SAT-based Model Checking

SAT-based Model Checking

- Key problems with BDD's:
  - they can explode in space
  - an expert user can make the difference (e.g. reordering, algorithms)
- A possible alternative:
  - Propositional Satis£biality Checking
  - SAT technology is very advanced
- Advantages:
  - reduced memory requirements
  - limited sensitivity: one good setting, does not require expert users
  - much higher capacity (more variables) than BDD based techniques

DPLL procedure for propositional satis£ability

- Davis-Putnam-Longemann-Loveland
- Input formula in Conjunctive Normal Form:
  - conjunction of clauses $\phi \doteq c_1 \wedge c_2 \wedge \ldots \wedge c_n$
  - clause as disjuction of literals $c_i \doteq l_1^i \vee \ldots \vee l_n^i$
  - literal is either v or $\neg v$
- Incremental construction of satisfying assignment
  - select one variable
  - give it a truth value
  - propagate consequences of assignment
- All clauses must be satis£ed
  - backtrack when satisfying assignment yields false clauses
  - ¤ip previous choice
- Terminate when
  - assignment makes all clauses true (we found a model)
  - all assignments have been explored (the formula has no models)

DPLL Propositional Satis£ability Checking

boolean *TopDPLL*($formula\ \varphi$)
    $\mu = \emptyset$;
    return *DPLL*(φ, μ);

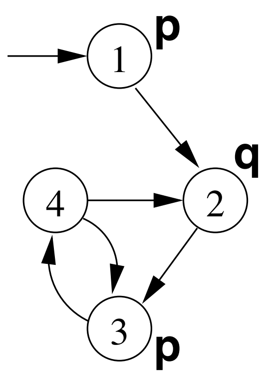
boolean *DPLL*($formula\ \varphi, assignment\ \mu$)
    if ($\varphi == \top$)   /* formula simpli£ed to true */
        return (μ) ;
    if ($\varphi == \bot$)   /* inconsistent assignment */
        return *False*;
    if {a literal l occurs in φ as a unit clause}   /* unit propagation */
        return DPLL($assign(l, \varphi), \mu \cup \{l\}$);
    l = *choose-literal*(φ);   /* split and recur */
    return (DPLL($assign(l, \varphi), \mu \cup \{l\}$) **or**
        DPLL($assign(\neg l, \varphi), \mu \cup \{\neg l\}$));

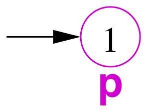
SAT-based Bounded Model Checking

Key ideas:

- look for counter-example paths of increasing length k
  - oriented to £nding bugs
- for each k, builds a boolean formula that is satis£able iff there is a counter-example of length k
  - can be expressed using $k \cdot |\mathbf{s}|$ variables
  - formula construction is not subject to state explosion
- satis£ability of the boolean formulas is checked using a *SAT procedure*
  - can manage complex formulae on several 100K variables
  - returns satisfying assignment (i.e., a counter-example)

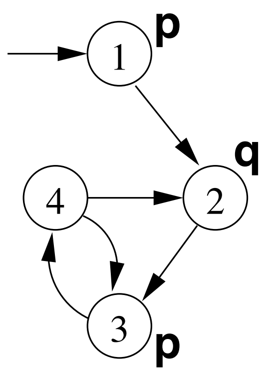
Bounded Model Checking: Example



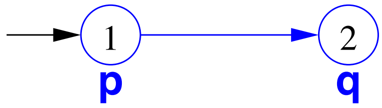
- LTL Formula: **G(p -> F q)**
- Negated Formula (violation): **F(p & G ! q)**
- $k = 0$: 

- No counter-example found.

Bounded Model Checking: Example

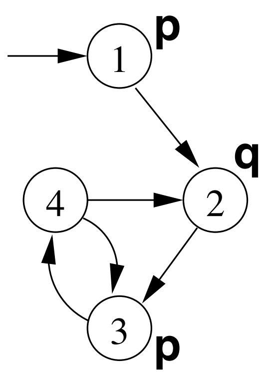


- LTL Formula: **G(p -> F q)**
- Negated Formula (violation): **F(p & G ! q)**
- $k = 1$:

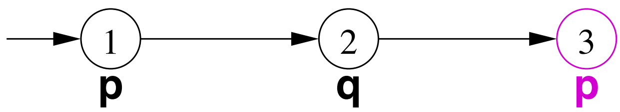


- No counter-example found.

Bounded Model Checking: Example

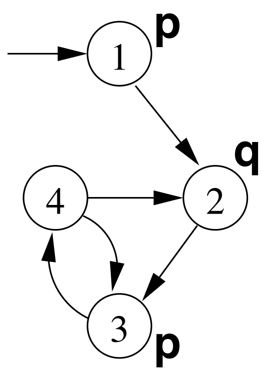


- LTL Formula: **G(p -> F q)**
- Negated Formula (violation): **F(p & G ! q)**
- $k = 2$:

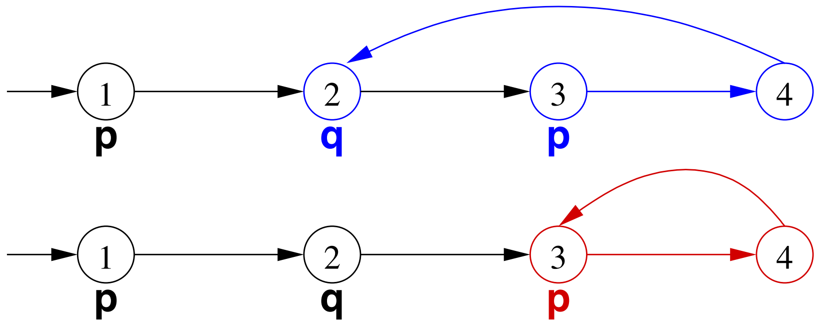


- No counter-example found.

Bounded Model Checking: Example



- LTL Formula: **G(p -> F q)**
- Negated Formula (violation): **F(p & G ! q)**
- $k = 3$:



- The 2nd trace is a counter-example!

SAT-based Bounded Model Checking

- Given a Kripke structure $\mathcal{M}$, an LTL property ϕ and a bound $k \geq 0$:

$$\mathcal{M} \models_k \phi$$

- This is equivalent to the satis£ability problem on formula:

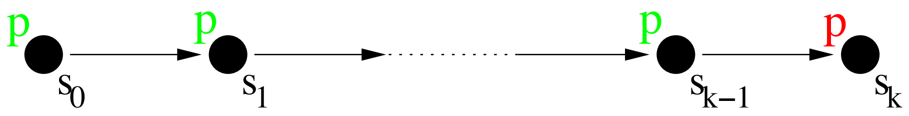
$$[\![\mathcal{M}, \phi]\!]_k \equiv [\![\mathcal{M}]\!]_k \wedge [\![\phi]\!]_k$$

where:
  - the vector of propositional variables is replicated k+1 times
    - $V_0, \ldots, V_k$
  - $[\![\mathcal{M}]\!]_k$ is a k-path compatible with $\mathcal{I}$ and R:
    - $\mathcal{I}(V_0) \wedge R(V_0, V_1) \wedge \ldots R(V_{k-1}, V_k)$
  - $[\![\phi]\!]_k$ encodes the fact that the k-path satis£es ϕ

Model for a Reachability Property

- a £nite path can show that the property holds
- $\phi = \mathrm{F}\, p$

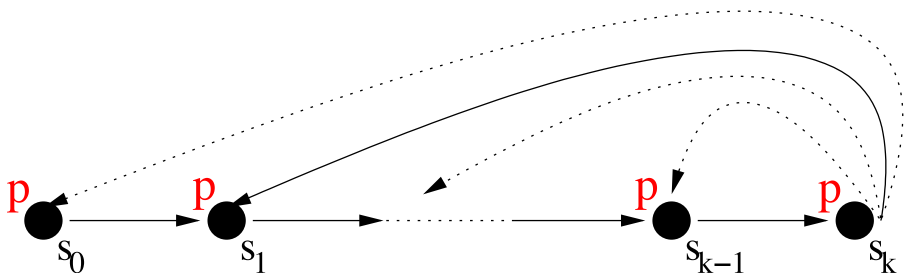
$$[\![\mathrm{F}\, p]\!]_k = \bigvee_{i=0}^{k} p(V_i)$$



The Need for Loop Backs

- We need to produce an in£nite behaviour, with a £nite number of transitions
- We can do it by imposing that the path loops back
- $\phi = \mathrm{G}\, p$

$$[\![\mathrm{G}\, p]\!]_k = \bigvee_{i=0}^{k} \left(R(V_k, V_i) \wedge \bigwedge_{i=0}^{k} p(V_i) \right)$$



Bounded Model Checking Encoding

- In general, the encoding for a formula f with k steps

$$[[f]]_k$$

is the disjunction of

  - the constraints needed to express a model without loopback,

$$(\neg(\bigvee_{l=0}^{k} R(V_k, V_l)) \wedge \ [[f]]_k^0)$$

  - the constraints needed to express a model given a loopback, for all possible points of loopback

$$\bigvee_{l=0}^{k} (R(V_k, V_l) \wedge \ _l[[f]]_k^0)$$