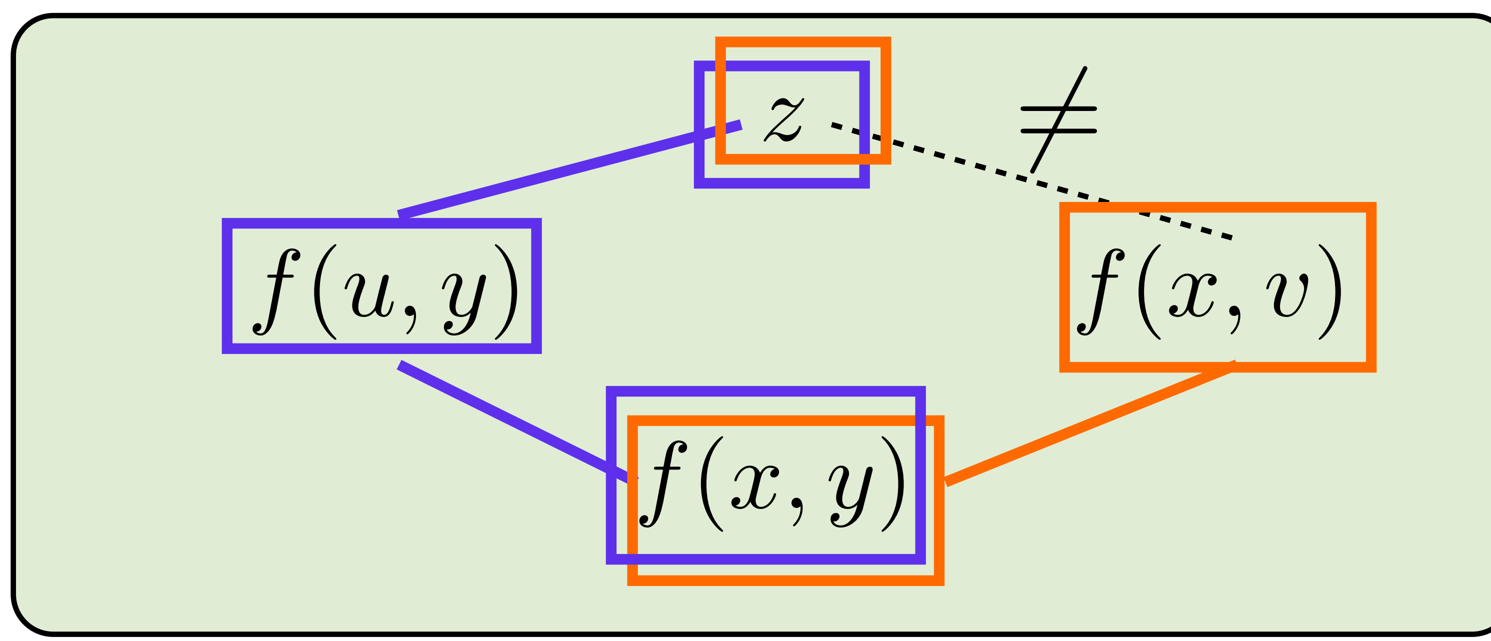
Coloured Congruence Graphs



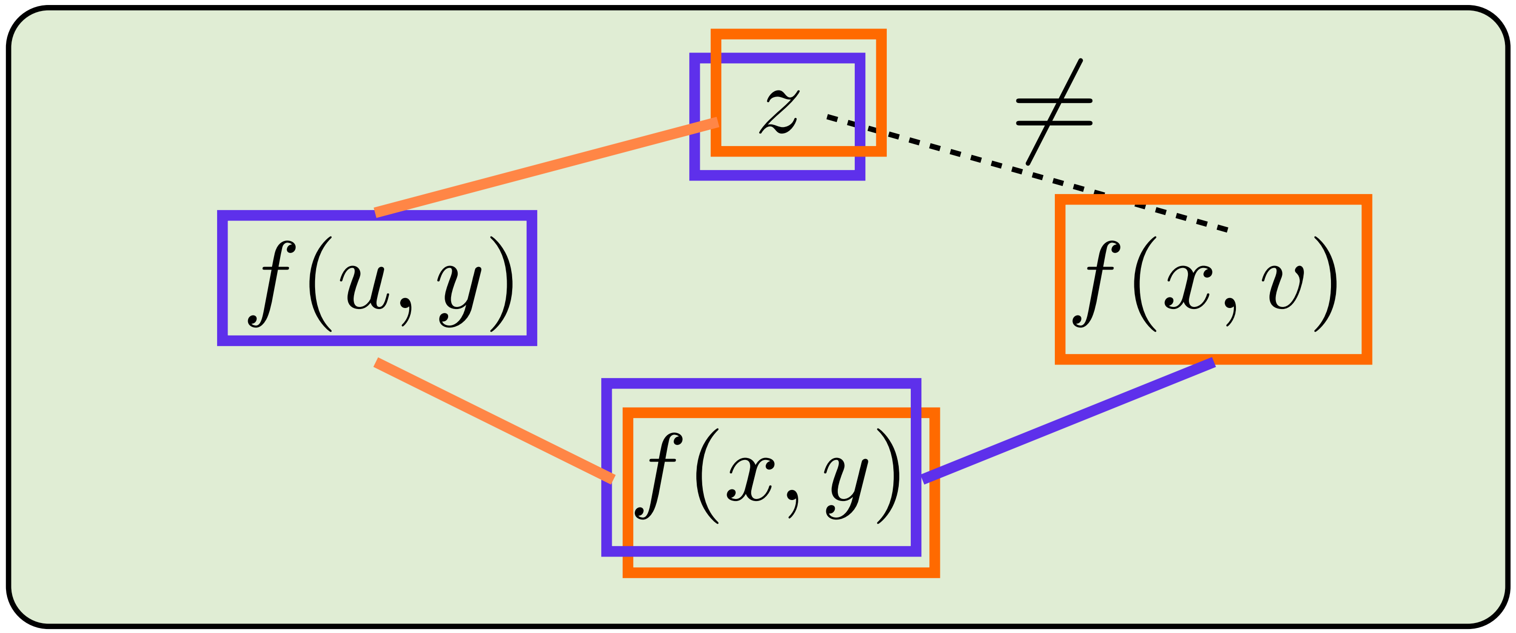
$$\boxed{A = u = x \wedge f(u,y) = z}$$

$$\boxed{B = v = y \wedge f(x,v) \neq z}$$

$$I = f(x,y) = z$$



Interpolation from Coloured Congruence Graphs



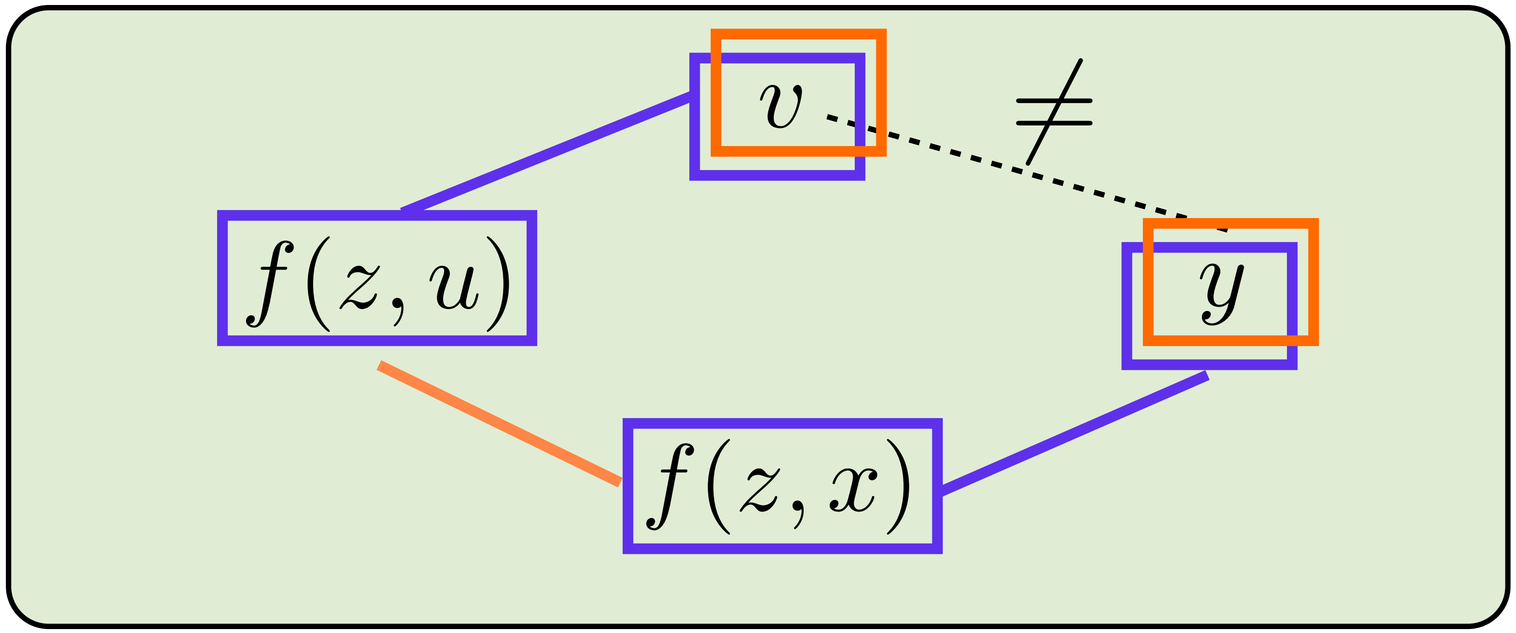
$A = u = x \wedge f(u,y) = z$

$B = v = y \wedge f(x,v) \neq z$

$I = f(x,y) = z$

- Modify graph to be colourable
- Take summaries A-paths by endpoints that are over the shared vocabulary
- Summarize B-paths as premises for A-summaries

Interpolation with B-Premises



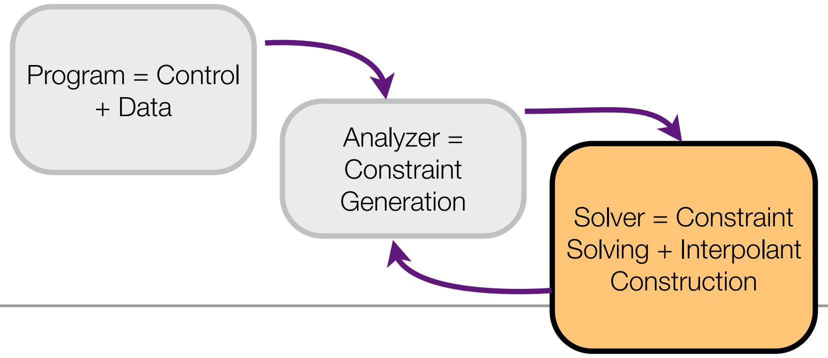
$$A = v = f(z,u) \wedge y = f(z,x)$$

$$B = u = x \wedge v \neq y$$

$$I = u = x \implies v = y$$

- Mixes A and B reasoning
- Endpoints of B-reasoning paths are antecedents of implications for A
- Implication introduced by combination of congruence and shared reasoning

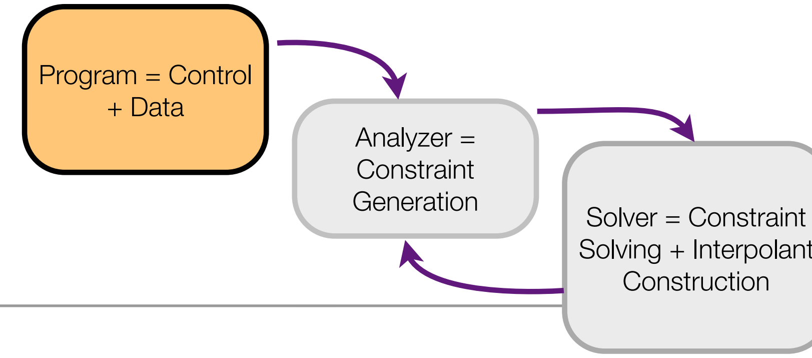
Interpolation and SMT



1995	Huang, Constructing Craig Interpolation Formulas. (OTTER)
2001	Amir, McIlraith, Partition-Based Logical Reasoning.
2003	McMillan, Interpolation and SAT-Based Model Checking.
200	Henziger, Jhala,Majumdar,McMillan, Abstractions from Proofs
2005	McMillan, An Interpolating Theorem Prover

2005 to present
Interpolation for theories: numeric, bit-vectors, strings, arrays, etc.
Interpolation for equality and theory combinations.
Quantified interpolants.
Sequence, tree and DAG interpolants.

Analysis with Interpolants

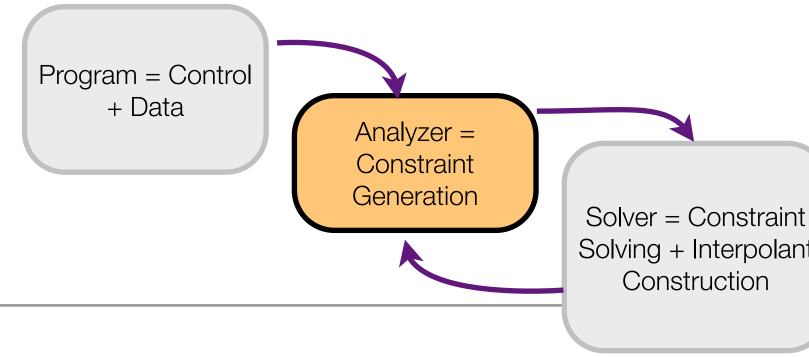


1957	1960	1970	1980	1990	2000	2010

2003	McMillan, Interpolation and SAT-Based Model Checking.
2004	Henzinger, Jhala, Majumdar, McMillan, Abstraction from Proofs.
2006	McMillan, Lazy Abstraction with Interpolants
2009	Vizel, Grumberg, Interpolation-Sequence Based Model Checking
2010	Heizmann, Hoenicke, Podelski, Nested Interpolants
2012	Albarghouthi, Gurfinkel, Chechik, Whale: An Inteprolation-Based Algorithm for Inter-Procedural Verification

2006 onwards	
Loops	Interpolant Sequence
Recursion	Tree Interpolant
Multiple Paths	DAG Interpolant
Frameworks	Horn Clauses

Analysis of Interpolants



2006	Jhala, McMillan, A Practical and Complete Approach to Predicate Refinement.	Theory Independent
2010	D. Kroening, Purandare, Weissenbacher, Interpolant Strength.	Propositional Logic
2012	Rollini, Sery, Sharygina. Leveraging Interpolant Strength in Model Checking.	
2012	Alberti, Brutomesso, Ghilardi, Ranise, Sharygina, Lazy Abstraction with Interpolants for Arrays.	
2013	Albarghouthi, McMillan, Beautiful Interpolants.	
2013	Ruemmer, Subotic, Exploring Interpolants.	

Further Reading: Propositional Interpolants

1995	Huang, Constructing Craig Interpolation Formulas. (OTTER)
1997	Jan Krajíček, Interpolation theorems, lower bounds for proof systems, and independence results for bounded arithmetic.
1997	Pudlák, Lower Bounds for Resolution and Cutting Plane Proofs and Monotone Computations
2003	McMillan, Interpolation and SAT-Based Model Checking.
2006	Yorsh, Musuvathi, A Combination Method for Generating Interpolants.
2009	Biere, Bounded Model Checking (in Handbook of Satisfiability).
2010	D. Kroening, Purandare, Weissenbacher. Interpolant Strength.

Further Reading: Equality Interpolants

1996	Fitting, First-Order Logic and Automated Theorem Proving
2005	McMillan, An Interpolating Theorem Prover
2006	Yorsh, Musuvathi, A Combination Method for Generating Interpolants.
2009	Fuchs, Goel, Grundy, Krstic, Tinelli, Ground Interpolation for the Theory of Equality.
2014	Bonacina, Johansson, Interpolation Systems for Ground Proofs in Automated Reasoning

Interpolation in Theories

2005	McMillan. Interpolating Theorem Prover	LA(Q)
2006	Kapur, Majumdar, Zarba, Interpolation for Data Structures	Datatype theories
2007	Rybalchenko, Sofronie-Stokkermans, Constraint Solving for Interpolation	LA(Q)
2008	Cimatti, Griggio, Sebastiani, Efficient Interpolant Generation in Satisfiability Modulo Theories	LA(Q), DL(Q), UTVPI
2008	Jain, Clarke, Grumberg, Efficient Craig Interpolation for Linear Diophantine (dis)Equations and Linear Modular Equations	LDE, LME
2009	Cimatti, Griggio, Sebastiani, Interpolant Generation for UTVPI	UTVPI
2011	Griggio, Effective Word-Level Interpolation for Software Verification	Bit-Vectors

Interpolation in Theory Combinations

2005	McMillan. Interpolating Theorem Prover	LA(Q) over EUF over Bool
2005	Yorsh and Musuvathi, A Combination Method for Generating Interpolants	Nelson-Oppen
2009	Cimatti, Griggio, Sebastiani, Efficient Generation of Craig Interpolants in Satisfiability Modulo Theories	Delayed Theory Combination
2009	Goel, Krstic, Tinelli, Ground Interpolation for Combined Theories	Proof transformation
2012	Kovacs, Voronkov, Playing in the Gray Area of Proofs	Proof Transformation