Algorithms for SAT

Automated Program Verification (APV)
Fall 2019

Prof. Arie Gurfinkel

UNIVERSITY OF WATERLOO

Syntax of Propositional Logic

An *atomic formula* has a form A_i , where i = 1, 2, 3 …

Formulas are defined inductively as follows:

- All atomic formulas are formulas
- For every formula F, $\neg F$ (called not F) is a formula
- For all formulas F and G, $F \wedge G$ (called and) and $F \vee G$ (called or) are formulas

Abbreviations

- use A, B, C, … instead of A_1, A_2, …
- use $F_1 \rightarrow F_2$ instead of $\neg F_1 \vee F_2$ (implication)
- use $F_1 \leftrightarrow F_2$ instead of $(F_1 \rightarrow F_2) \wedge (F_2 \rightarrow F_1)$ (iff)

Syntax of Propositional Logic (PL)

$$
\begin{aligned}
\text{truth\_symbol} &::= \top(\text{true}) \mid \bot(\text{false}) \\
\text{variable} &::= p, q, r, \ldots \\
\text{atom} &::= \text{truth\_symbol} \mid \text{variable} \\
\text{literal} &::= \text{atom} | \neg \text{atom} \\
\text{formula} &::= \text{literal} \mid \\
&\quad\; \neg\text{formula} \mid \\
&\quad\; \text{formula} \land \text{formula} \mid \\
&\quad\; \text{formula} \lor \text{formula} \mid \\
&\quad\; \text{formula} \rightarrow \text{formula} \mid \\
&\quad\; \text{formula} \leftrightarrow \text{formula}
\end{aligned}
$$

Normal Forms: CNF and DNF

A *literal* is either an atomic proposition v or its negation ~v

A *clause* is a disjunction of literals

- e.g., (v1 || ~v2 || v3)

A formula is in *Conjunctive Normal Form* (CNF) if it is a conjunction of disjunctions of literals (i.e., a conjunction of clauses):

- e.g., (v1 || ~v2) && (v3 || v2)

$$\bigwedge_{i=1}^{n} \left(\bigvee_{j=1}^{m_i} L_{i,j} \right)$$

A formula is in *Disjunctive Normal Form* (DNF) if it is a disjuction of conjunctions of literals

$$\bigvee_{i=1}^{n} \left(\bigwedge_{j=1}^{m_i} L_{i,j} \right)$$

Boolean Satisfiability (CNF-SAT)

Let V be a set of variables

A *literal* is either a variable v in V or its negation ~v

A *clause* is a disjunction of literals

- e.g., (v1 || ~v2 || v3)

A Boolean formula in *Conjunctive Normal Form* (CNF) is a conjunction of clauses

- e.g., (v1 || ~v2) && (v3 || v2)

An *assignment* *s* of Boolean values to variables *satisfies* a clause *c* if it evaluates at least one literal in *c* to true

An assignment *s* *satisfies* a formula *C* in CNF if it satisfies every clause in *C*

Boolean Satisfiability Problem (CNF-SAT):

- determine whether a given CNF C is satisfiable

CNF Examples

CNF 1

- ~b
- ~a || ~b || ~c
- a
- sat: s(a) = True; s(b) = False; s(c) = False

CNF 2

- ~b
- ~a || b || ~c
- a
- ~a || c
- unsat

Algorithms for SAT

SAT is NP-complete

- solution can be checked in polynomial time
- no polynomial algorithms for finding a solution are known

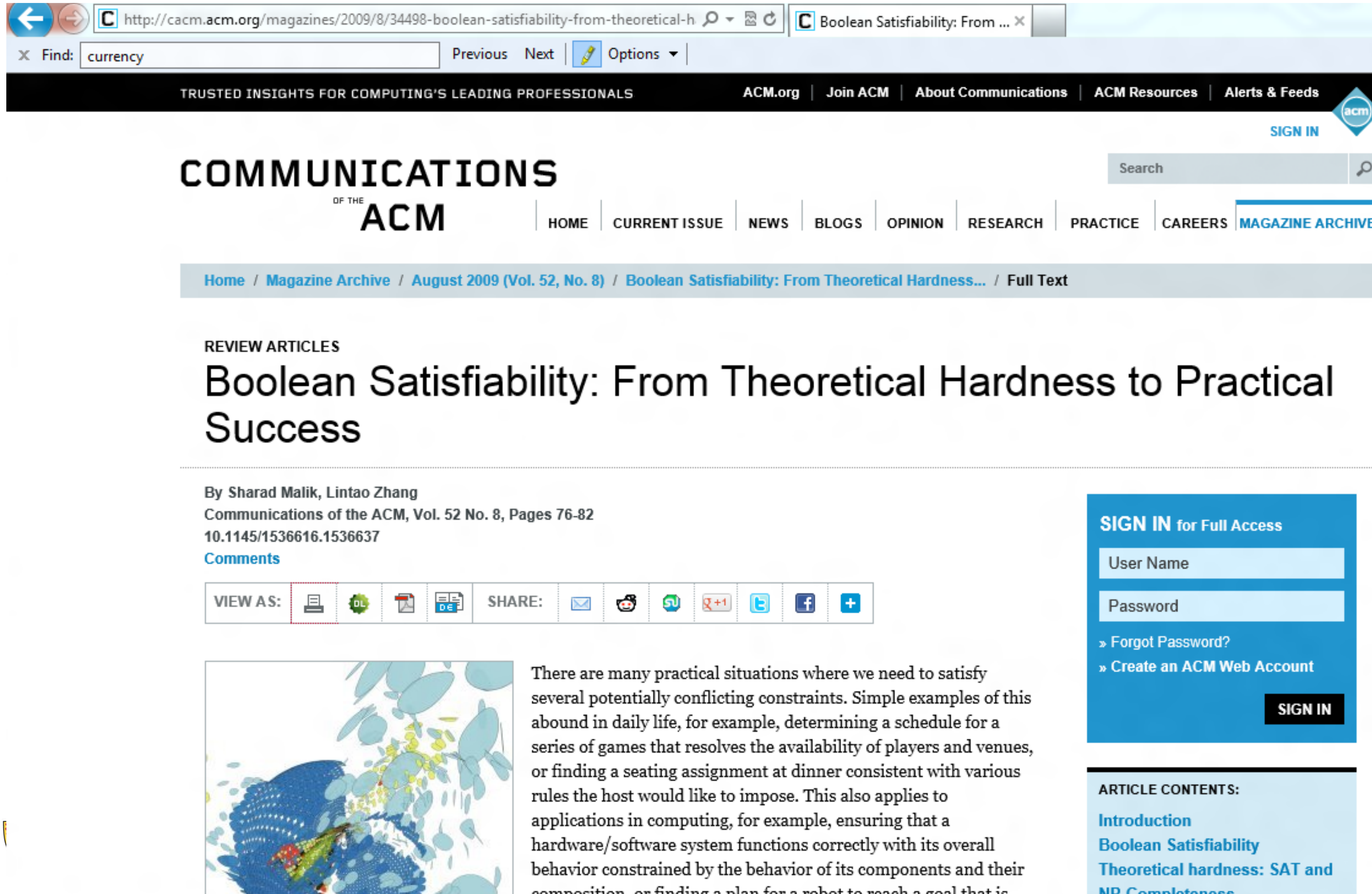
DPLL (Davis-Putnam-Logemman-Loveland, ‘60)

- smart enumeration of all possible SAT assignments
- worst-case EXPTIME
- alternate between deciding and propagating variable assignments

CDCL (GRASP ‘96, Chaff ‘01)

- conflict-driven clause learning
- extends DPLL with
  - smart data structures, backjumping, clause learning, heuristics, restarts…
- scales to millions of variables
- N. Een and N. Sörensson, “An Extensible SAT-solver”, in SAT 2013.

Background Reading: SAT



http://cacm.acm.org/magazines/2009/8/34498-boolean-satisfiability-from-theoretical-h

Boolean Satisfiability: From ...

Find: currency Previous Next Options

TRUSTED INSIGHTS FOR COMPUTING'S LEADING PROFESSIONALS

ACM.org | Join ACM | About Communications | ACM Resources | Alerts & Feeds

SIGN IN

COMMUNICATIONS OF THE ACM

Search

HOME | CURRENT ISSUE | NEWS | BLOGS | OPINION | RESEARCH | PRACTICE | CAREERS | MAGAZINE ARCHIVE

Home / Magazine Archive / August 2009 (Vol. 52, No. 8) / Boolean Satisfiability: From Theoretical Hardness... / Full Text

REVIEW ARTICLES

Boolean Satisfiability: From Theoretical Hardness to Practical Success

By Sharad Malik, Lintao Zhang

Communications of the ACM, Vol. 52 No. 8, Pages 76-82

10.1145/1536616.1536637

Comments

VIEW AS: SHARE:

There are many practical situations where we need to satisfy several potentially conflicting constraints. Simple examples of this abound in daily life, for example, determining a schedule for a series of games that resolves the availability of players and venues, or finding a seating assignment at dinner consistent with various rules the host would like to impose. This also applies to applications in computing, for example, ensuring that a hardware/software system functions correctly with its overall behavior constrained by the behavior of its components and their

SIGN IN for Full Access

User Name

Password

» Forgot Password?

» Create an ACM Web Account

SIGN IN

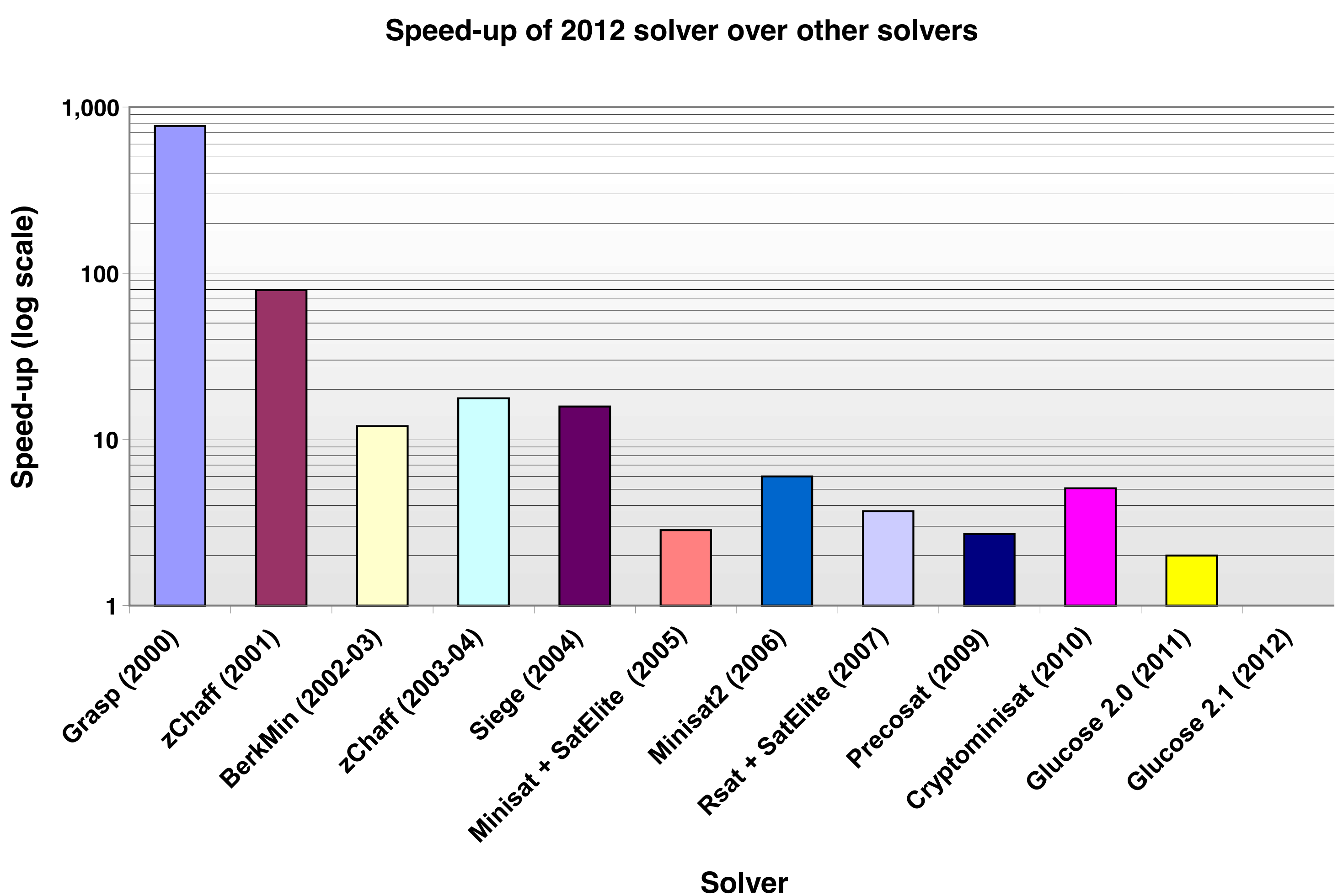
ARTICLE CONTENTS:

Introduction

Boolean Satisfiability

Theoretical hardness: SAT and

Some Experience with SAT Solving



from M. Vardi, https://www.cs.rice.edu/~vardi/papers/highlights15.pdf

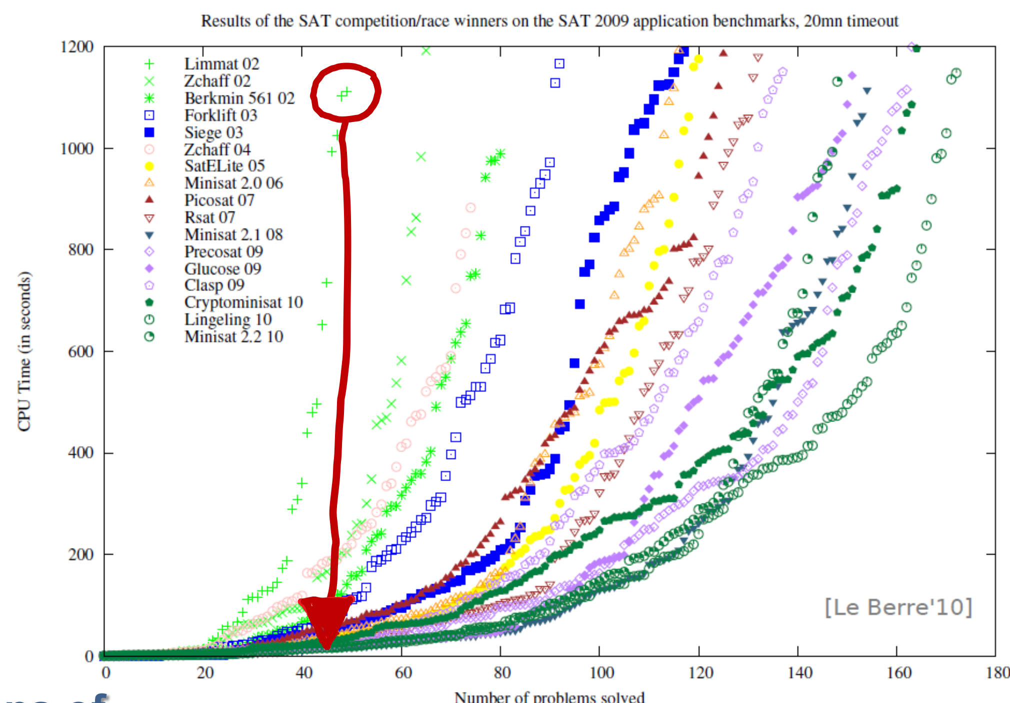
SAT - Milestones

Problems impossible 10 years ago are trivial today

year	Milestone
1960	Davis-Putnam procedure
1962	Davis-Logeman-Loveland
1984	Binary Decision Diagrams
1992	DIMACS SAT challenge
1994	SATO: clause indexing
1997	GRASP: conflict clause learning
1998	Search Restarts
2001	zChaff: 2-watch literal, VSIDS
2005	Preprocessing techniques
2007	Phase caching
2008	Cache optimized indexing
2009	In-processing, clause management
2010	Blocked clause elimination

Concept

2002 **2010**



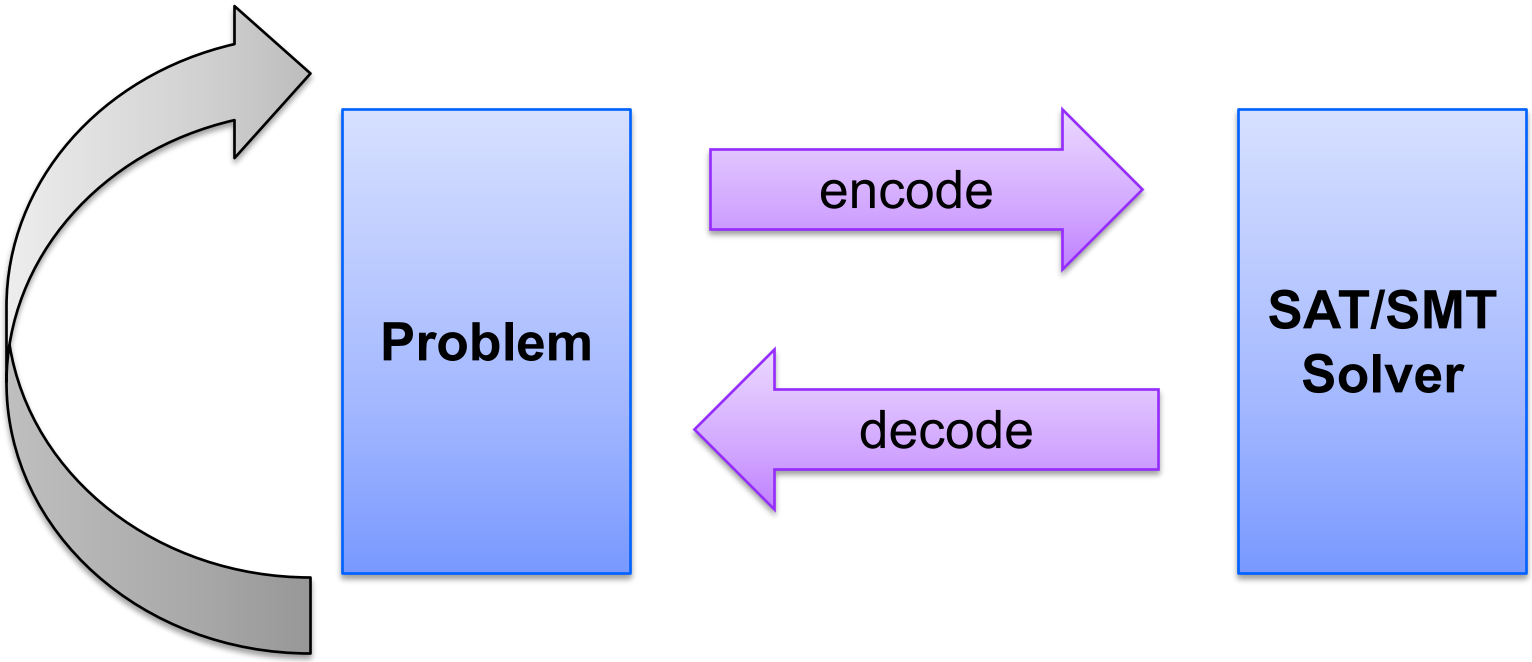
Millions of variables from HW designs

Courtesy Daniel le Berre

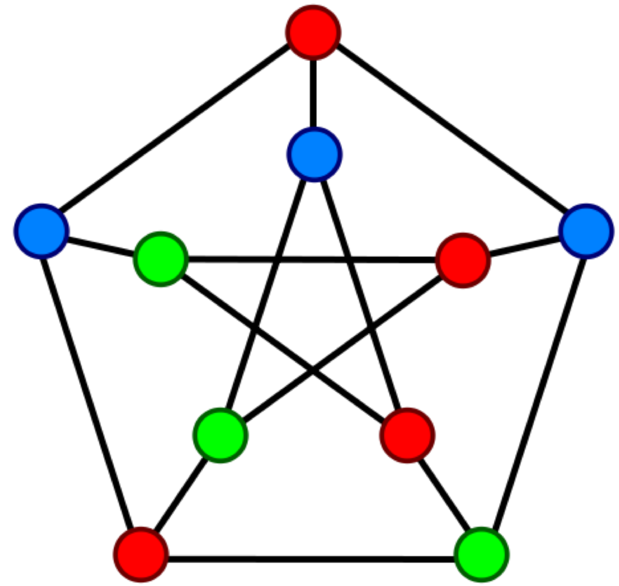
NP is the new P!

Solve any computational problem by effective reduction to SAT/SMT

- iterate as necessary



Graph k-Coloring



Given a graph $G = (V, E)$, and a natural number $k > 0$ is it possible to assign colors to vertices of G such that no two adjacent vertices have the same color.

Formally:

- does there exists a function f : V → [0..k) such that
- for every edge (u, v) in E, f(u) != f(v)

Graph coloring for k > 2 is NP-complete

Problem: Encode k-coloring of G into CNF

- construct CNF *C* such that C is SAT iff G is k-colorable

UNIVERSITY OF WATERLOO

https://en.wikipedia.org/wiki/Graph_coloring

k-coloring as CNF

Let a Boolean variable $f_{v,i}$ denote that vertex *v* has color *i*

- if $f_{v,i}$ is true if and only if $f(v) = i$

Every vertex has at least one color

$$\bigvee_{0 \le i < k} f_{v,i} \qquad\qquad (v \in V)$$

No vertex is assigned two colors

$$\bigwedge_{0 \le i < j < k} (\neg f_{v,i} \vee \neg f_{v,j}) \qquad\qquad (v \in V)$$

No two adjacent vertices have the same color

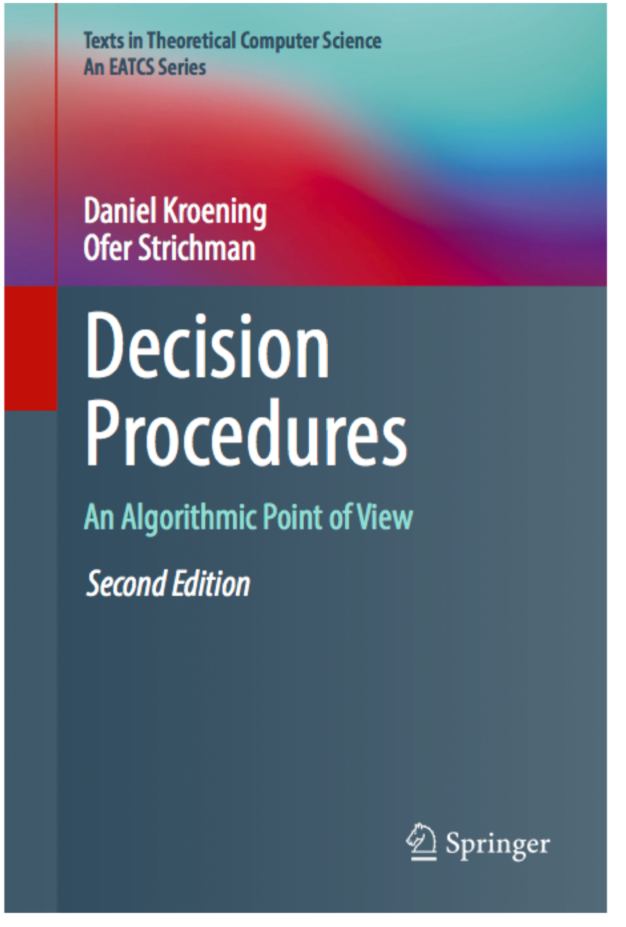
$$\bigwedge_{0 \le i < k} (\neg f_{v,i} \vee \neg f_{u,i}) \qquad\qquad ((v,u) \in E)$$

Davis Putnam Logemann Loveland

DPLL PROCEDURE

References

Chapter 2: Decision Procedures for Propositional Logic



https://link.springer.com/book/10.1007%2F978-3-540-74105-3

Decision Procedure for Satisfiability

Algorithm that in some finite amount of computation decides if a given propositional logic (PL) formula F is satisfiable

- NP-complete problem

Modern decision procedures for PL formulae are called SAT solvers

Naïve approach

- Enumerate models (i.e., truth tables)
- Enumerate resolution proofs

Modern SAT solvers

- DPLL algorithm
  - Davis-Putnam-Logemann-Loveland
- Combines model- and proof-based search
- Operates on Conjunctive Normal Form (CNF)

Propositional Resolution

Pivot

$$\frac{C \vee p \qquad\qquad D \vee \neg p}{C \vee D}$$

Resolvent

Res({C, p}, {D, !p}) = {C, D}

Given two clauses {C, p} and {D, !p} that contain a literal p of different polarity, create a new clause by taking the union of literals in C and D

Resolution Lemma

Lemma:

Let F be a CNF formula. Let R be a resolvent of two clauses X and Y in F. Then, $F \cup \{R\}$ is equivalent to F

Resolution Theorem

Let F be a set of clauses

$Res(F) = F \cup \{R \mid R \text{ is a resolvent of two clauses in } F\}$

$$Res^0(F) = F$$

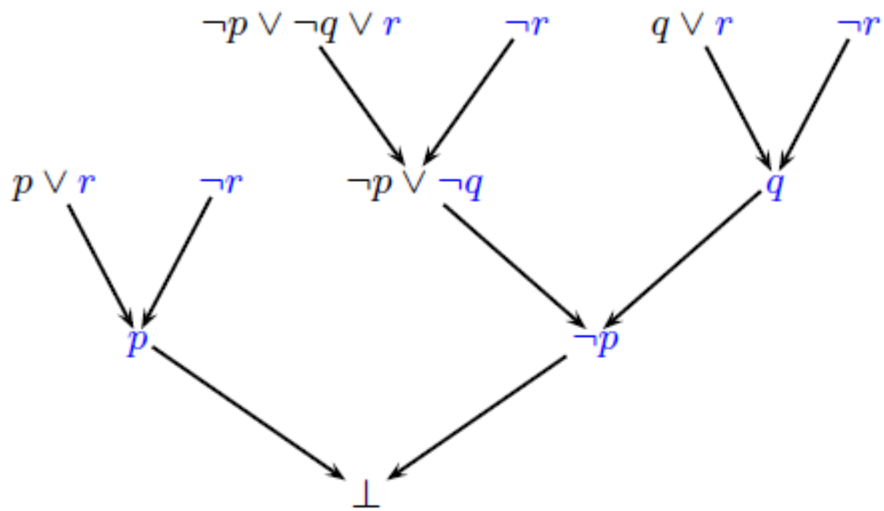
$$Res^{n+1}(F) = Res(Res^n(F)), \text{ for } n \geq 0$$

$$Res^*(F) = \bigcup_{n \geq 0} Res^n(F)$$

Theorem: A CNF F is UNAT iff Res*(F) contains an empty clause

Example of a resolution proof

A refutation of $\neg p \vee \neg q \vee r$, $p \vee r$, $q \vee r$, $\neg r$:



Resolution Proof Example

Show by resolution that the following CNF is UNSAT

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

$$\dfrac{\dfrac{\dfrac{\neg a \vee b \vee \neg c \qquad a}{b \vee \neg c} \qquad b}{\neg c} \qquad \dfrac{a \qquad \neg a \vee c}{c}}{\bot}$$

UNIVERSITY OF WATERLOO

Proof of the Resolution Theorem

(Soundness) By Resolution Lemma, F is equivalent to $Res^i(F)$ for any i. Let n be such that $Res^{n+1}(F)$ contains an empty clause, but $Res^n(F)$ does not. Then $Res^n(F)$ must contain to unit clauses L and ¬L. Hence, it is UNSAT.

(Completeness) By induction on the number of different atomic propositions in F.

Base case is trivial: F contains an empty clause.

IH: Assume F has atomic propositions A1, … A_{n+1}

Let F_0 be the result of replacing A_{n+1} by 0

Let F_1 be the result of replacing A_{n+1} by 1

Apply IH to F_0 and F_1 . Restore replaced literals. Combine the two resolutions.

Proof System

$$P_1, \ldots, P_n \vdash C$$

An inference rule is a tuple (P_1, …, P_n, C)

- where, P_1, …, P_n, C are formulas
- P_i are called premises and C is called a conclusion
- intuitively, the rules says that the conclusion is true if the premises are

A proof system P is a collection of inference rules

A proof in a proof system P is a tree (or a DAG) such that

- nodes are labeled by formulas
- for each node *n*, (parents(n), n) is an inference rule in P

UNIVERSITY OF WATERLOO

Propositional Resolution

$$\frac{C \lor p \qquad\qquad D \lor \neg p}{C \lor D}$$

Propositional resolution is a sound inference rule

Proposition resolution system consists of a single propositional resolution rule

DP Procedure: SAT solving by resolution

Assume that input formula F is in CNF

1. Pick two clauses C_1 and C_2 in F that can be resolved
2. If the resolvent C is an empty clause, return UNSAT
3. Otherwise, add C to F and go to step 1
4. If no new clauses can be resolved, return SAT

Termination: finitely many derived clauses

DPLL: David Putnam Logemann Loveland

Combines pure resolution-based search with case splitting on decisions

Proof search is restricted to unit resolution

- can be done very efficiently (polynomial time)

Case split restores completeness

DPLL can be described by the following two rules

- F is the input formula in CNF

$$\frac{F}{F, p \quad \mid \quad F, \neg p}\ \textbf{split} \qquad p \ and \ \neg p \ are \ not \ in \ F$$

$$\frac{F,\ C \vee \ell, \neg \ell}{F, C, \neg \ell}\text{unit}$$

Davis, Martin; Logemann, George; Loveland, Donald (1962). "A Machine Program for Theorem Proving". C. ACM. 5 (7): 394–397. doi:10.1145/368273.368557

UNIVERSITY OF WATERLOO

The original DPLL procedure

Incrementally builds a satisfying truth assignment M for the input CNF formula F

M is grown by

- deducing the truth value of a literal from M and F, or
- guessing a truth value

If a wrong guess for a literal leads to an inconsistency, the procedure backtracks and tries the opposite value

DPLL: Illustration

M | F

Partial model

Set of clauses

DPLL: Decide

Guessing (Decide)

$$p \mid p \vee q, \neg q \vee r$$

$$\Downarrow$$

$$p, \neg q \mid p \vee q, \neg q \vee r$$

DPLL: Boolean Constraint Propagation

Deducing (Unit Propagation or BCP)

$$p \mid p \vee q, \neg p \vee s$$

$$\Downarrow$$

$$p, s \mid p \vee q, \neg p \vee s$$

DPLL: Backtracking

Backtracking

$$p, \neg s,\ q \ \mid\ p \vee q,\ s \vee q,\ \neg p \vee \neg q$$

$$\Downarrow$$

$$p,\ s \mid p \vee q,\ s \vee q,\ \neg p \vee \neg q$$

Pure Literals

A literal is pure if only occurs positively or negatively.

Example :

$\varphi = (\boxed{\neg x_1} \vee x_2) \wedge (\boxed{x_3} \vee \neg x_2) \wedge (x_4 \vee \neg x_5) \wedge (x_5 \vee \neg x_4)$

$\neg x_1$ and x_3 are pure literals

Pure literal rule :

Clauses containing pure literals can be removed from the formula (i.e. just satisfy those pure literals)

$$\varphi_{\neg x_1, x_3} = (x_4 \vee \neg x_5) \wedge (x_5 \vee \neg x_4)$$

Preserve satisfiability, not logical equivalency !

DPLL Procedure

- Standard backtrack search
- DPLL(F) :
  - Apply unit propagation
  - If conflict identified, return UNSAT
  - Apply the pure literal rule
  - If F is satisfied (empty), return SAT
  - Select decision variable x
    - If DPLL($F \wedge x$)=SAT return SAT
    - return DPLL($F \wedge \neg x$)

The Original DPLL Procedure – Example

assign	$1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg 1 \vee \neg 2,\ \neg 1 \vee \neg 3 \vee \neg 4,\ \boxed{1}$
Deduce 1	
1	$1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \boxed{\neg 1 \vee \neg 2},\ \neg 1 \vee \neg 3 \vee \neg 4,\ 1$
Deduce ¬2	
1, 2	$1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg 1 \vee \neg 2,\ \neg 1 \vee \neg 3 \vee \neg 4,\ 1$
Guess 3	
1, 2, 3	$1 \vee 2,\ \boxed{2 \vee \neg 3 \vee 4},\ \neg 1 \vee \neg 2,\ \neg 1 \vee \neg 3 \vee \neg 4,\ 1$
Deduce 4	
1, 2, 3, 4	$1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg 1 \vee \neg 2,\ \neg 1 \vee \neg 3 \vee \neg 4,\ 1$
Conflict	

UNIVERSITY OF WATERLOO

The Original DPLL Procedure – Example

assign	$1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg 1 \vee \neg 2,\ \neg 1 \vee \neg 3 \vee \neg 4,\ \boxed{1}$
Deduce 1	
1	$1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \boxed{\neg 1 \vee \neg 2,}\ \neg 1 \vee \neg 3 \vee \neg 4,\ 1$
Deduce ¬2	
1, 2	$1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg 1 \vee \neg 2,\ \neg 1 \vee \neg 3 \vee \neg 4,\ 1$
Guess 3	
1, 2, 3	$1 \vee 2,\ \boxed{2 \vee \neg 3 \vee 4,}\ \neg 1 \vee \neg 2,\ \neg 1 \vee \neg 3 \vee \neg 4,\ 1$
Deduce 4	
1, 2, 3, 4	$1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg 1 \vee \neg 2,\ \neg 1 \vee \neg 3 \vee \neg 4,\ 1$
Undo 3	

The Original DPLL Procedure – Example

assign	$1 \lor 2,\ 2 \lor \neg 3 \lor 4,\ \neg 1 \lor \neg 2,\ \neg 1 \lor \neg 3 \lor \neg 4,\ 1$
Deduce 1	
1	$1 \lor 2,\ 2 \lor \neg 3 \lor 4,\ \neg 1 \lor \neg 2,\ \neg 1 \lor \neg 3 \lor \neg 4,\ 1$
Deduce $\neg 2$	
1, 2	$1 \lor 2,\ 2 \lor \neg 3 \lor 4,\ \neg 1 \lor \neg 2,\ \neg 1 \lor \neg 3 \lor \neg 4,\ 1$
Guess $\neg 3$	
1, 2, 3	$1 \lor 2,\ 2 \lor \neg 3 \lor 4,\ \neg 1 \lor \neg 2,\ \neg 1 \lor \neg 3 \lor \neg 4,\ 1$
Model Found	

An Abstract Framework for DPLL

The DPLL procedure can be described declaratively by simple sequent-style calculi

Such calculi, however, cannot model meta-logical features such as backtracking, learning, and restarts

We model DPLL and its enhancements as transition systems instead

A transition system is a binary relation over states, induced by a set of conditional transition rules

An Abstract Framework for DPLL

State

- **fail** or M ‖ F
- where
  - F is a CNF formula, a set of clauses, and
  - M is a sequence of annotated literals denoting a partial truth assignment

Initial State

- ∅ ‖ F, where F is to be checked for satisfiability

Expected final states:

- **fail** if F is unsatisfiable
- M ‖ G
  where
  - M is a model of G
  - G is logically equivalent to F

Transition Rules for DPLL

Extending the assignment:

UnitProp $M \parallel F, C \vee l \rightarrow M\, l \parallel F, C \vee l$ $\begin{cases} M \models \neg C \\ l \text{ is undefined in } M \end{cases}$

Decide $M \parallel F, C \rightarrow M\, l^d \parallel F, C$ $\begin{cases} l \text{ or } \neg l \text{ occur in } C \\ l \text{ is undefined in } M \end{cases}$

Notation: l^d is a decision literal

Transition Rules for DPLL

Repairing the assignment:

Fail $\quad M \parallel F, C \rightarrow \text{fail} \quad \begin{cases} M \models \neg C \\ M \text{ does not contain decision literals} \end{cases}$

Backtrack $\quad M\ l^d\ N \parallel F, C \rightarrow M\ \neg l \parallel F, C \quad \begin{cases} M\ l^d\ N \models \neg C \\ l \text{ is the last decision literal} \end{cases}$

Transition Rules DPLL – Example

$\emptyset \parallel 1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg 1 \vee \neg 2,\ \neg 1 \vee \neg 3 \vee \neg 4,\ 1$

UnitProp 1

$1 \parallel 1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg 1 \vee \neg 2,\ \neg 1 \vee \neg 3 \vee \neg 4,\ 1$

UnitProp ¬2

$1, 2 \parallel 1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg 1 \vee \neg 2,\ \neg 1 \vee \neg 3 \vee \neg 4,\ 1$

Decide 3

$1, 2, 3^d \parallel 1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg 1 \vee \neg 2,\ \neg 1 \vee \neg 3 \vee \neg 4,\ 1$

UnitProp 4

$1, 2, 3^d, 4 \parallel 1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg 1 \vee \neg 2,\ \neg 1 \vee \neg 3 \vee \neg 4,\ 1$

Backtrack 3

Transition Rules DPLL – Example

$\emptyset \parallel 1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg 1 \vee \neg 2,\ \neg 1 \vee \neg 3 \vee \neg 4,\ 1$

UnitProp 1

$1 \parallel 1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg 1 \vee \neg 2,\ \neg 1 \vee \neg 3 \vee \neg 4,\ 1$

UnitProp $\neg 2$

$1, 2 \parallel 1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg 1 \vee \neg 2,\ \neg 1 \vee \neg 3 \vee \neg 4,\ 1$

Decide 3

$1, 2, 3^d \parallel 1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg 1 \vee \neg 2,\ \neg 1 \vee \neg 3 \vee \neg 4,\ 1$

UnitProp 4

$1, 2, 3 \parallel 1 \vee 2,\ 2 \vee \neg 3 \vee 4,\ \neg 1 \vee \neg 2,\ \neg 1 \vee \neg 3 \vee \neg 4,\ 1$

Backtrack 3

Transition Rules for DPLL (on one slide)

UnitProp $M \parallel F, C \vee l \to M\, l \parallel F, C \vee l$ $\begin{cases} M \models \neg C \\ l \text{ is undefined in } M \end{cases}$

Decide $M \parallel F, C \to M\, l^d \parallel F, C$ $\begin{cases} l \text{ or } \neg l \text{ occur in } C \\ l \text{ is undefined in } M \end{cases}$

Fail $M \parallel F, C \to \text{fail}$ $\begin{cases} M \models \neg C \\ M \text{ does not contain decision literals} \end{cases}$

Backtrack $M\, l^d\, N \parallel F, C \to M\, \neg l \parallel F, C$ $\begin{cases} M\, l^d\, N \models \neg C \\ l \text{ is the last decision literal} \end{cases}$

UNIVERSITY OF WATERLOO

The DPLL System – Correctness

Some terminology

- Irreducible state: state to which no transition rule applies.
- Execution: sequence of transitions allowed by the rules and starting with states of the form $\emptyset \parallel F$.
- Exhausted execution: execution ending in an irreducible state

Proposition (Strong Termination) Every execution in DPLL is finite

Proposition (Soundness) For every exhausted execution starting with $\emptyset \parallel F$ and ending in $M \parallel F$, $M \models F$

Proposition (Completeness) If F is unsatisfiable, every exhausted execution starting with $\emptyset \parallel F$ ends with fail

Maintained in more general rules + theories

Modern DPLL: CDCL

Conflict **D**riven **C**lause **L**earning

- two watched literals – efficient index to find clauses that can be used in unit resolution
- periodically restart backtrack search
- activity-based decision heuristic to choose decision variable
- **conflict resolution via clausal learning**

We will briefly look at clausal learning

More details on CDCL are available in

- Chapter 2 of Decision Procedures book
- ECE750 with Vijay Ganesh

Conflict Directed Clause Learning

Lemma learning

$\neg t, p, q, s \mid t \vee \neg p \vee q, \neg q \vee s, \neg p \vee \neg s$

$\neg t, p, q, s \mid t \vee \neg p \vee q, \neg q \vee s, \neg p \vee \neg s \mid \neg p \vee \neg s$

$\neg t, p, q, s \mid t \vee \neg p \vee q, \neg q \vee s, \neg p \vee \neg s \mid \neg p \vee \neg q$

$\neg t, p, q, s \mid t \vee \neg p \vee q, \neg q \vee s, \neg p \vee \neg s \mid \neg p \vee t$

UNIVERSITY OF WATERLOO

Learned Clause by Resolution

A new clause is learned by resolving the conflict clause with clauses deduced from the last decision

$$\neg t, p, q, s \mid t \lor \neg p \lor q, \neg q \lor s, \neg p \lor \neg s$$

$$\dfrac{\dfrac{\neg p \lor \neg s \qquad \neg q \lor s}{\neg p \lor \neg q} \qquad t \lor \neg p \lor q}{t \lor \neg p}$$

Trivial Resolution: at every resolution step, at least one clause is an input clause

Modern CDCL: Abstract Rules

Initialize	$\epsilon \mid F$	F *is a set of clauses*
Decide	$M \mid F \implies M, \ell \mid F$	ℓ *is unassigned*
Propagate	$M \mid F, C \vee \ell \implies M, \ell^{C \vee \ell} \mid F, C \vee \ell$	C *is false under* M
Sat	$M \mid F \implies M$	F *true under* M
Conflict	$M \mid F, C \implies M \mid F, C \mid C$	C *is false under* M
Learn	$M \mid F \mid C \implies M \mid F, C \mid C$	
Unsat	$M \mid F \mid \emptyset \implies Unsat$	
Backjump	$MM' \mid F \mid C \vee \ell \implies M\ell^{C \vee \ell} \mid F$	$\bar{C} \subseteq M, \neg \ell \in M'$
Resolve	$M \mid F \mid C' \vee \neg \ell \implies M \mid F \mid C' \vee C$	$\ell^{C \vee \ell} \in M$
Forget	$M \mid F, C \implies M \mid F$	C is a learned clause
Restart	$M \mid F \implies \epsilon \mid F$	

Model

Proof

Conflict Resolution

[Nieuwenhuis, Oliveras, Tinelli J.ACM 06] customized

UNIVERSITY OF WATERLOO

Conjuctive Normal Form

$$\begin{aligned}
\varphi \leftrightarrow \psi &\Rightarrow_{\mathrm{CNF}} \varphi \rightarrow \psi \wedge \psi \rightarrow \varphi \\
\varphi \rightarrow \psi &\Rightarrow_{\mathrm{CNF}} \neg\varphi \vee \psi \\
\neg(\varphi \vee \psi) &\Rightarrow_{\mathrm{CNF}} \neg\varphi \wedge \neg\psi \\
\neg(\varphi \wedge \psi) &\Rightarrow_{\mathrm{CNF}} \neg\varphi \vee \neg\psi \\
\neg\neg\varphi &\Rightarrow_{\mathrm{CNF}} \varphi \\
(\varphi \wedge \psi) \vee \xi &\Rightarrow_{\mathrm{CNF}} (\varphi \vee \xi) \wedge (\psi \vee \xi)
\end{aligned}$$

Every propositional formula can be put in CNF

PROBLEM: (potential) exponential blowup of the resulting formula

Tseitin Transformation – Main Idea

Introduce a fresh variable e_i for every subformula G_i of F

- intuitively, e_i represents the truth value of G_i

Assert that every e_i and G_i pair are equivalent

- $e_i \leftrightarrow G_i$
- and express the assertion as CNF

Conjoin all such assertions in the end

Formula to CNF Conversion

```
def cnf (ϕ):
  p, F = cnf_rec (ϕ)
  return p ∧ F

def cnf_rec (ϕ):
  if is_atomic (ϕ): return (ϕ, True)
  elif ϕ == ψ ∧ ξ:
    q, F₁ = cnf_rec (ψ)
    r, F₂ = cnf_rec (ξ)

    p = mk_fresh_var ()
    # C is CNF for p↔(q∧r)
    C = (¬p∨q)∧(¬p∨r)∧(p∨¬q∨¬r)
    return (p, F₁∧F₂∧C)
  elif ϕ == ψ∨ξ:
    …
```

`mk_fresh_var()` returns a fresh variable not used anywhere before

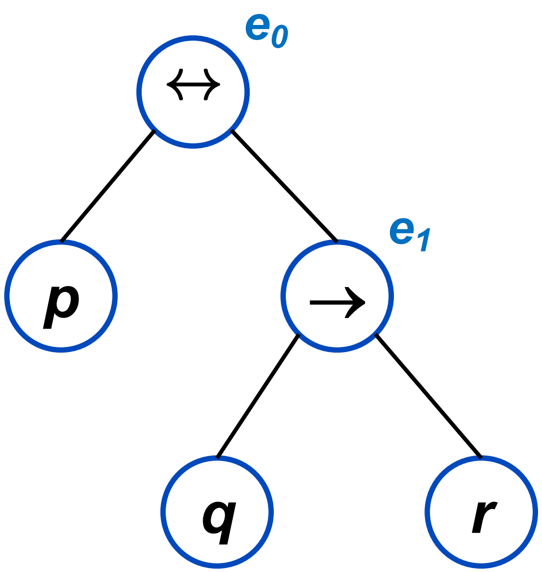
Exercise: Complete cases for

$\phi == \psi \vee \xi$, $\phi == \neg\psi$, $\phi == \psi \leftrightarrow \xi$

UNIVERSITY OF WATERLOO

Tseitin Transformation: Example

$G : p \leftrightarrow (q \rightarrow r)$

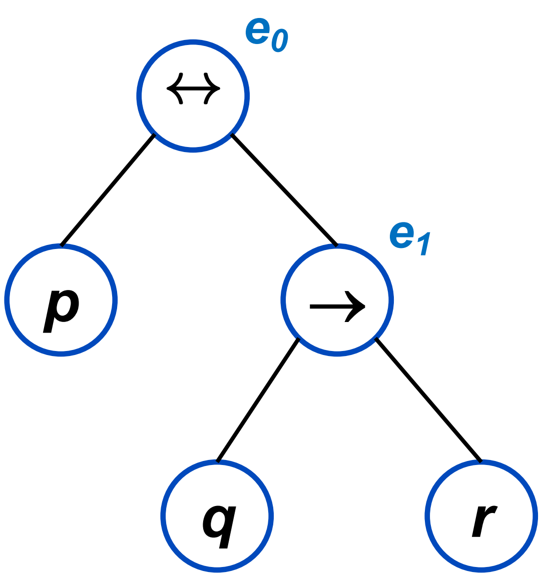


$G : e_0 \wedge (e_0 \leftrightarrow (p \leftrightarrow e_1)) \wedge (e_1 \leftrightarrow (q \rightarrow r))$

$$\begin{aligned}
& e_1 \leftrightarrow (q \rightarrow r) \\
= \quad & (e_1 \rightarrow (q \rightarrow r)) \wedge ((q \rightarrow r) \rightarrow e_1) \\
= \quad & (\neg e_1 \vee \neg q \vee r) \wedge ((\neg q \vee r) \rightarrow e_1) \\
= \quad & (\neg e_1 \vee \neg q \vee r) \wedge (\neg q \rightarrow e_1) \wedge (r \rightarrow e_1) \\
= \quad & (\neg e_1 \vee \neg q \vee r) \wedge (q \vee e_1) \wedge (\neg r \vee e_1)
\end{aligned}$$

Tseitin Transformation: Example

$G : p \leftrightarrow (q \rightarrow r)$

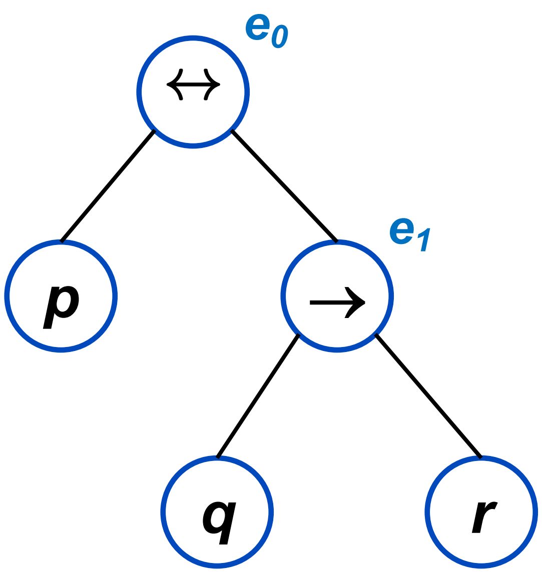


$G : e_0 \wedge (e_0 \leftrightarrow (p \leftrightarrow e_1)) \wedge (e_1 \leftrightarrow (q \rightarrow r))$

$$\begin{aligned}
& e_0 \leftrightarrow (p \leftrightarrow e_1) \\
= \quad & (e_0 \rightarrow (p \leftrightarrow e_1)) \wedge ((p \leftrightarrow e_1)) \rightarrow e_0) \\
= \quad & (e_0 \rightarrow (p \rightarrow e_1)) \wedge (e_0 \rightarrow (e_1 \rightarrow p)) \wedge \\
& (((p \wedge e_1) \vee (\neg p \wedge \neg e_1)) \rightarrow e_0) \\
= \quad & (\neg e_0 \vee \neg p \vee e_1) \wedge (\neg e_0 \vee \neg e_1 \vee p) \wedge \\
& (\neg p \vee \neg e_1 \vee e_0) \wedge (p \vee e_1 \vee e_0)
\end{aligned}$$

Tseitin Transformation: Example

$G : p \leftrightarrow (q \rightarrow r)$



$G : e_0 \wedge (e_0 \leftrightarrow (p \leftrightarrow e_1)) \wedge (e_1 \leftrightarrow (q \rightarrow r))$

$G : e_0 \wedge (\neg e_0 \vee \neg p \vee e_1) \wedge (\neg e_0 \vee p \vee \neg e_1) \wedge (e_0 \vee p \vee e_1) \wedge (e_0 \vee \neg p \vee \neg e_1) \wedge$
$(\neg e_1 \vee \neg q \vee r) \wedge (e_1 \vee q) \wedge (e_1 \vee \neg r)$

Tseitin Transformation [1968]

Used in practice

- No exponential blow-up
- CNF formula size is linear with respect to the original formula

Does not produce an equivalent CNF

However, given F, the following holds for the computed CNF F':

- F' is equisatisfiable to F

- Every model of F' can be translated (i.e., projected) to a model of F

- Every model of F can be translated (i.e., completed) to a model of F'

No model is lost or added in the conversion

DIMACS CNF File Format

Textual format to represent CNF-SAT problems

```
c start with comments
c
c
p cnf 5 3
1 -5 4 0
-1 5 3 4 0
-3 -4 0
```

Format details

- comments start with `c`
- header line: `p cnf nbvar nbclauses`
  - `nbvar` is # of variables, `nbclauses` is # of clauses
- each clause is a sequence of distinct numbers terminating with 0
  - positive numbers are variables, negative numbers are negations

BOUNDED MODEL CHECKING

SAT-based Model Checking

Main idea

Translate the model and the specification to propositional formulas

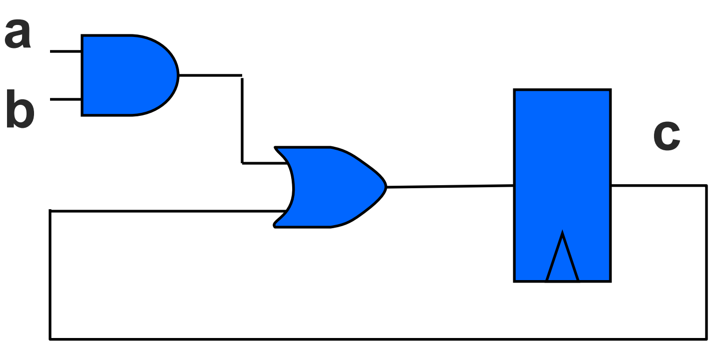
(p, $\neg p$, $p \vee q$, $p \wedge q$, $p \rightarrow q$…)

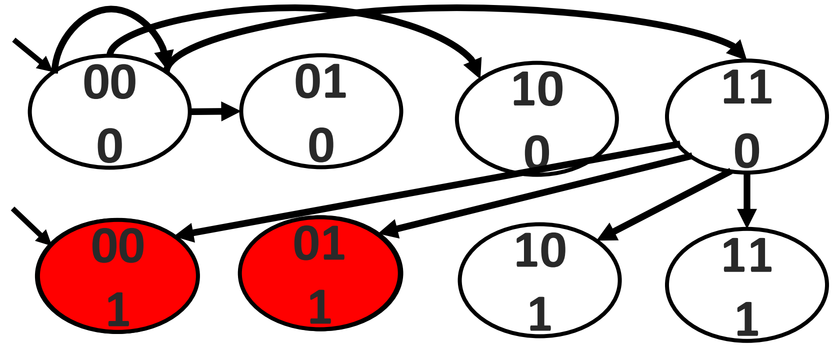
Reduce the model checking problem to satisfiability of propositional formulas

SAT
∈ NPC…

Use efficient tools (SAT solvers) for solving the satisfiability problem

Modeling with Propositional Formulas





Finite-State System is modeled as (V, INIT, T):

- V – finite set of Boolean variables
  - Boolean variables: a b c ➔ 8 states: 000,001,…
- INIT(V) – describes the set of initial states
  - $\text{INIT} = \neg a \wedge \neg b$
- T(V,V') – describes the set of transitions
  - $T(a,b,c,a',b',c') = (c' \leftrightarrow (a \wedge b) \vee c)$

state = valuation to variables

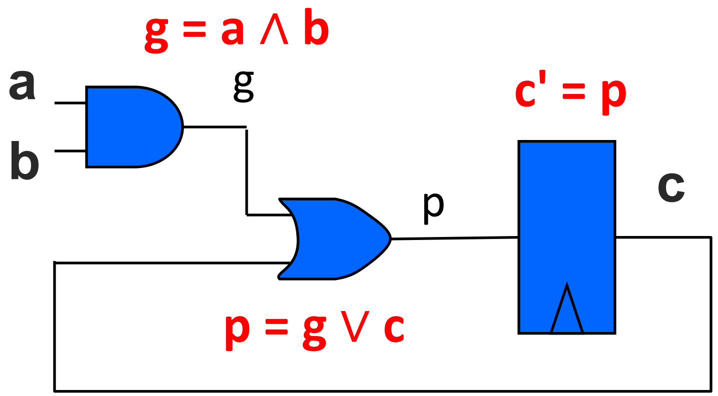
note: $c = c_t$ and $c' = c_{t+1}$

Property:

- p(V) - describes the set of states satisfying p
  - $p = a \vee \neg c$ (Bad $= \neg p = \neg a \wedge c$)

UNIVERSITY OF WATERLOO

Modeling in CNF (Tseitin encoding)



$T(a,b,c,g,p,a',b',c') =$

$g \leftrightarrow a \wedge b,$

$p \leftrightarrow g \vee c,$

$c' \leftrightarrow p$

Each circuit element is a constraint

Bounded model checking (BMC) for checking AGp

Given

- A finite transition system M= (V, INIT(V), T(V,V'))
- A safety property AG p, where p = p(V)
- A bound k

Determine

- Does M contain a counterexample to p of *k transitions (or fewer)* ?

*** BMC can handle all of LTL formulas**

A. Biere, A. Cimatti, E. M. Clarke, M. Fujita, Y. Zhu, DAC'99

Bounded model checking for checking AGp

Unwind the model for k levels, i.e., construct all computations of length k

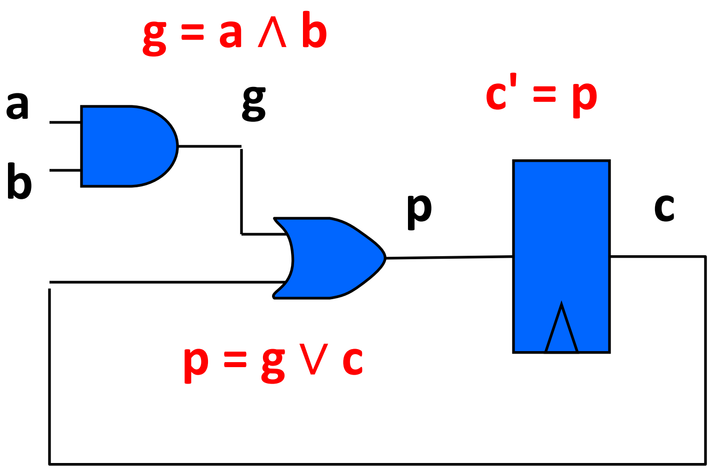
If a state satisfying $\neg$p is encountered, then produce a counterexample

The method is suitable for **falsification**, not verification

Can be translated to a SAT problem

Bounded model checking with SAT

Construct a formula $f_{M,k}$ describing all possible computations of M of length k



$T(a,b,c,a',b',c') =$

$g \leftrightarrow a \wedge b,$

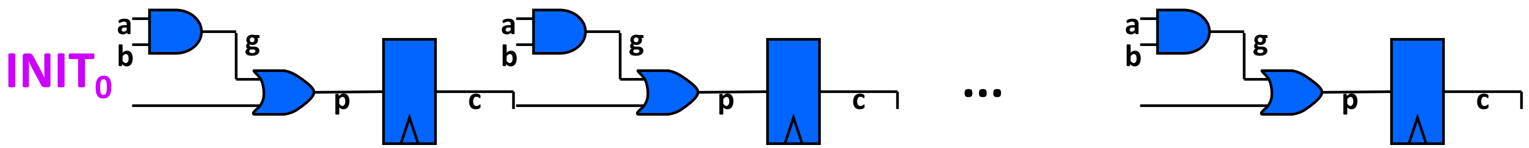
$p \leftrightarrow g \vee c,$

$c' \leftrightarrow p$

Bounded model checking with SAT

Construct a formula $f_{M,k}$ describing all possible computations of M of length k

$$f_{M,k} = INIT_0 \wedge T_0 \wedge T_1 \wedge \ldots \wedge T_{k-1}$$



$a_0,b_0,c_0,$ $a_1,b_1,c_1,$ $a_{k-1},b_{k-1},c_{k-1},$ $a_k,b_k,c_k,$

g_0,p_0 g_1,p_1 g_{k-1},p_{k-1} g_k,p_k

$INIT_0 = INIT(V_0)$

$T_i = T(V_i,V_{i+1})$

Bounded model checking with SAT

Construct a formula $\mathbf{f_{M,k}}$ describing all possible computations of M of length k

Construct a formula $\mathbf{f_{\varphi,k}}$ expressing that $\boldsymbol{\varphi = EF\neg p}$ holds within k computation steps

$\mathbf{f_{\varphi,k}} = \bigvee_{i=0,..k} (\neg \mathbf{p_i})$ [Sometimes $\mathbf{f_{\varphi,k}} = \neg \mathbf{p_k}$]

$\mathbf{p_i = p(V_i)}$

UNIVERSITY OF WATERLOO

Bounded model checking with SAT

Construct a formula $\mathbf{f_{M,k}}$ describing all possible computations of M of length k

Construct a formula $\mathbf{f_{\varphi,k}}$ expressing that $\boldsymbol{\varphi = EF\neg p}$ holds within k computation steps

Check whether $\mathbf{f = f_{M,k} \wedge f_{\varphi,k}}$ is satisfiable

If f is satisfiable then $M \not\models AGp$

The satisfying assignment is a **counterexample**

BMC for checking AG p with SAT

Biere, et al. TACAS99

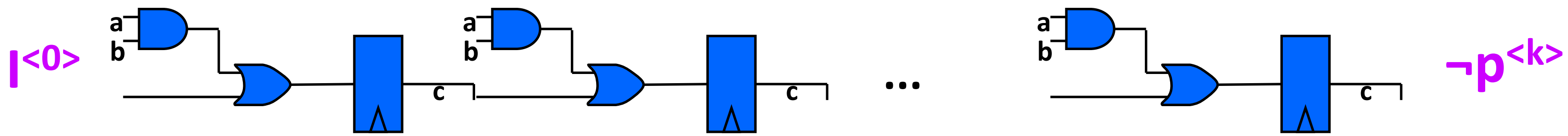
Unfold the model k times:

- $U = T^{<0>} \wedge T^{<1>} \wedge \ldots \wedge T^{<k-1>}$

$I^{<0>} = I(V_0)$

$T^{<i>} = T(V_i, V_{i+1})$

$p^{<k>} = p(V_k)$



- Use SAT solver to check satisfiability of

  $I^{<0>} \wedge U \wedge \neg p^{<k>}$

- If satisfiable: the satisfying assignment describes a counterexample of length k
- If unsatisfiable: property has no counterexample of length k

Example – shift register

Shift register of 3 bits: <x, y, z>

Transition relation:

$$T(x,y,z,x',y',z') = \quad x' \leftrightarrow y \;\wedge\; y' \leftrightarrow z \;\wedge\; \underbrace{z'=1}_{\text{error}}$$

Initial condition:

$$INIT(x,y,z) = \; x=0 \vee y=0 \vee z=0$$

Specification: $AG\,(\, x=0 \vee y=0 \vee z=0)$

Propositional formula for k=2

$f_{M,2} = (x_0=0 \vee y_0=0 \vee z_0=0) \wedge$
$(x_1 \leftrightarrow y_0 \wedge y_1 \leftrightarrow z_0 \wedge z_1=1) \wedge$
$(x_2 \leftrightarrow y_1 \wedge y_2 \leftrightarrow z_1 \wedge z_2=1)$

INIT = $x=0 \vee y=0 \vee z=0$

T = $x' \leftrightarrow y \wedge y' \leftrightarrow z \wedge z'=1$

$f_{\varphi,2} = V_{i=0,..2} (x_i=1 \wedge y_i=1 \wedge z_i=1)$

P = $x=0 \vee y=0 \vee z=0$

Satisfying assignment: 101 011 111

This is a counterexample!

A remark

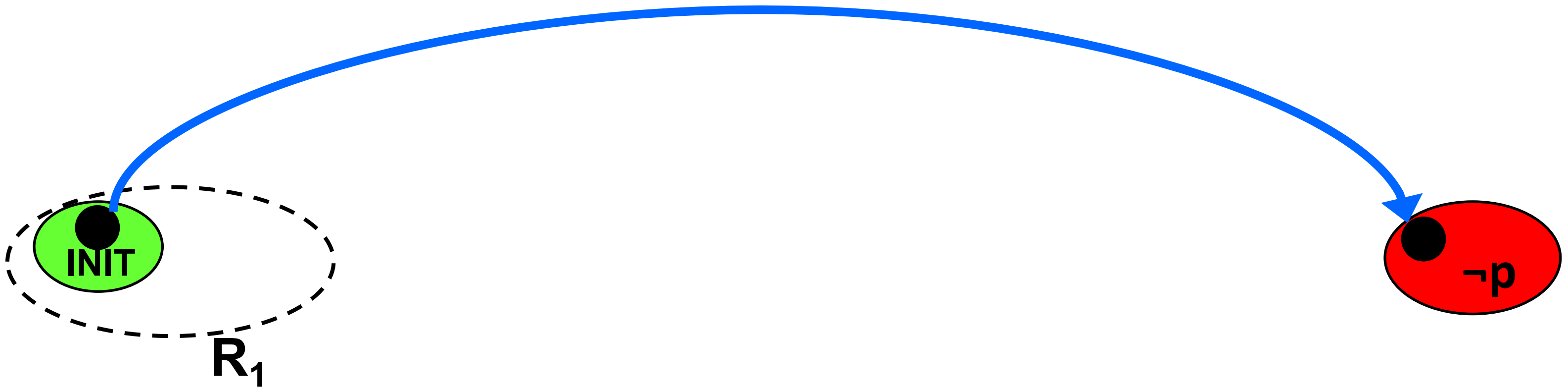
In order to describe a computation of length k by a propositional formula we need k+1 copies of the state variables.

With BDDs we use only two copies: for current and next states.

BMC for checking φ=AGp

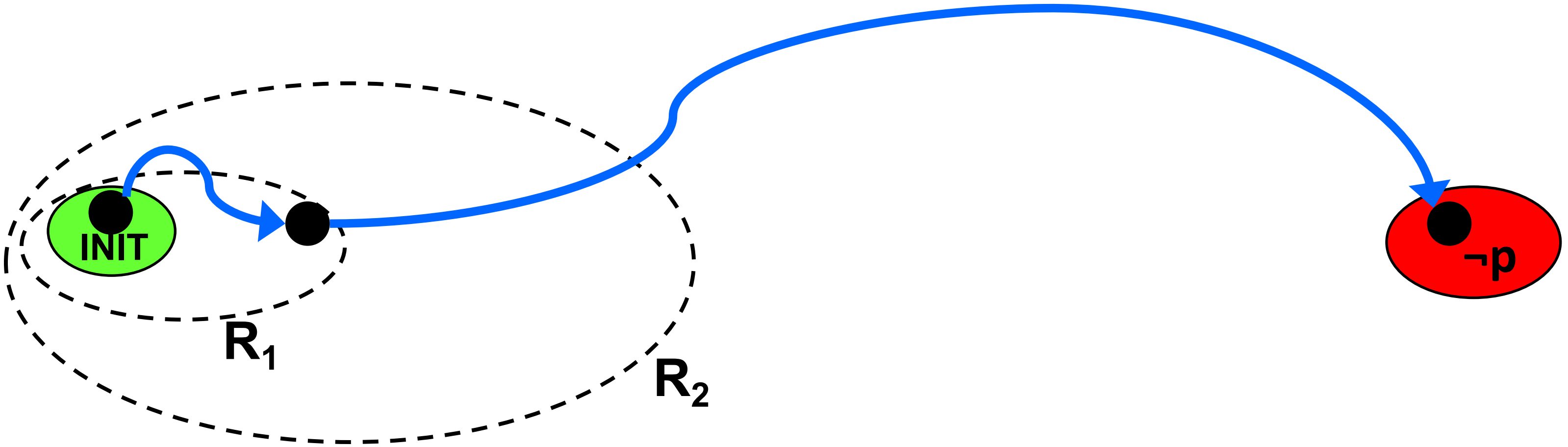
1. $k=1$
2. Build a propositional formula $\mathbf{f_M^k}$ describing all prefixes of length k of paths of M from an initial state
3. Build a propositional formula $\mathbf{f_\varphi^k}$ describing all prefixes of length k of paths satisfying $F\neg p$
4. If $(\mathbf{f_M^k} \wedge \mathbf{f_\varphi^k})$ is **satisfiable**,
   return the satisfying assignment as a **counterexample**
5. Otherwise, increase k and return to 2.

Bounded Model Checking



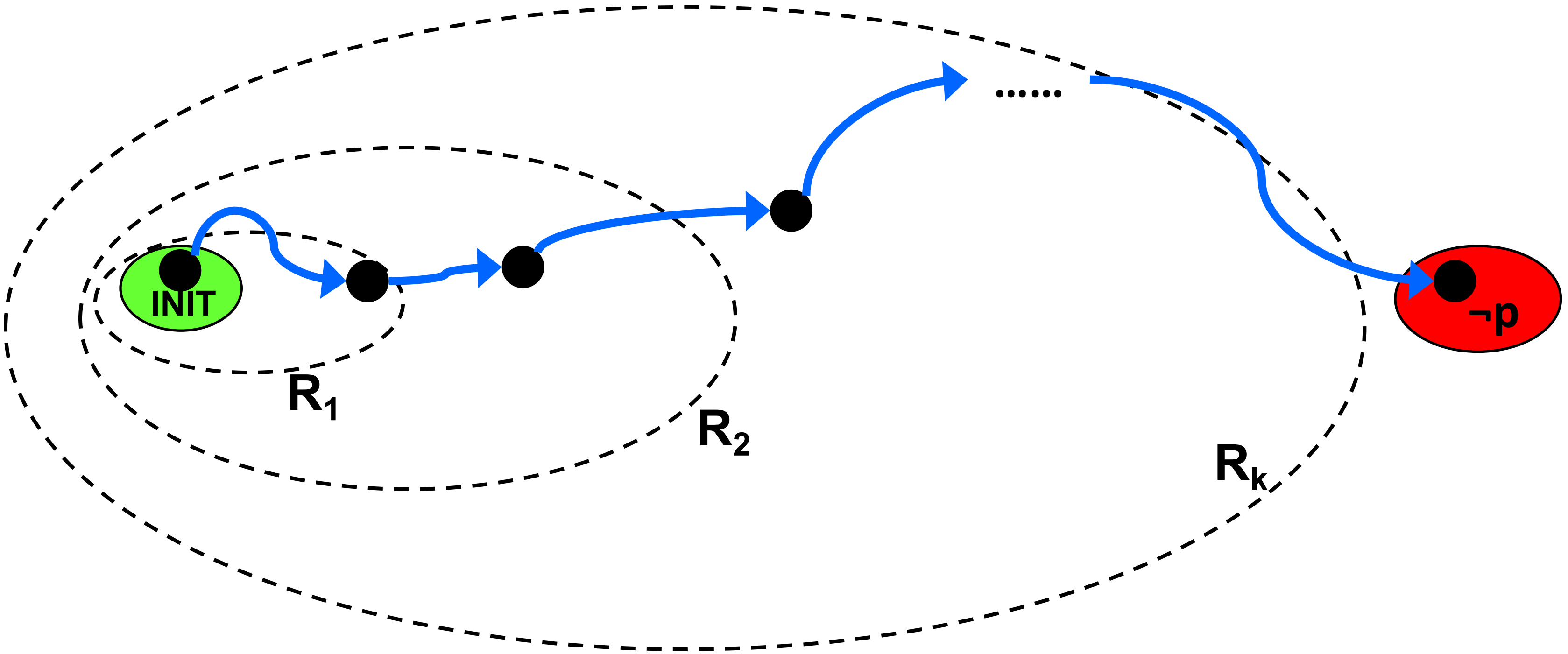
$$\mathrm{INIT}(V^0) \;\wedge T(V^0,V^1) \wedge \neg p(V^1)$$

Bounded Model Checking



$$\mathrm{INIT}(V^0) \quad \wedge T(V^0,V^1) \;\wedge T(V^1,V^2) \wedge \neg p(V^2)$$

Bounded Model Checking



$$INIT(V^0) \wedge T(V^0,V^1) \wedge \ldots \wedge T(V^{k-1},V^k) \wedge \neg p(V^k)$$

UNIVERSITY OF WATERLOO

BMC for checking AFp (φ=EG$\neg$p)

Is there an infinite path in M

- From an initial state
- all of its states satisfying $\neg$**p**
- Over k+1 states ?

If exists, there must also exist a lasso

BMC for checking AFp (φ=EG$\neg$p)

An infinite path in M, from an initial state, over k+1 states, all satisfying $\neg$p:

f_M^k $(V_0,\ldots,V_k)$ =

$$INIT(V_0) \wedge \bigwedge_{i=0,\cdots k-1} T(V_i,V_{i+1}) \wedge \bigvee_{i=0,\cdots k-1} (V_k=V_i)$$

- $V_k=V_i$ means bitwise equality: $\bigwedge_{j=0,\ldots n} (v_{kj} \leftrightarrow v_{ij})$

f_φ^k $(V_0,\ldots,V_k)$ = $\bigwedge_{i=0,\cdots k} \neg p(V_i)$

Remark: BMC can handle all of LTL formulas

Bounded model checking

Can handle all of **LTL** formulas

Can be used for **verification** by choosing k which is large enough

- Need bound on length of the shortest counterexample.
  - *diameter* bound. The diameter is the maximum length of the shortest path between any two states.

Using such k is often **not practical** due to the size of the model

- Worst case diameter is exponential. Obtaining better bounds is sometimes possible, but generally intractable.

Bounded Model Checking

Terminates

- with a counterexample or
- with time- or memory-out

=> The method is suitable for **falsification**, not verification

Can be used for **verification** by choosing k which is large enough

- Need bound on length of the shortest counterexample.
  - *diameter* bound. The diameter is the maximum length of the shortest path between any two states.

Using such k is often **not practical**

- Worst case diameter is exponential. Obtaining better bounds is sometimes possible, but generally intractable.