# Applications of Quantified Boolean Formulae Decision Procedures

Armin Biere

Institute for Formal Models and Verification
Johannes Kepler University Linz, Austria

Colloquium
Department of Electrical Engineering
Informatics and Mathematics

University of Paderborn, Germany
5. July 2005

# Quantified Boolean Formulae (QBF)



A. Biere, FMV, JKU Linz

- propositional logic (SAT $\subseteq$ QBF)
  - constants $0, 1$
  - operators $\wedge, \neg, \rightarrow, \leftrightarrow, \ldots$
  - variables $x, y, \ldots$ over boolean domain $\mathbb{B} = \{0, 1\}$
- quantifiers over boolean variables
  - valid $\forall x[\exists y[x \leftrightarrow y]]$ (read $\leftrightarrow$ as $=$)
  - invalid $\exists x[\forall y[x \leftrightarrow y]]$

- semantics given as expansion of quantifiers

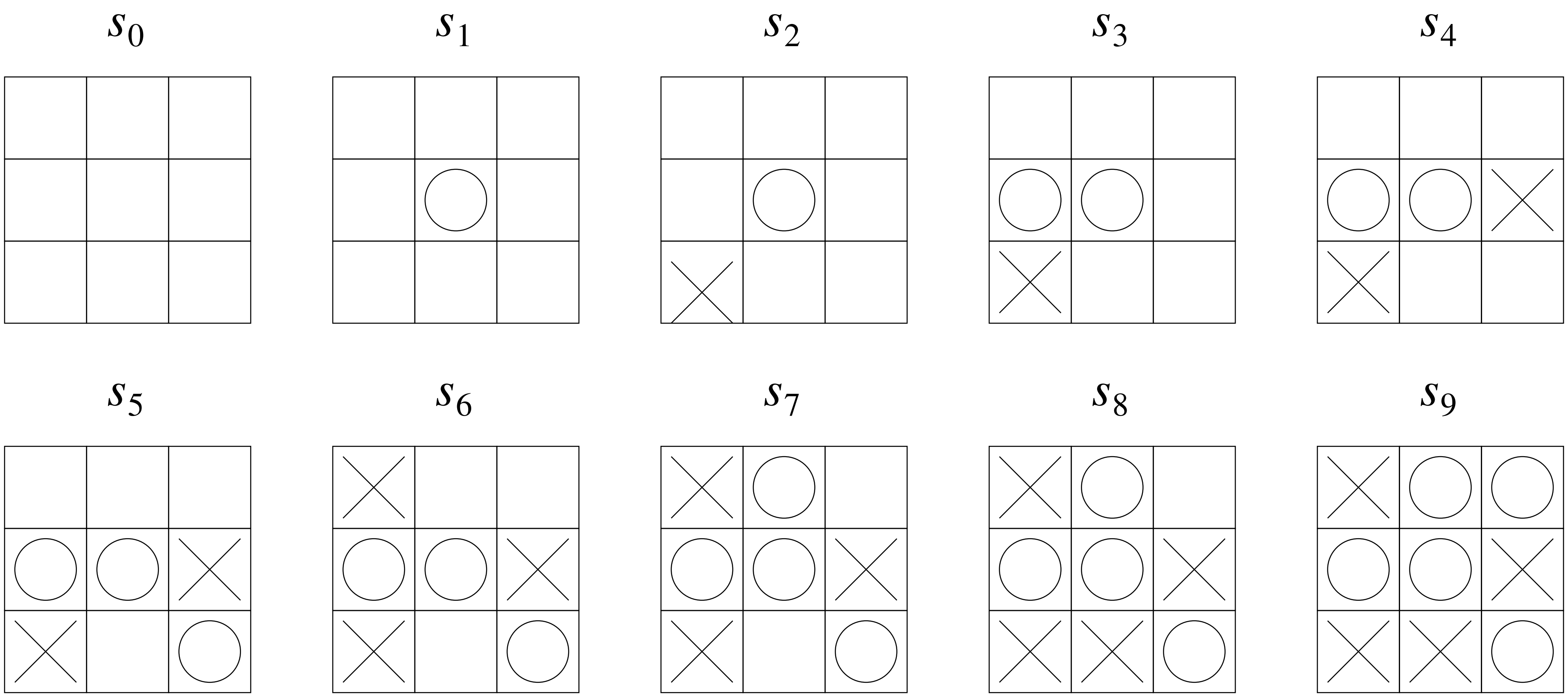
$$\exists x[f] \;\equiv\; f[0/x] \vee f[1/x] \qquad \forall x[f] \;\equiv\; f[0/x] \wedge f[1/x]$$

- expansion as translation from SAT to QBF is exponential
  - SAT problems have only existential quantifiers
  - expansion of universal quantifiers doubles formula size
- most likely no polynomial translation from SAT to QBF
  - otherwise PSPACE = NP

# Tic-Tac-Toe



A. Biere, FMV, JKU Linz



# No Winning Strategy for Tic-Tac-Toe

A. Biere, FMV, JKU Linz

$$
\begin{array}{l}
\forall s_0[\mathit{empty}(s_0) \rightarrow \\
\quad \forall x_1[\mathit{circle}(s_0, x_1, s_1) \rightarrow \\
\quad\quad \exists y_2[\mathit{cross}(s_1, y_2, s_2) \wedge \\
\quad\quad\quad \forall x_3[\mathit{circle}(s_2, x_3, s_3) \rightarrow \\
\quad\quad\quad\quad \exists y_4[\mathit{cross}(s_3, y_4, s_4) \wedge \\
\quad\quad\quad\quad\quad \forall x_5[\mathit{circle}(s_4, x_5, s_5) \rightarrow \\
\quad\quad\quad\quad\quad\quad \exists y_6[\mathit{cross}(s_5, y_6, s_6) \wedge \\
\quad\quad\quad\quad\quad\quad\quad \forall x_7[\mathit{circle}(s_6, x_7, s_7) \rightarrow \\
\quad\quad\quad\quad\quad\quad\quad\quad \exists y_8[\mathit{cross}(s_7, y_8, s_8) \wedge \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad \forall x_9[\mathit{circle}(s_8, x_9, s_9) \rightarrow \neg \mathit{win}(s_9)]]]]]]]]]
\end{array}
$$

$x_i, y_i$ plays (4 bits each)

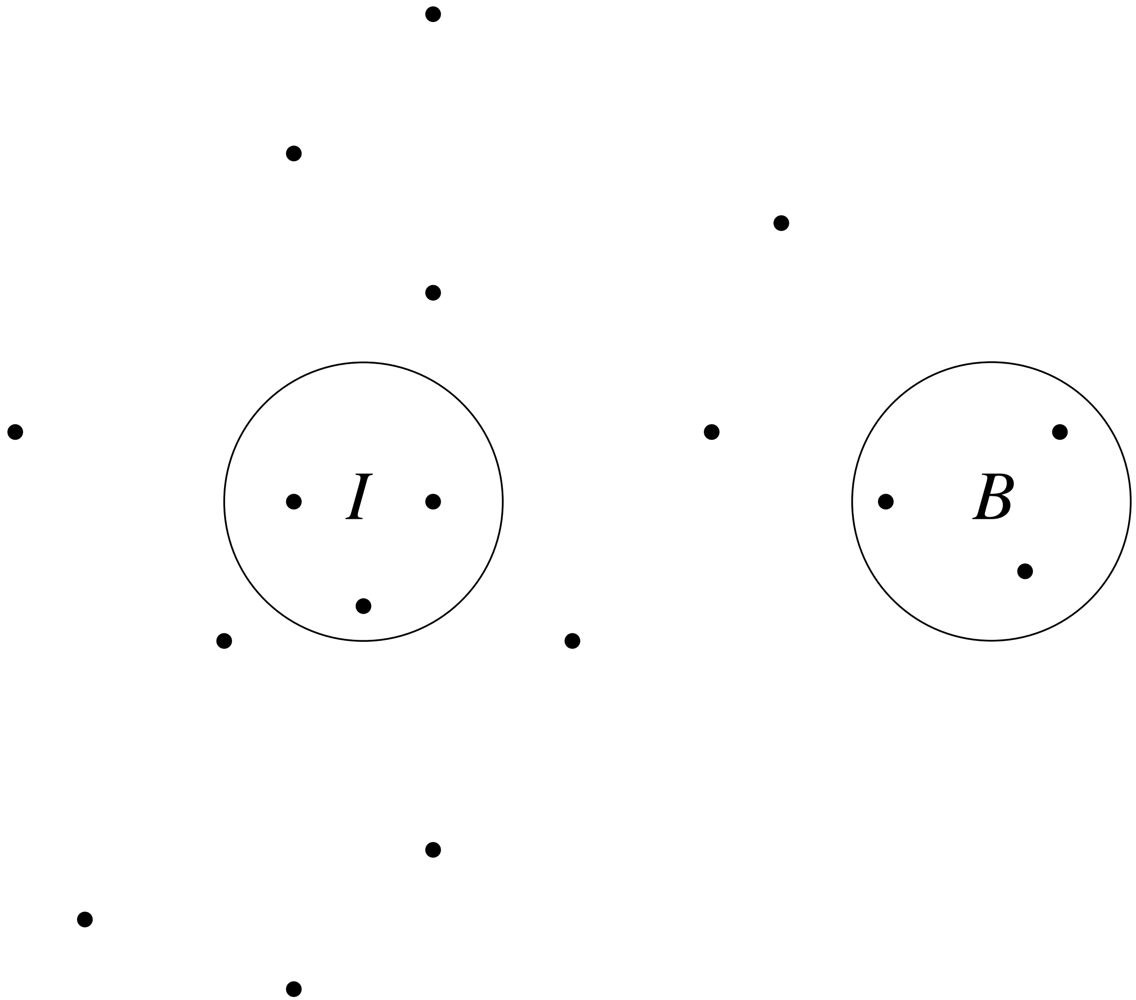
$s_i$ configurations

($9 \times 3$ bits each)

- explicit model checking [ClarkeEmerson’82], [Holzmann’91]
  - program presented symbolically (no transition matrix)
  - traversed state space represented explicitly
  - e.g. reached states are explicitly saved bit for bit in hash table
  
  $\Rightarrow$ State Explosion Problem (state space exponential in program size)

- symbolic model checking [McMillan Thesis’93], [CoudertMadre’89]
  - use symbolic representations for sets of states
  - originally with Binary Decision Diagrams [Bryant’86]
  - Bounded Model Checking using SAT [BiereCimattiClarkeZhu’99]

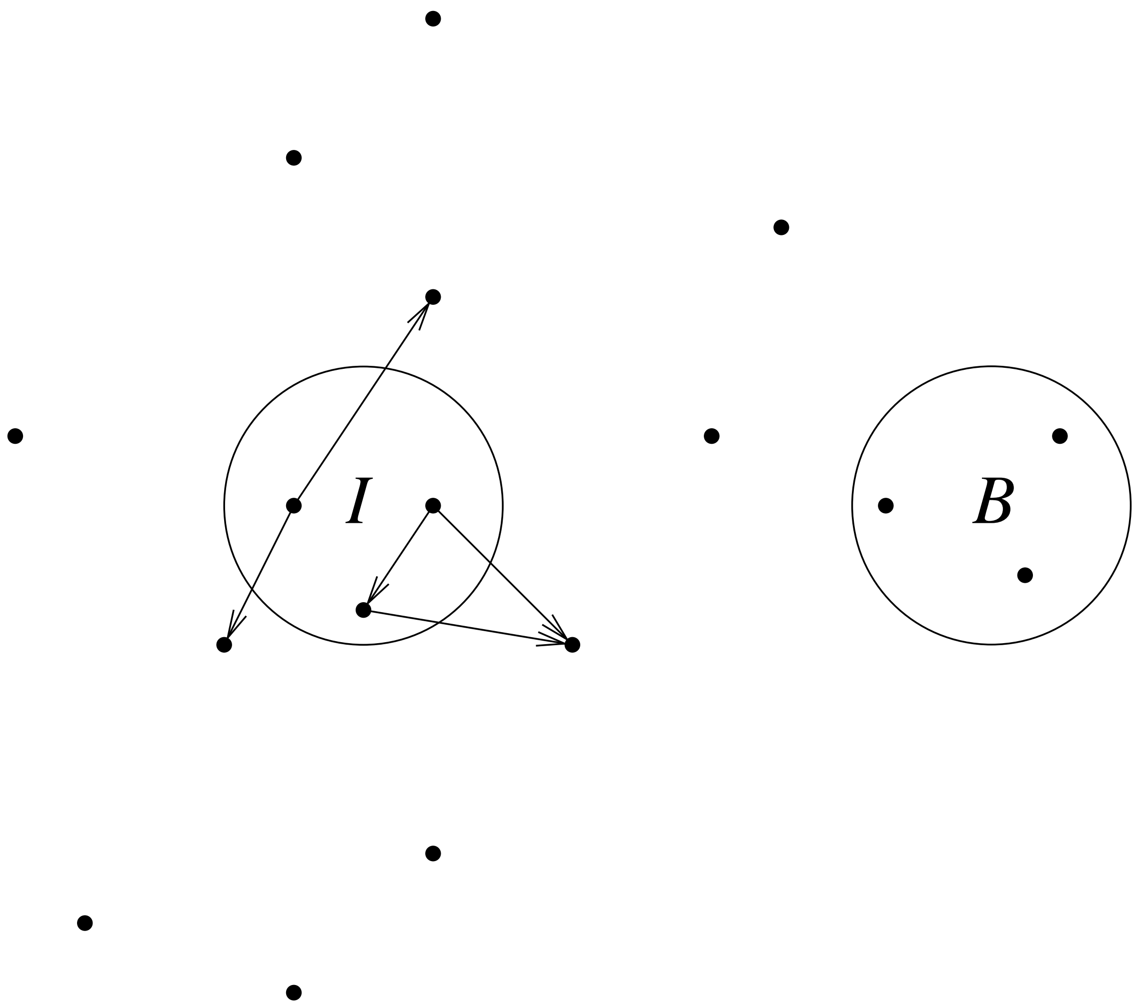
A. Biere, FMV, JKU Linz

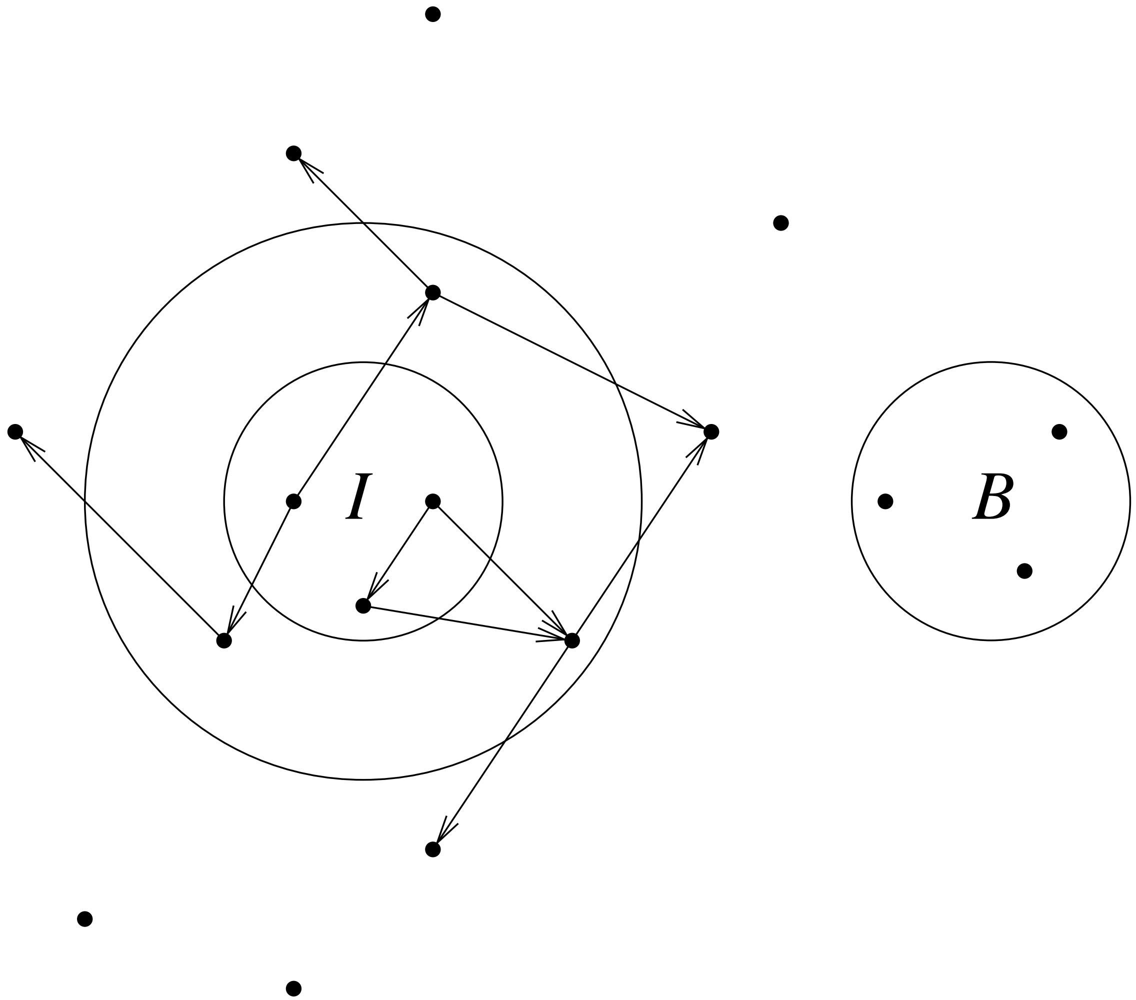


# Forward Fixpoint Algorithm: Step 1



A. Biere, FMV, JKU Linz



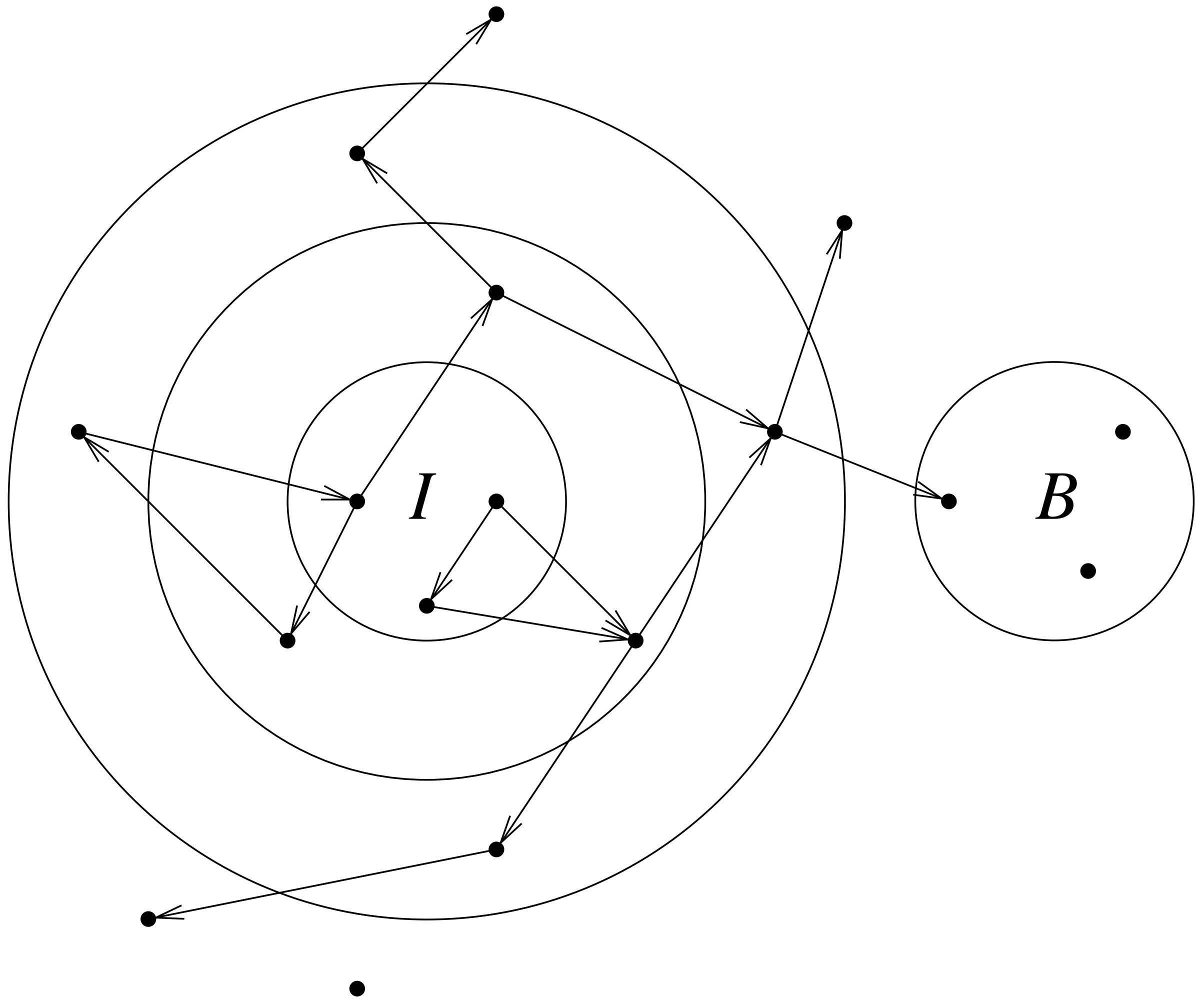


I
B

# Forward Fixpoint Algorithm: Step 3



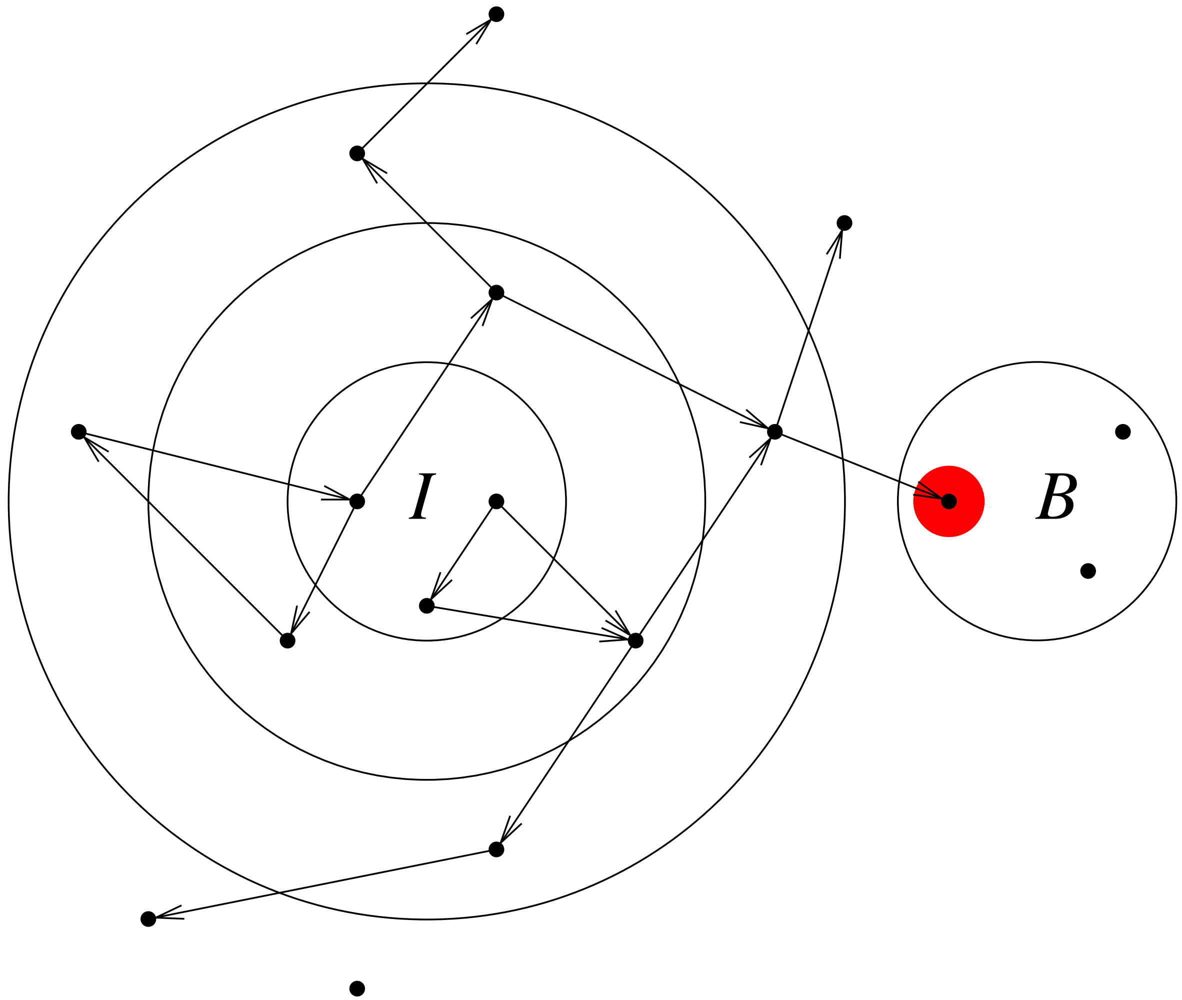
A. Biere, FMV, JKU Linz

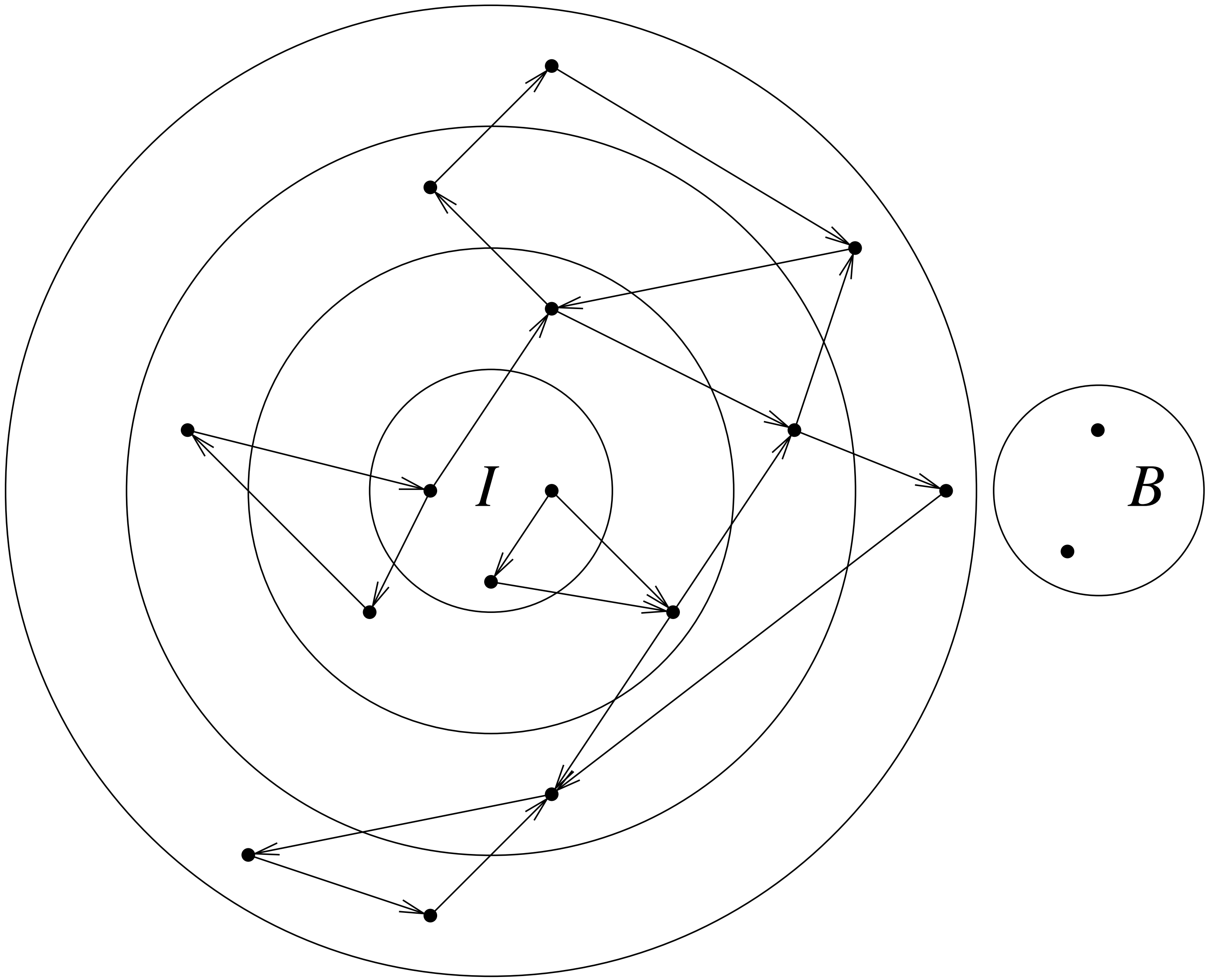


# Forward Fixpoint Algorithm: Bad State Reached



A. Biere, FMV, JKU Linz




I
B

# Forward Least Fixpoint Algorithm for Model Checking Safety

A. Biere, FMV, JKU Linz

initial states $I$, transition relation $T$, bad states $B$

<u>model-check</u>$^{\mu}_{\text{forward}}$ $(I, T, B)$
  $S_C = \emptyset$; $S_N = I$;
  **while** $S_C \neq S_N$ **do**
    **if** $B \cap S_N \neq \emptyset$ **then**
      **return** "found error trace to bad states";
    $S_C = S_N$;
    $S_N = S_C \cup Img(S_C)$;
  **done**;
  **return** "no bad state reachable";

symbolic model checking represents set of states in this BFS symbolically

- BDDs are canonical representation for boolean functions

  – states encoded as bit vectors $\in \mathbb{B}^n$

  – set of states $S \subseteq \mathbb{B}^n$ as BDDs for characteristic function $f_S \colon \mathbb{B}^n \rightarrow \mathbb{B}$

$$f_S(s) = 1 \quad \Leftrightarrow \quad s \in S$$

- for all *set operations* there are linear BDD operations

  – except for $Img$ which is exponential (often also in practice)

$$s \in Img(f) \quad \Leftrightarrow \quad \exists\, t \in \mathbb{B}^n [f(s) \wedge T(s,t)]$$

- variable ordering has strong influence on size of BDDs

- conjunctive partitioning of transition relation is a must

# Unrolling of Forward Least Fixpoint Algorithm

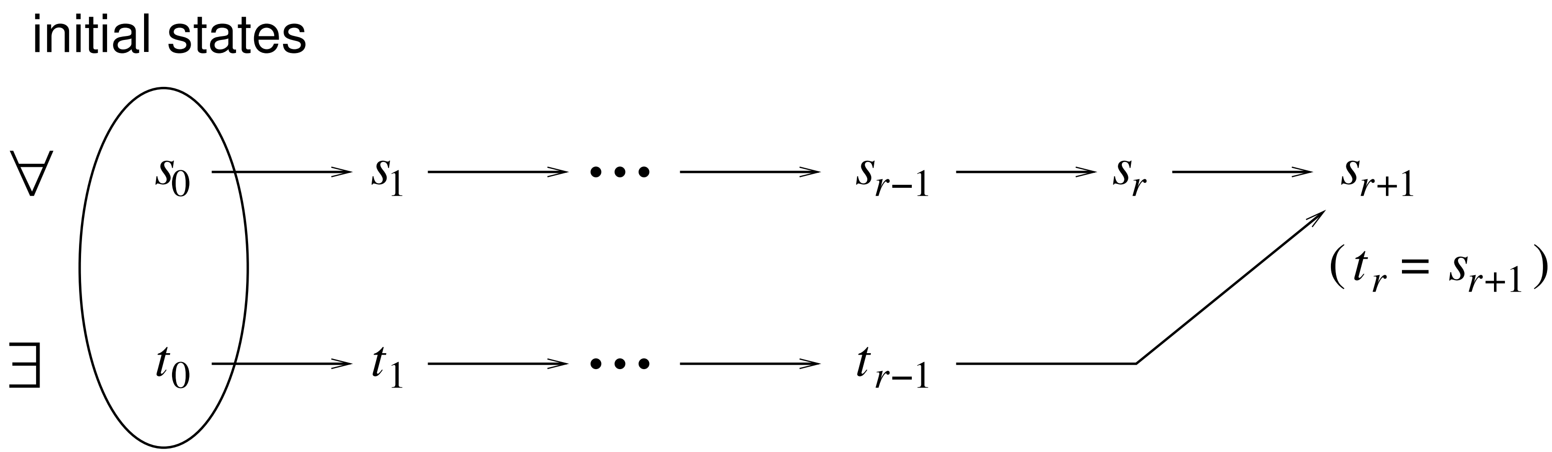


A. Biere, FMV, JKU Linz

| | | |
|---|---|---|
| 0: continue? | $S_C^0 \neq S_N^0$ | $\exists s_0[I(s_0)]$ |
| 0: terminate? | $S_C^0 = S_N^0$ | $\forall s_0[\neg I(s_0)]$ |
| 0: bad state? | $B \cap S_N^0 \neq \emptyset$ | $\exists s_0[I(s_0) \wedge B(s_0)]$ |
| 1: continue? | $S_C^1 \neq S_N^1$ | $\exists s_0, s_1[I(s_0) \wedge T(s_0, s_1) \wedge \neg I(s_1)]$ |
| 1: terminate? | $S_C^1 = S_N^1$ | $\forall s_0, s_1[I(s_0) \wedge T(s_0, s_1) \rightarrow I(s_1)]$ |
| 1: bad state? | $B \cap S_N^1 \neq \emptyset$ | $\exists s_0, s_1[I(s_0) \wedge T(s_0, s_1) \wedge B(s_1)]$ |
| 2: continue? | $S_C^2 \neq S_N^2$ | $\exists s_0, s_1, s2[I(s_0) \wedge T(s_0, s_1) \wedge T(s_1, s_2) \wedge$ $\neg(I(s_2) \vee \exists t_0[I(t_0) \wedge T(t_0, s_2)])]$ |
| 2: terminate? | $S_C^2 = S_N^2$ | $\forall s_0, s_1, s2[I(s_0) \wedge T(s_0, s_1) \wedge T(s_1, s_2) \rightarrow$ $I(s_2) \vee \exists t_0[I(t_0) \wedge T(t_0, s_2)]]$ |
| 2: bad state? | $B \cap S_N^1 \neq \emptyset$ | $\exists s_0, s_1, s_2[I(s_0) \wedge T(s_0, s_1) \wedge T(s_1, s_2) \wedge B(s_2)]$ |

$$\forall s_0, \ldots, s_{r+1} [\, I(s_0) \wedge \bigwedge_{i=0}^{r} T(s_i, s_{i+1}) \rightarrow$$

$$\exists t_0, \ldots, t_r [\, I(t_0) \;\wedge\; s_{r+1} = t_r \;\wedge\; \bigwedge_{i=0}^{r-1} (t_i = t_{i+1} \;\vee\; T(t_i, t_{i+1}))]]$$



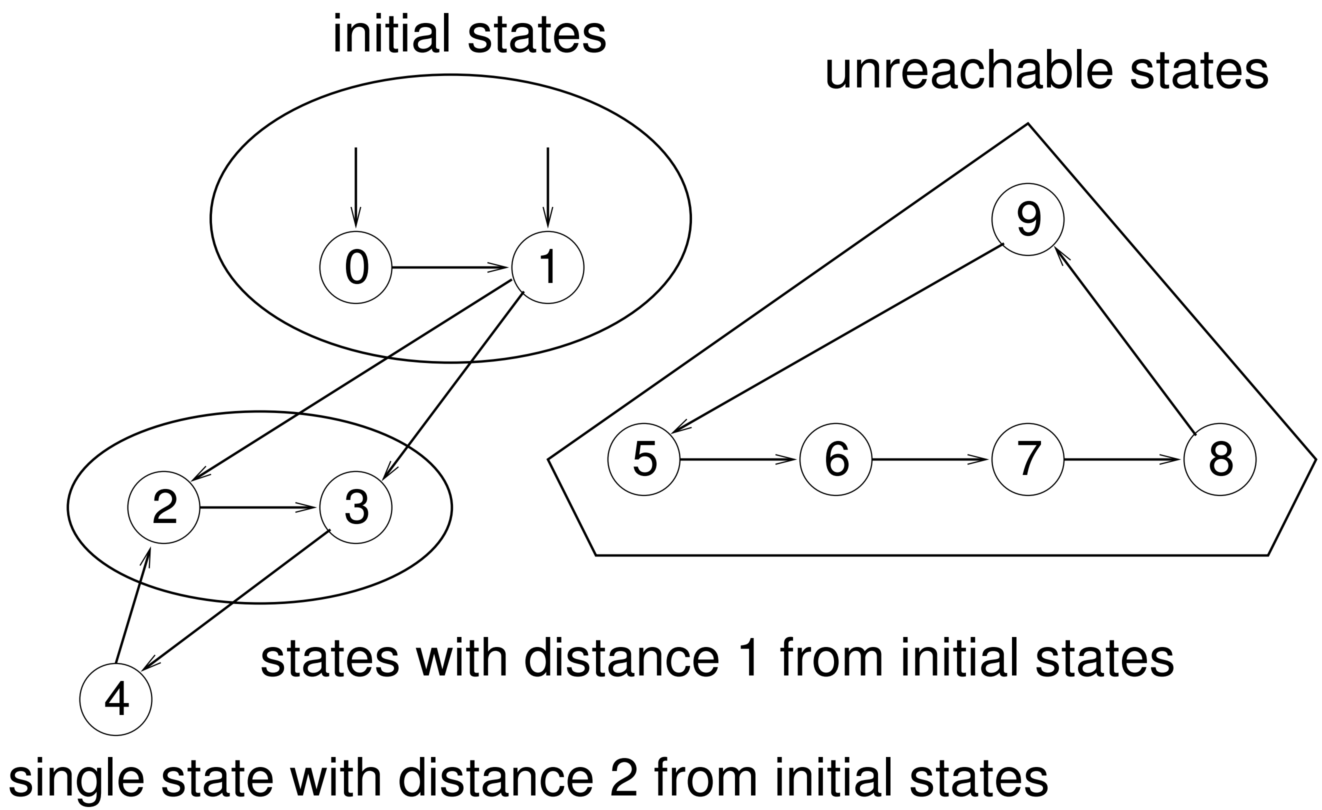

(we allow $t_{i+1}$ to be identical to $t_i$ in the lower path)

**radius** is smallest $r$ for which formula is true

# Radius Example



A. Biere, FMV, JKU Linz



- checking $S_C \neq S_N$ in 2nd iteration results in QBF decision problem

$$\forall s_0, s_1, s2[I(s_0) \wedge T(s_0, s_1) \wedge T(s_1, s_2) \rightarrow I(s_2) \vee \exists t_0[I(t_0) \wedge T(t_0, s_2)]]$$

- not **eliminating quantifiers** results in QBF with one alternation
  - checking whether bad state is reached only needs SAT
  - number iterations bounded by radius $r = O(2^n)$

- so why not forget about termination and concentrate on bug finding?

$\Rightarrow$ Bounded Model Checking (BMC)

# BMC Part of Fixpoint Algorithm



A. Biere, FMV, JKU Linz

~~0: continue? $S_C^0 \neq S_N^0$ $\exists s_0[I(s_0)]$~~

~~0: terminate? $S_C^0 = S_N^0$ $\forall s_0[\neg I(s_0)]$~~

0: bad state? $B \cap S_N^0 \neq \emptyset$ $\exists s_0[I(s_0) \wedge B(s_0)]$

~~1: continue? $S_C^1 \neq S_N^1$ $\exists s_0, s_1[I(s_0) \wedge T(s_0, s_1) \wedge \neg I(s_1)]$~~

~~1: terminate? $S_C^1 = S_N^1$ $\forall s_0, s_1[I(s_0) \wedge T(s_0, s_1) \rightarrow I(s_1)]$~~

1: bad state? $B \cap S_N^1 \neq \emptyset$ $\exists s_0, s_1[I(s_0) \wedge T(s_0, s_1) \wedge B(s_1)]$

~~2: continue? $S_C^2 \neq S_N^2$ $\exists s_0, s_1, s2[I(s_0) \wedge T(s_0, s_1) \wedge T(s_1, s_2) \wedge$~~
$\neg(I(s_2) \vee \exists t_0[I(t_0) \wedge T(t_0, s_2)])]$

~~2: terminate? $S_C^2 = S_N^2$ $\forall s_0, s_1, s2[I(s_0) \wedge T(s_0, s_1) \wedge T(s_1, s_2) \rightarrow$~~
~~$I(s_2) \vee \exists t_0[I(t_0) \wedge T(t_0, s_2)]]$~~

2: bad state? $B \cap S_N^1 \neq \emptyset$ $\exists s_0, s_1, s_2[I(s_0) \wedge T(s_0, s_1) \wedge T(s_1, s_2) \wedge B(s_2)]$

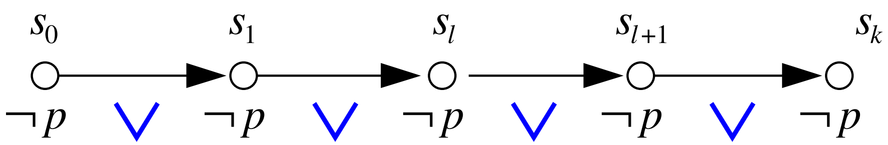
# Bounded Model Checking (BMC)



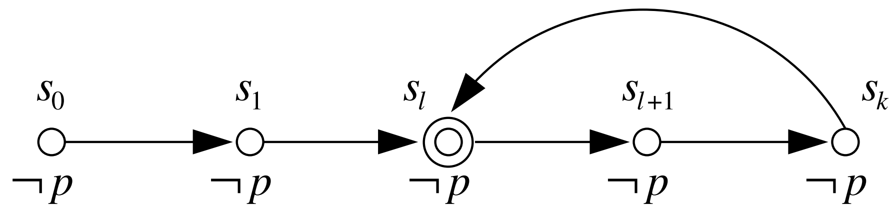
A. Biere, FMV, JKU Linz

[BiereCimattiClarkeZhu TACAS'99]

- look only for counter example made of $k$ states (the bound)




or




- simple for safety properties $\mathbf{G}p$ (e.g. $p = \neg B$)

$$I(s_0) \wedge (\bigwedge_{i=0}^{k-1} T(s_i, s_{i+1})) \wedge \bigvee_{i=0}^{k} \neg p(s_i)$$

- harder for liveness properties $\mathbf{F}p$

$$I(s_0) \wedge (\bigwedge_{i=0}^{k-1} T(s_i, s_{i+1})) \wedge (\bigvee_{l=0}^{k} T(s_k, s_l)) \wedge \bigwedge_{i=0}^{k} \neg p(s_i)$$

- increase in efficiency of SAT solvers [zchaff]

- SAT more robust than BDDs in bug finding

  (shallow bugs are easily reached by explicit model checking or testing)

- better unbounded but still SAT based model checking algorithms

  - $k$-induction [SinghSheeranStålmarck'00]

  - interpolation [McMillan'03]

- 3rd Intl. Workshop on Bounded Model Checking (BMC'05)

  (11. July, Edinburgh, Scotland)

- other logics beside LTL and better encodings

**Transitive Closure**

$$T^* \equiv T^{2^n}$$

**Standard Linear Unfolding**

$$T^{i+1}(s,t) \equiv \exists m[T^i(s,m) \wedge T(m,t)]$$

**Iterative Squaring via Copying**

$$T^{2\cdot i}(s,t) \equiv \exists m[T^i(s,m) \wedge T^i(m,t)]$$

**Non Copying Iterative Squaring**

$$T^{2\cdot i}(s,t) \equiv \exists m[\forall c[\exists l,r[(c \rightarrow (l,r) = (s,m)) \wedge (\overline{c} \rightarrow (l,r) = (m,t)) \wedge T^i(l,r)]]]$$

```
dpll-sat(Assignment S)          [DavisLogemannLoveland62]
    boolean-constraint-propagation()
    if contains-empty-clause() then return false
    if no-clause-left() then return true
    v := next-unassigned-variable()
    return  dpll-sat(S ∪ {v ↦ false}) ∨ dpll-sat(S ∪ {v ↦ true})
```

```
dpll-qbf(Assignment S)          [CadoliGiovanardiSchaerf98]
    boolean-constraint-propagation()
    if contains-empty-clause() then return false
    if no-clause-left() then return true
    v := next- outermost -unassigned-variable()
    @ := is-existential(v) ? ∨ : ∧
    return  dpll-sat(S ∪ {v ↦ false}) @ dpll-sat(S ∪ {v ↦ true})
```

Why is QBF harder than SAT?

$$\models \quad \forall x \,.\, \exists y \,.\, (x \leftrightarrow y)$$

$$\not\models \quad \exists y \,.\, \forall x \,.\, (x \leftrightarrow y)$$

Decision order matters!

- most implementations DPLL alike: [Cadoli. . .98][Rintanen01]
  - **learning** was added [Giunchiglia. . .01] [Letz01] [ZhangMalik02]
  - top-down: split on variables from the outside to the inside
- multiple quantifier elimination procedures:
  - **enumeration** [PlaistedBiereZhu03] [McMillan02]
  - **expansion** [Aziz-Abdulla. . .00] [WilliamsBiere. . .00] [AyariBasin02]
  - bottom-up: eliminate variables from the inside to the outside
- **q-resolution** [KleineBüning. . .95], with **expansion** [Biere04]
- symbolic representations [PanVardi04] [Benedetti05] BDDs

A. Biere, FMV, JKU Linz

- **collect** variables in scopes, **order** scopes according to nesting depth:

$$\underbrace{\exists\, a,b,c,d.}_{\text{scope 0}} \quad \underbrace{\forall x,y,z.}_{\text{scope 1}} \quad \underbrace{\exists\, r,s,t.}_{\text{scope 2}} \quad (c \vee d)(a \vee \overline{c} \vee \overline{x} \vee y)(\overline{a} \vee x \vee s)(t \vee \ldots)\ \cdots$$

**attach** clauses to the scope of its innermost variables

- **remove** innermost univ. literals in clauses attached to univ. scopes:

$$(a \vee \overline{c} \vee \overline{x} \vee y) \quad \text{simplifies to} \quad (a \vee \overline{c})$$

- q-resolution = resolution + forall reduction [KleineBüning...95]

- all clauses are forall reduced

  $\Rightarrow$ innermost scope is always existential

  $\Rightarrow$ no clauses attached to universal scopes

- normalized structure of quantified CNF:

$$\Omega(S_1)\, S_1 \,.\quad \Omega(S_2)\, S_2 \,.\quad \ldots \quad \forall S_{m-1} \,.\quad \exists\, S_m \,.\quad f \wedge g \qquad m \geq 2$$

$$\begin{aligned} f &\equiv \text{clauses of scope} \quad S_m \\ g &\equiv \text{clauses of outer scopes} \quad S_i, \quad i < m-1 \\ S_\exists &\equiv S_m \\ S_\forall &\equiv S_{m-1} \end{aligned}$$

# Resolve and Expand Algorithm

resolve-and-expand()

**forever**

simplify()

**if** contains-empty-clause() **then return** *false*

**if** no-clause-left() **then return** *true*

**if** is-propositional() **then return** sat-solve($\emptyset$)

$v$ := schedule-cheapest-to-eliminate-variable()

**if** is-existential($v$) **then** resolve($v$)

**if** is-universal($v$) **then** expand($v$)

**original** clauses in which $v$ or $\neg v$ occurs:

$\neg r \vee v$

$s \vee v$

$x \vee y \vee v$

$\otimes$

$\neg v \vee r$

$\neg v \vee \neg x \vee \neg y \vee r$

**add** forall reduced non-trivial resolvents:

$(s \vee r)$, $(x \vee y \vee r)$, and $(s \vee \neg x \vee \neg y \vee r)$

**remove** original clauses

# Expand

A. Biere, FMV, JKU Linz

one-to-one mapping of variables: $u \in S_\exists$ mapped to $u' \in S'_\exists$

before expansion:

$$\Omega(S_1)\,S_1\,.\quad \Omega(S_2)\,S_2\,.\quad \ldots\quad \forall S_\forall\qquad .\quad \exists\,S_\exists\qquad .\qquad f \wedge g$$

after expansion:

$$\Omega(S_1)\,S_1\,.\quad \Omega(S_2)\,S_2\,.\quad \ldots\quad \forall(S_\forall - \{v\}).\quad \exists(S_\exists \cup S'_\exists).\quad f\{v/0\} \wedge f'\{v/1\} \wedge g$$

A. Biere, FMV, JKU Linz

- elimination cost: number of expected added literals

$$
\begin{aligned}
o(l) &\equiv \text{number of clauses with literal } l \\
s(l) &\equiv \text{sum of lengths of clauses with literal } l \\
s(S) &\equiv \text{sum lengths of clauses with scope } S
\end{aligned}
$$

- expansion cost: $\mathrm{s}(\mathrm{S}_\exists) \quad - \quad \Big(\mathrm{s}(\mathrm{v}) + \mathrm{s}(\neg\mathrm{v}) + \mathrm{o}(\mathrm{v}) + \mathrm{o}(\neg\mathrm{v})\Big)$

- resolution cost:

$$\mathrm{o}(\neg\mathrm{v}) \cdot \Big(\mathrm{s}(\mathrm{v}) - \mathrm{o}(\mathrm{v})\Big) + \mathrm{o}(\mathrm{v}) \cdot \Big(\mathrm{s}(\neg\mathrm{v}) - \mathrm{o}(\neg\mathrm{v})\Big) - \Big(\mathrm{s}(\mathrm{v}) + \mathrm{s}(\neg\mathrm{v})\Big)$$

A. Biere, FMV, JKU Linz

| | benchmark family | #inst | **decide** | **qube** | **semprop** | *expand* | **quantor** |
|---|---|---|---|---|---|---|---|
| 1 | adder* | 16 | 2 | 2 | 2 | 1 | **3** |
| 2 | Adder2* | 14 | 2 | 2 | 2 | 2 | **3** |
| 3 | C[0-9]* | 27 | 2 | 3 | 2 | 3 | **4** |
| 4 | CHAIN* | 11 | 10 | 7 | **11** | 4 | **11** |
| 5 | comp* | 5 | 4 | 4 | **5** | **5** | **5** |
| 6 | flip* | 7 | 6 | **7** | **7** | **7** | **7** |
| 7 | impl* | 16 | 12 | **16** | **16** | **16** | **16** |
| 8 | k* | 171 | 77 | 91 | 97 | 60 | **108** |
| 9 | mutex* | 2 | 1 | **2** | **2** | **2** | **2** |
| 10 | robots* | 48 | 0 | **36** | **36** | 15 | 24 |
| 11 | term1* | 4 | 2 | **3** | **3** | 1 | **3** |
| 12 | toilet* | 260 | 187 | **260** | **260** | 259 | 259 |
| 13 | TOILET* | 8 | **8** | 6 | **8** | **8** | **8** |
| 14 | tree* | 12 | 10 | **12** | **12** | 8 | **12** |
| #(among best in family) | | | **1** | **7** | **10** | **5** | **12** |
| #(single best in family) | | | **0** | **0** | **0** | **0** | **4** |

(families with no difference and two actually random families removed)

- rectification problems (actually a synthesis problem)

$$\exists p[\forall i[g(i,p) = s(i)]]$$

with parameters $p$, inputs $i$, generic circuit $g$, and specification $s$

- games, open systems, non-deterministic planning applications?

- model checking
  - termination check as in classical (BDD based) model checking

    (only one alternation)
  - acceleration as in PSPACE completeness for QBF proof

    (at most linear number of alternations in number of state bits $n$)

# QBF in Software Verification



[CookKröningSharygina – SMC'05]

- model: asynchronous boolean programs
  parallel version of those used in SLAM, BLAST or MAGIC

- symbolic representation of set of states
  - related work uses BDDs
  - [CookKröningSharygina] boolean formulas

- termination check for reachability (partially explicit)
  - trivial with BDDs as symbolic representation
  - QBF decision procedure for boolean formulas $\Leftarrow$ [quantor]

- SAT/QBF version seems to scale much better than BDDs

## Summary

- applications fuel interest in SAT
  - incredible capacity increase in recent years
    (instances with thousands or million variable are regularly solved)
  - SAT solver competition affiliated to SAT conference
  - SAT is becoming a core verification technology

- QBF is catching up
  - solvers are getting better
  - new applications
  - richer structure

# Solved Hard Instances of SAT'03 QBF Evaluation: QUANTOR

A. Biere, FMV, JKU Linz

| | hard instance | time | space | ∀ | ∃ | units | pure | subsu. | subst. | ∀red. |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Adder2-6-s | 29.6 | 19.7 | 90 | 13732 | 126 | 13282 | 174081 | 0 | 37268 |
| 2 | adder-4-sat | 0.2 | 2.8 | 42 | 1618 | 0 | 884 | 6487 | 0 | 960 |
| 3 | adder-6-sat | 36.6 | 22.7 | 90 | 13926 | 0 | 7290 | 197091 | 0 | 54174 |
| 4 | C49*1.*_0_0* | 27.9 | 13.3 | 1 | 579 | 0 | 0 | 48 | 84 | 0 |
| 5 | C5*1.*_0_0* | 56.2 | 16.0 | 2 | 2288 | 10 | 0 | 4552 | 2494 | 0 |
| 6 | k_path_n-15 | 0.1 | 0.8 | 32 | 977 | 66 | 82 | 2369 | 2 | 547 |
| 7 | k_path_n-16 | 0.1 | 0.8 | 34 | 1042 | 69 | 85 | 2567 | 2 | 597 |
| 8 | k_path_n-17 | 0.1 | 0.9 | 36 | 1087 | 72 | 100 | 3020 | 2 | 639 |
| 9 | k_path_n-18 | 0.1 | 0.9 | 36 | 1146 | 76 | 106 | 3242 | 2 | 725 |
| 10 | k_path_n-20 | 0.1 | 0.9 | 38 | 1240 | 84 | 149 | 3967 | 2 | 855 |
| 11 | k_path_n-21 | 0.1 | 1.0 | 40 | 1318 | 84 | 130 | 4470 | 2 | 909 |
| 12 | k_t4p_n-7 | 15.5 | 105.8 | 43 | 88145 | 138 | 58674 | 760844 | 8 | 215 |
| 13 | k_t4p_p-8 | 5.8 | 178.6 | 29 | 12798 | 206 | 5012 | 85911 | 4 | 138 |
| 14 | k_t4p_p-9 | 0.3 | 4.5 | 32 | 4179 | 137 | 1389 | 23344 | 10 | 142 |
| 15 | k_t4p_p-10 | 27.9 | 152.9 | 35 | 130136 | 193 | 63876 | 938973 | 4 | 137 |
| 16 | k_t4p_p-11 | 86.0 | 471.5 | 38 | 196785 | 204 | 79547 | 1499430 | 4 | 140 |
| 17 | k_t4p_p-15 | 84.6 | 354.7 | 50 | 240892 | 169 | 181676 | 1336774 | 9 | 226 |
| 18 | k_t4p_p-20 | 3.6 | 16.1 | 65 | 27388 | 182 | 21306 | 197273 | 11 | 325 |

time in seconds, space in MB

- specific workshop: Satisfiability Modulo Theories (SMT'05)

- examples

  - processor verification [BurchDill CAV'94] [VelevBryant JSC'03]

  - translation validation [PnueliStrichmanSiegel'98]

- eager approach: translate into SAT

- lazy approach

  - augment SAT solver to handle non-propositional constraints

  - in each branch: SAT part satisfiable, check non-propositional theory

# Examples for Using SAT in Software Verification



A. Biere, FMV, JKU Linz

- [JacksonVaziri ISSTA'00] Alloy
  - bounded model checking of OO modelling language Alloy
  - checks properties of symbolic simulations with bounded heap size
- [KroeningClarkeYorav DAC03] CBMC
  - targets equivalence checking of hardware models
  - bounded model checking of C resp. Verilog programs
- [XieAiken POPL'05] Saturn
  - LINT for lock usage in large C programs (latest Linux kernel)
  - neither sound nor complete, but 179 bugs out of 300 warnings