Learning to integrate deduction and search in reasoning about quantified Boolean formulas

Luca Pulina and Armando Tacchella

Laboratory of Machine Intelligence for Diagnosis (MIND-Lab)
Department of Computer, Communication and System Sciences (DIST)
University of Genoa - Italy

What is a quantified Boolean Formula?

Consider a Boolean formula in **conjunctive normal form** (CNF), e.g.,

$$\underbrace{(x_1 \vee x_2)}_{\textit{clause}} \wedge (\overbrace{\neg x_1}^{\textit{literal}} \vee \overbrace{x_2}^{\textit{literal}})$$

Adding **existential** "$\exists$" and **universal** "$\forall$" quantifiers, e.g.,

$$\underbrace{\forall x_1 \exists x_2}_{\textit{prefix}} \underbrace{(x_1 \vee x_2) \wedge (\neg x_1 \vee x_2)}_{\textit{matrix}}$$

yields a **quantified Boolean formula** (QBF).

What is the meaning of a QBF?

The QBF

$$\forall x_1 \exists x_2 (x_1 \vee x_2) \wedge (\neg x_1 \vee x_2)$$

is true if and only if

> *for **every value of** x_1 there **exist a value of** x_2 such that $(x_1 \vee x_2) \wedge (\neg x_1 \vee x_2)$ is propositionally satisfiable.*

Given any QBF ψ:

- if $\psi = \forall x \varphi$ then ψ is true iff $\varphi_{|x=0} \wedge \varphi_{|x=1}$ is true
- if $\psi = \exists x \varphi$ then ψ is true iff $\varphi_{|x=0} \vee \varphi_{|x=1}$ is true

Problem QSAT

Decide whether a given QBF is true or false.

Why QBFs?

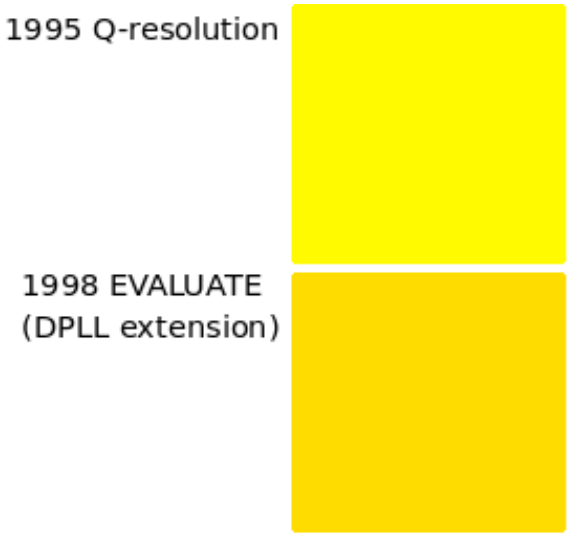
- QSAT is **PSPACE-complete**, i.e., the (supposedly) hardest class of problems for which we could not prove EXPTIME-hardness.

- Several reasoning tasks admit a **compact** QBF encoding
  - **Conformant planning**: does there **exist** a sequence of actions such that **for all** initial conditions we can reach the goal?
  - **"Black box" circuit verification**: does there **exist** a set of inputs to a circuit such that **for all** possible realizations of some of its modules, the output is not correct?
  - **Adversarial games**: does there **exist** a sequence of moves such that **for all** possible counter-moves of my adversary I am guaranteed to win?

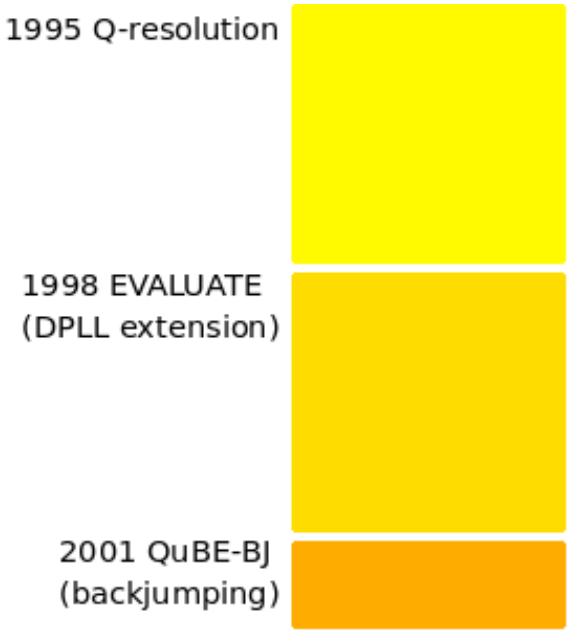
A capsule history of reasoning with QBFs

1995 Q-resolution

A capsule history of reasoning with QBFs

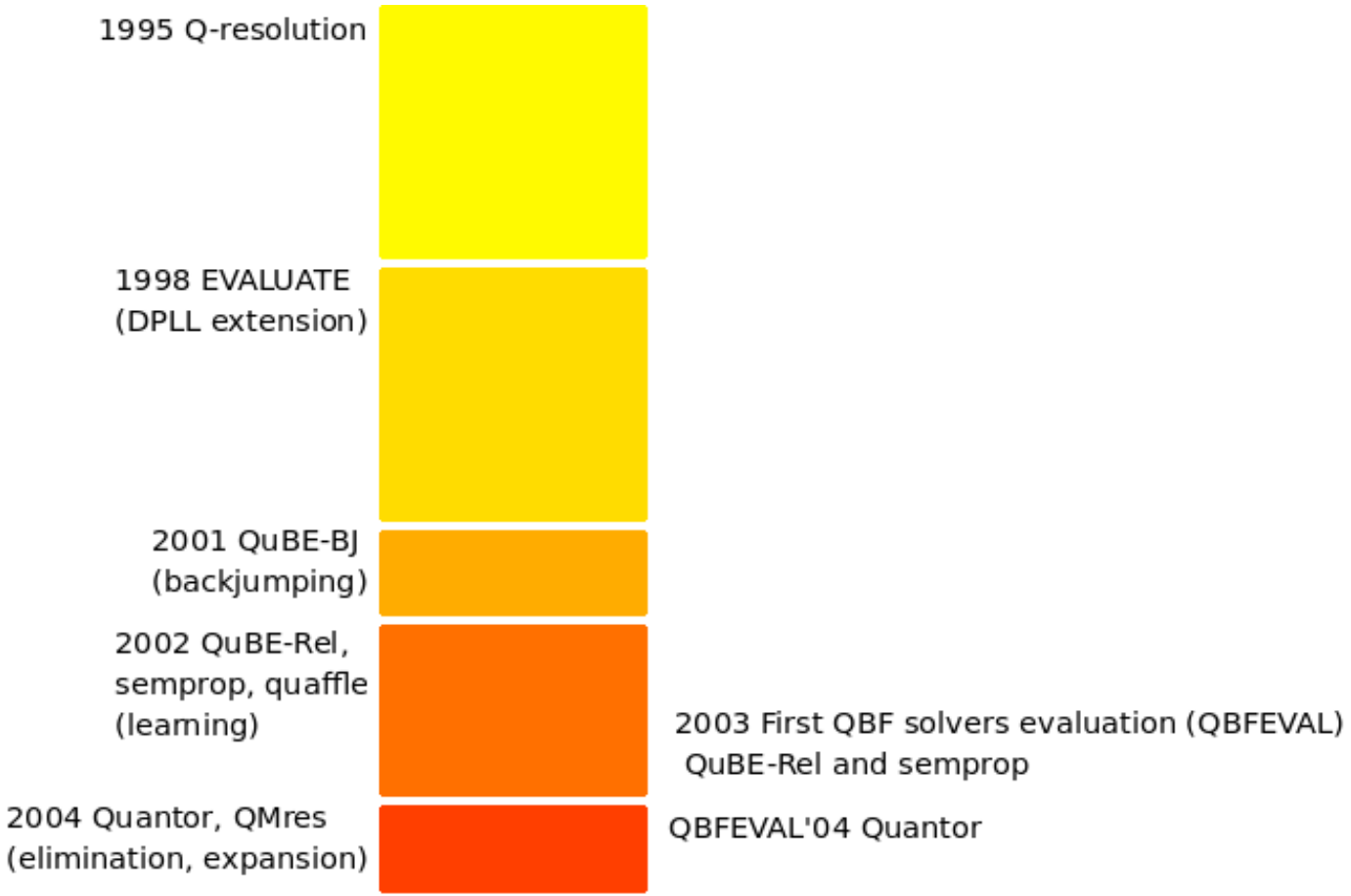


A capsule history of reasoning with QBFs

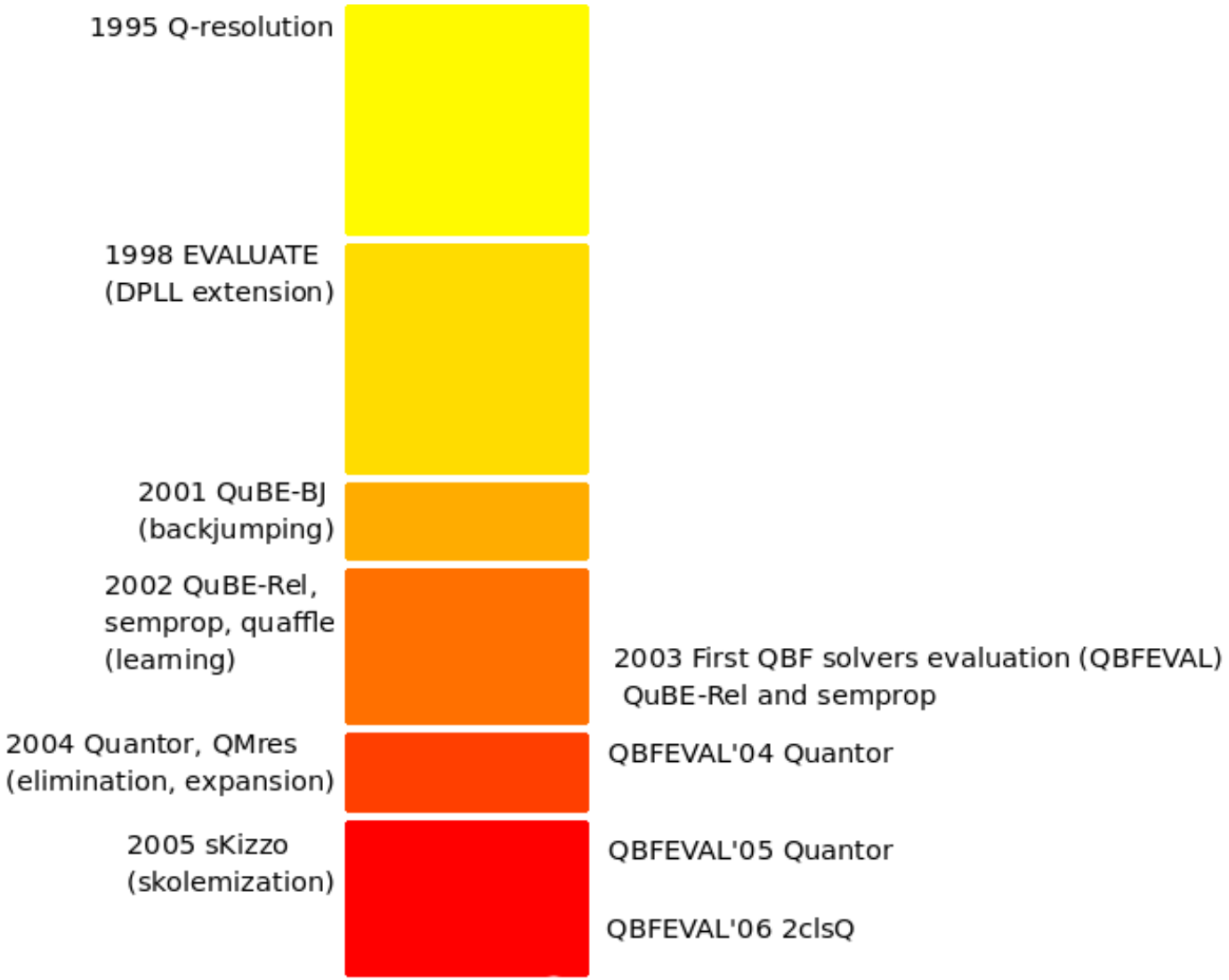


A capsule history of reasoning with QBFs

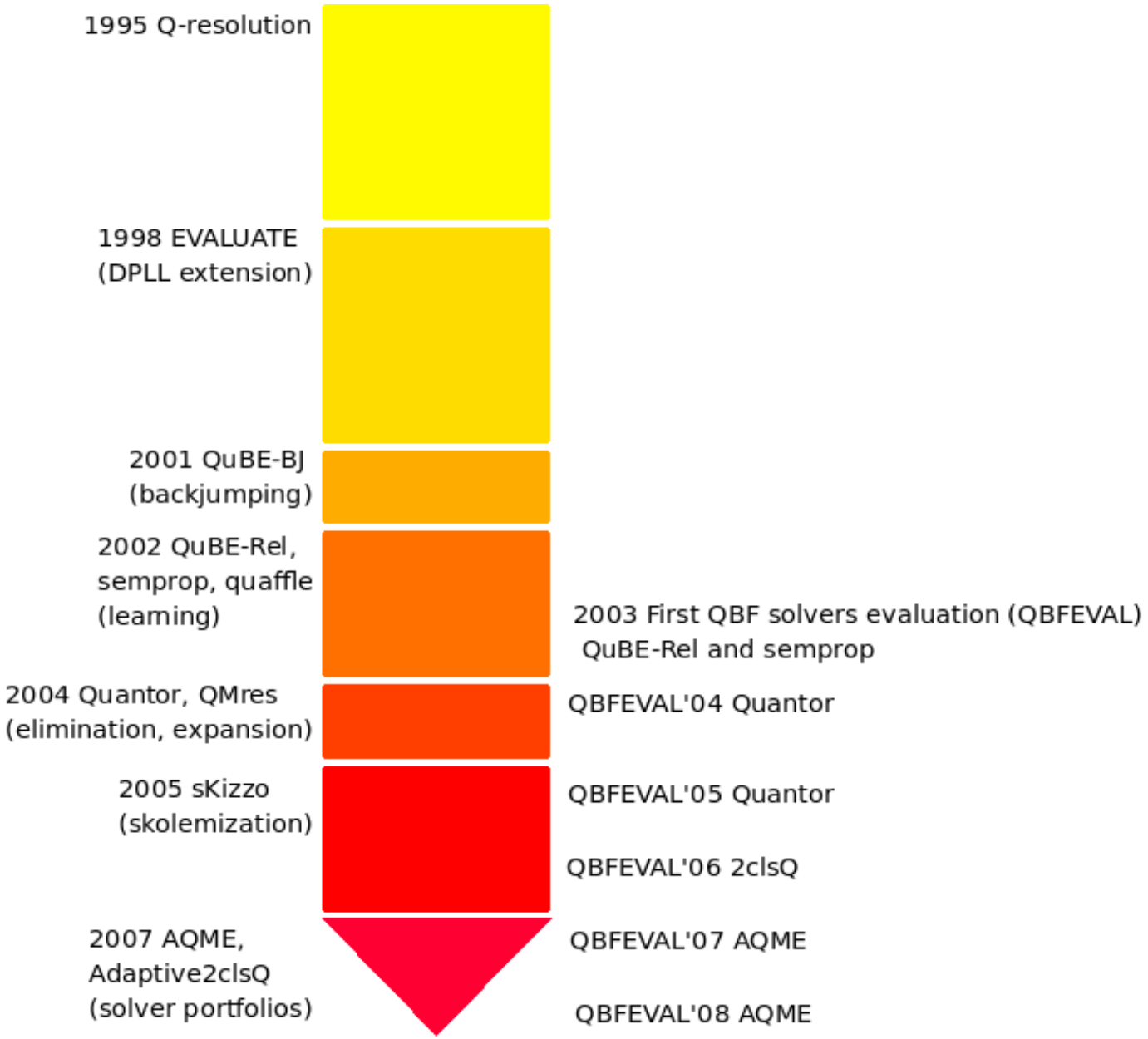
A capsule history of reasoning with QBFs



A capsule history of reasoning with QBFs



A capsule history of reasoning with QBFs



Our research: motivations, aims, current results

In QBFEVAL'08 even the **most sophisticated** QBF solvers **failed** on encodings from several **real-world** applications.

Our research: motivations, aims, current results

In QBFEVAL'08 even the **most sophisticated** QBF solvers **failed** on encodings from several **real-world** applications.

Our (long term?) research goal is **QBF reasoning made practical**.

Our research: motivations, aims, current results

In QBFEVAL'08 even the **most sophisticated** QBF solvers **failed** on encodings from several **real-world** applications.

Our (long term?) research goal is **QBF reasoning made practical**.

In this paper we show that

- **Integrating** search and resolution enables a **structure-aware** QBF solver to outperform both of them.
- **Machine learning** can be used to derive an effective combination strategy from example runs.
- The resulting solver is **competitive** with sophisticated state-of-the-art tools.

Agenda

1. Structure and basic algorithms
   - QBFs as graphs
   - Resolution
   - Search
2. Learning to integrate resolution and search
   - Learning to reason in QuReS
   - Experimental results
3. Final remarks
   - QuReS performances: can we do better?
   - Conclusions and future work

What is the structure of a QBF?

QBF φ
(prenex CNF)

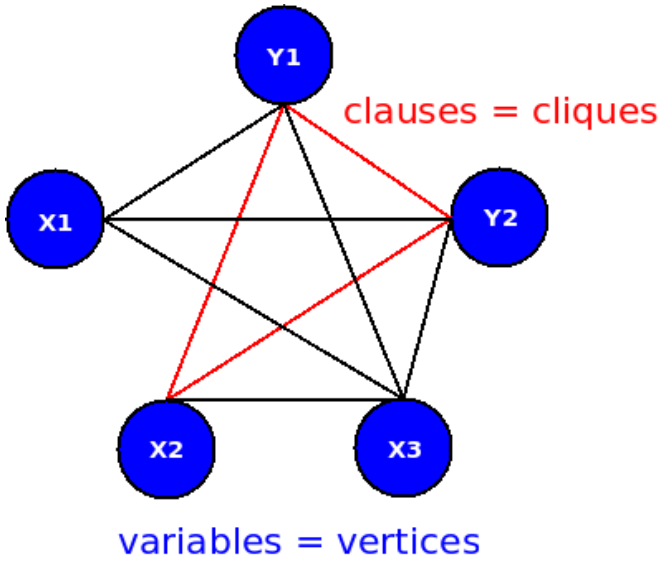
$$\overbrace{\forall y_1 \exists x_1 \forall y_2 \exists x_2 \exists x_3}^{\textit{variables}} \overbrace{(y_1 \vee y_2 \vee x_2)}^{\textit{clause}} \wedge (y_1 \vee \neg y_2 \vee \neg x_2 \vee \neg x_3) \wedge (y_1 \vee \neg x_2 \vee x_3) \wedge (\neg y_1 \vee x_1 \vee x_3) \wedge (\neg y_1 \vee y_2 \vee x_2) \wedge (\neg y_1 \vee y_2 \vee \neg x_2) \wedge (\neg y_1 \vee \neg x_1 \vee \neg y_2 \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_3)).$$

What is the structure of a QBF?

QBF φ
(prenex CNF)

$$\overbrace{\forall y_1 \exists x_1 \forall y_2 \exists x_2 \exists x_3}^{\textit{variables}} \overbrace{(y_1 \vee y_2 \vee x_2)}^{\textit{clause}} \wedge (y_1 \vee \neg y_2 \vee \neg x_2 \vee \neg x_3) \wedge (y_1 \vee \neg x_2 \vee x_3) \wedge (\neg y_1 \vee x_1 \vee x_3) \wedge (\neg y_1 \vee y_2 \vee x_2) \wedge (\neg y_1 \vee y_2 \vee \neg x_2) \wedge (\neg y_1 \vee \neg x_1 \vee \neg y_2 \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_3)).$$

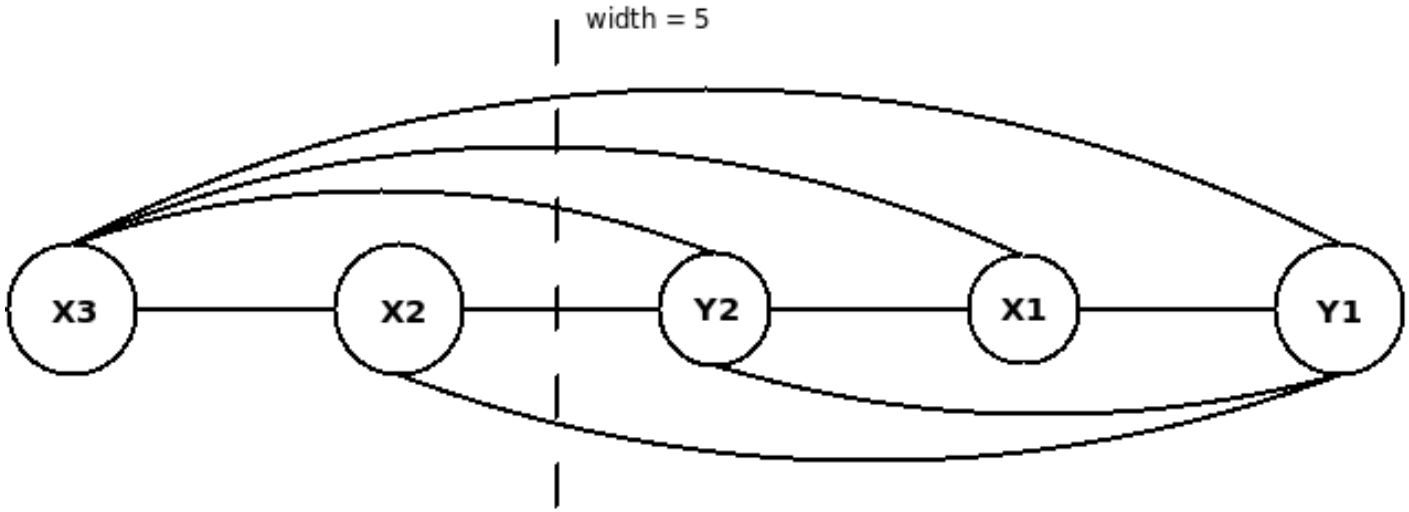
Gaifman
graph G_φ



Treewidth and QBFs (1)

The **Treewidth** $tw(G)$ of a graph $G = (V, E)$ measures its "tree-likeness"

- Given an ordering σ on the vertices of G we can define as "parents" of $v \in V$ its neighbors $u \in V$ s.t. $\sigma(u) \leq \sigma(v)$.
- The **width** of G along σ is the maximum number of parents of a node.

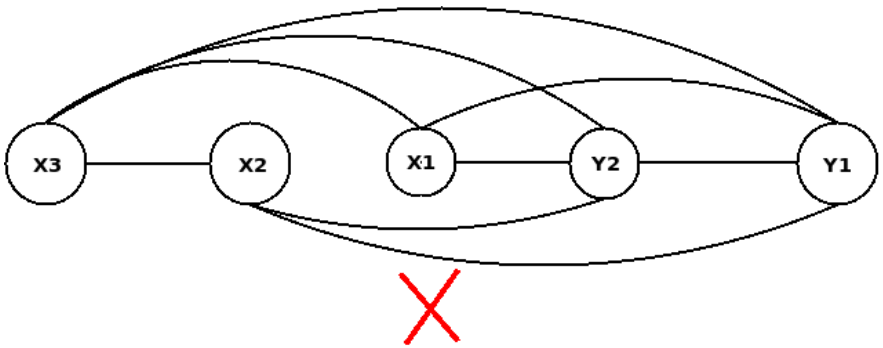


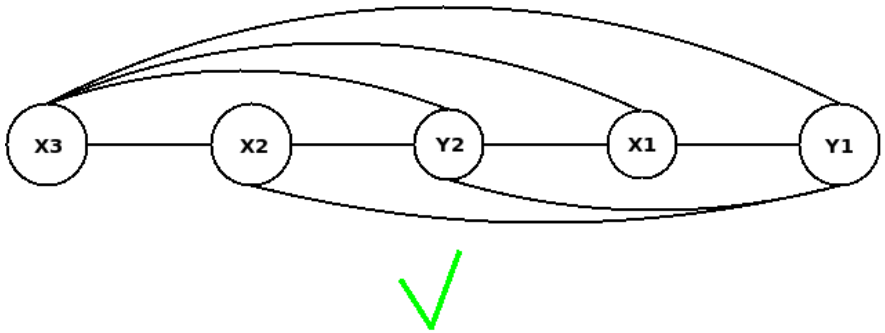
- **Treewidth** is the minimum width along all orderings for G.

Treewidth and QBFs (2)

When it comes to QBFs:

- $tw(G_\varphi) = tw(\varphi)$ is **not enough** to capture the structure of φ
- **Prefix matters**: orderings cannot bypass alternations.
- **Quantified treewidth** $tw_p(\varphi)$ minimizes over orderings which are compatible with the prefix of φ.
- Because of this, $tw_p(\varphi) \geq tw(\varphi)$.





Deduction

Q-resolution

Given two clauses $Q \vee x$ and $R \vee \neg x$ of a QBF φ, the clause $\min(Q \vee R)$ can be derived, where

- x is an **existential** variable,
- Q and R do not share any literal l such that l occurs in Q and $\bar{l}$ occurs in R, and
- $\min(C)$ is obtained from C by removing the **universal literals** appearing in the prefix of φ **after all the existential literals** in C.

Deduction

Q-resolution

Given two clauses $Q \vee x$ and $R \vee \neg x$ of a QBF φ, the clause $\min(Q \vee R)$ can be derived, where

- x is an **existential** variable,
- Q and R do not share any literal l such that l occurs in Q and $\bar{l}$ occurs in R, and
- $\min(C)$ is obtained from C by removing the **universal literals** appearing in the prefix of φ **after all the existential literals** in C.

Variable Elimination

Backtrack-free **control strategy**. Given a QBF φ:

- Start from the innermost variable in the prefix of φ.
- If it is universal, eliminate it, or
- if it is existential, perform all the possible Q-resolutions on it.

Stop when all variables are eliminated (φ is true) or an emtpy clause is derived (φ is false).

Q-resolution: an example

$$\forall y_1 \exists x_1 \forall y_2 \exists x_2 \exists x_3 ((y_1 \vee y_2 \vee x_2)_1 \wedge (y_1 \vee \neg y_2 \vee \neg x_2 \vee \neg x_3)_2 \wedge (y_1 \vee \neg x_2 \vee x_3)_3 \wedge (\neg y_1 \vee x_1 \vee x_3)_4 \wedge (\neg y_1 \vee y_2 \vee x_2)_5 \wedge (\neg y_1 \vee y_2 \vee \neg x_2)_6 \wedge (\neg y_1 \vee \neg x_1 \vee \neg y_2 \vee \neg x_3)_7 \wedge (\neg x_2 \vee \neg x_3)_8)$$

Resolving away the variable x_3 yields the QBF:

$$\forall y_1 \exists x_1 \forall y_2 \exists x_2 ((y_1 \vee y_2 \vee x_2)_1 \wedge (y_1 \vee \neg y_2 \vee \neg x_2)_{2,3} \wedge (y_1 \vee x_1 \vee \neg y_2 \vee \neg x_2)_{2,4} \wedge (\neg y_1 \vee y_2 \vee x_2)_5 \wedge (\neg y_1 \vee y_2 \vee \neg x_2)_6 \wedge (y_1 \vee \neg x_2)_{8,3} \wedge (\neg y_1 \vee x_1 \vee \neg x_2)_{8,4})$$

Variable elimination: an example

Given the QBF:

$$\exists x_1 \forall y \exists x_3 (x_1 \vee y \vee x_3) \wedge (y \vee \neg x_3) \wedge (\neg x_1 \vee y \vee x_3)$$

Variable elimination: an example

Given the QBF:

$$\exists x_1 \forall y \exists x_3 (x_1 \vee y \vee x_3) \wedge (y \vee \neg x_3) \wedge (\neg x_1 \vee y \vee x_3)$$

we can eliminate x_3 by performing the resolutions

$$\frac{(x_1 \vee y \vee x_3) \quad (y \vee \neg x_3)}{x_1 \vee y} \qquad \frac{(\neg x_1 \vee y \vee x_3) \quad (y \vee \neg x_3)}{\neg x_1 \vee y}$$

Variable elimination: an example

Given the QBF:

$$\exists x_1 \forall y \exists x_3 (x_1 \vee y \vee x_3) \wedge (y \vee \neg x_3) \wedge (\neg x_1 \vee y \vee x_3)$$

we can eliminate x_3 by performing the resolutions

$$\frac{(x_1 \vee y \vee x_3) \quad (y \vee \neg x_3)}{x_1 \vee y} \qquad \frac{(\neg x_1 \vee y \vee x_3) \quad (y \vee \neg x_3)}{\neg x_1 \vee y}$$

which yields

$$\exists x_1 \forall y (x_1 \vee y) \wedge (\neg x_1 \vee y)$$

Variable elimination: an example

Given the QBF:

$$\exists x_1 \forall y \exists x_3 (x_1 \vee y \vee x_3) \wedge (y \vee \neg x_3) \wedge (\neg x_1 \vee y \vee x_3)$$

we can eliminate x_3 by performing the resolutions

$$\frac{(x_1 \vee y \vee x_3) \quad (y \vee \neg x_3)}{x_1 \vee y} \qquad \frac{(\neg x_1 \vee y \vee x_3) \quad (y \vee \neg x_3)}{\neg x_1 \vee y}$$

which yields

$$\exists x_1 \forall y (x_1 \vee y) \wedge (\neg x_1 \vee y)$$

Removing y we obtain

$$\exists x_1 (x_1) \wedge (\neg x_1)$$

Variable elimination: an example

Given the QBF:

$$\exists x_1 \forall y \exists x_3 (x_1 \vee y \vee x_3) \wedge (y \vee \neg x_3) \wedge (\neg x_1 \vee y \vee x_3)$$

we can eliminate x_3 by performing the resolutions

$$\frac{(x_1 \vee y \vee x_3) \quad (y \vee \neg x_3)}{x_1 \vee y} \qquad \frac{(\neg x_1 \vee y \vee x_3) \quad (y \vee \neg x_3)}{\neg x_1 \vee y}$$

which yields

$$\exists x_1 \forall y (x_1 \vee y) \wedge (\neg x_1 \vee y)$$

Removing y we obtain

$$\exists x_1 (x_1) \wedge (\neg x_1)$$

which is trivially false.

Search

AND-OR branching

Initially, the current QBF is φ

- If $\varphi = \exists x \psi$ then create an OR-node, whose children are the checks $\varphi_{|x=1}$ is true **or** $\varphi_{|x=0}$ is true.
- If $\varphi = \forall y \psi$ then create an AND-node, whose children are the checks of whether $\varphi_{|y=1}$ **and** $\varphi_{|y=0}$ are true.

Search

AND-OR branching

Initially, the current QBF is φ

- If $\varphi = \exists x \psi$ then create an OR-node, whose children are the checks $\varphi_{|x=1}$ is true **or** $\varphi_{|x=0}$ is true.
- If $\varphi = \forall y \psi$ then create an AND-node, whose children are the checks of whether $\varphi_{|y=1}$ **and** $\varphi_{|y=0}$ are true.

The leaves are of two kinds:

- conflicts, if current QBF is trivially false;
- solutions, if current QBF is trivially true.

Backtracking

Backtracking amounts to:

- from a conflict, reaching back to the deepest open OR-node: if there is no such node, then φ is false;
- from a solution, reaching back to the deepest open AND-node: if there is no such node, then φ is true.

Search: an example

$$\exists x_1 \forall y \exists x_2 \exists x_3 \{\{\overline{x}_1, \overline{y}, x_2\}, \{x_1, \overline{y}, \overline{x}_3\}, \{\overline{y}, \overline{x}_2\}, \{y, x_2, \overline{x}_3\}, \{x_2, x_3\}\}$$

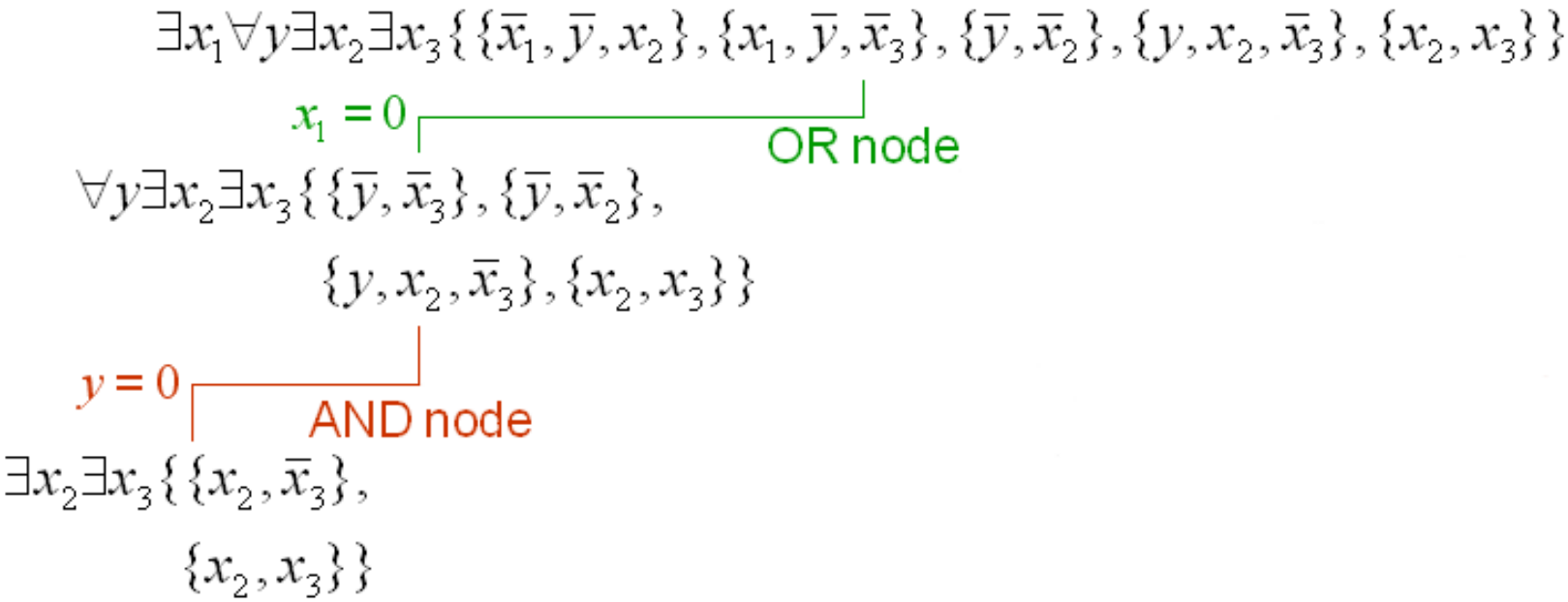
Search: an example

$$\exists x_1 \forall y \exists x_2 \exists x_3 \{\{\bar{x}_1, \bar{y}, x_2\}, \{x_1, \bar{y}, \bar{x}_3\}, \{\bar{y}, \bar{x}_2\}, \{y, x_2, \bar{x}_3\}, \{x_2, x_3\}\}$$

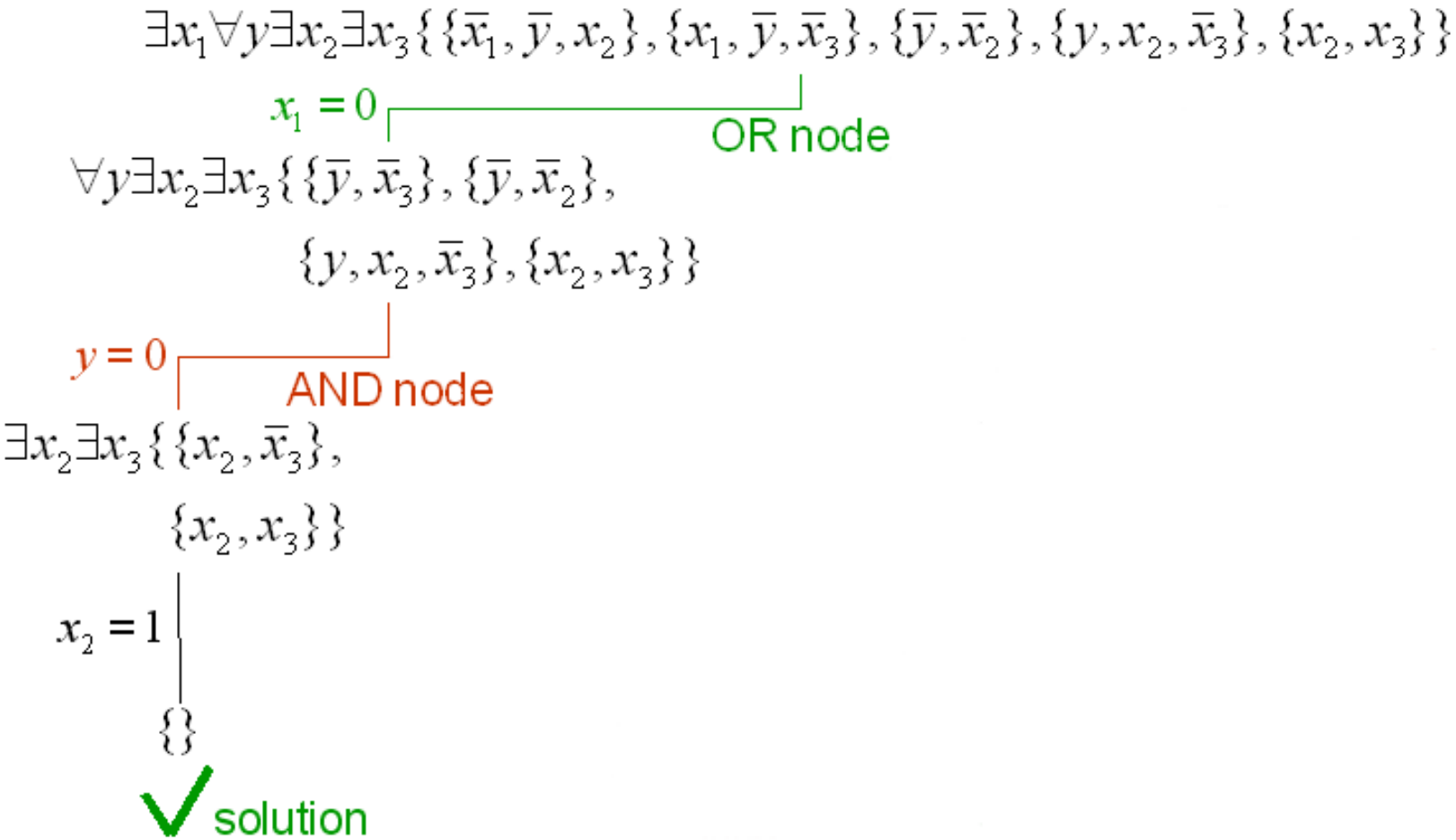
$x_1 = 0$ — OR node

$$\forall y \exists x_2 \exists x_3 \{\{\bar{y}, \bar{x}_3\}, \{\bar{y}, \bar{x}_2\}, \{y, x_2, \bar{x}_3\}, \{x_2, x_3\}\}$$

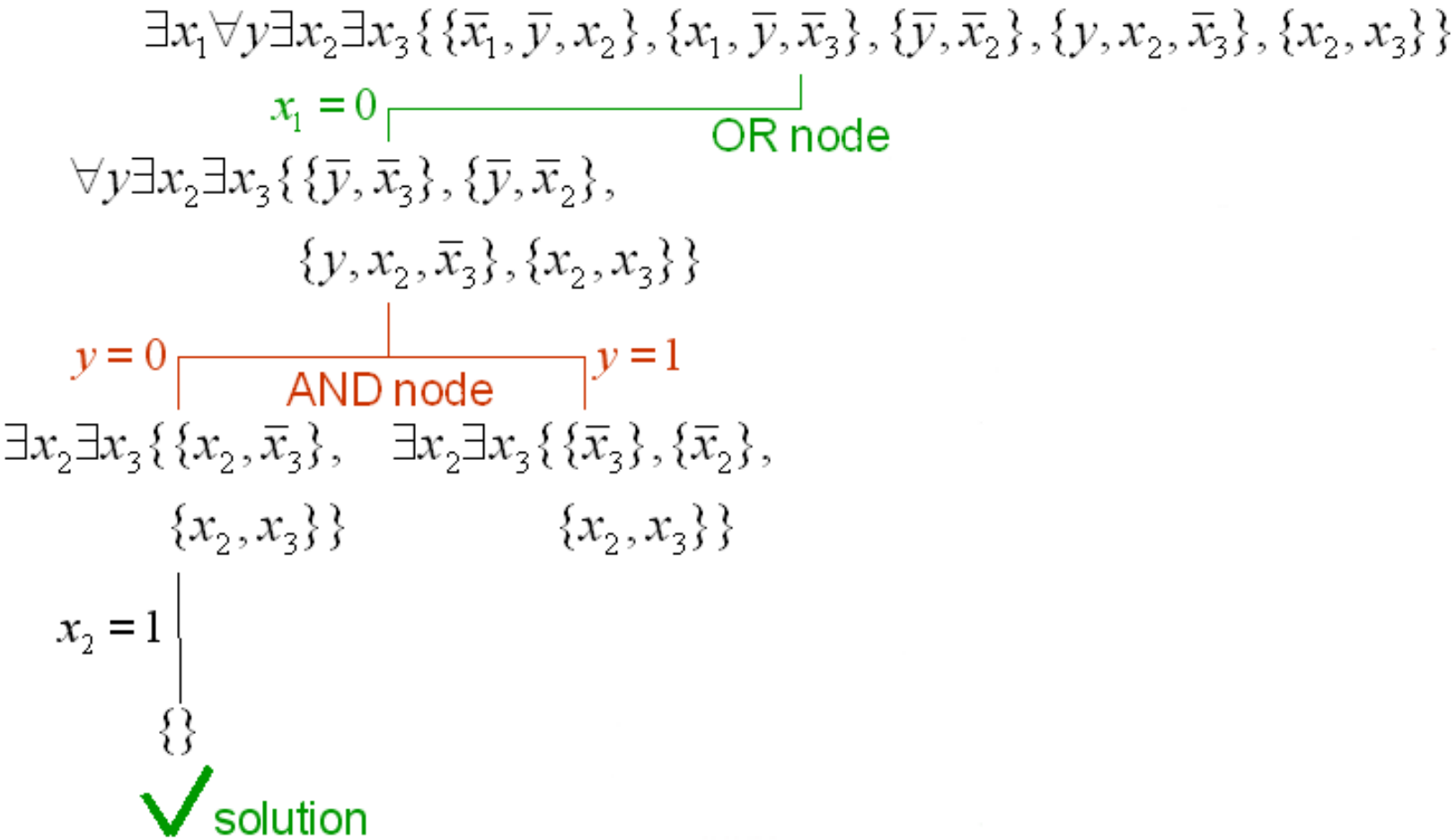
Search: an example



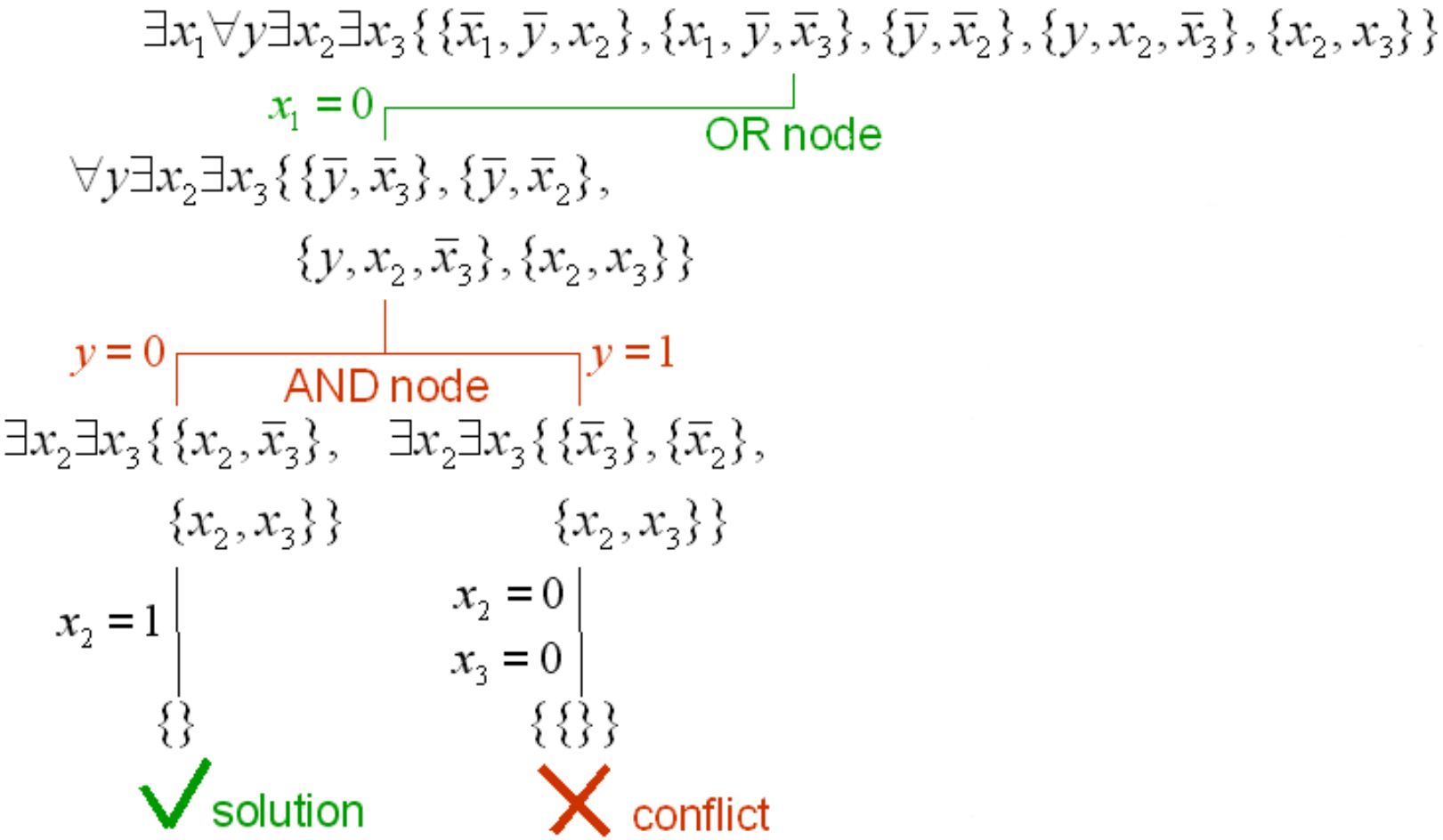
Search: an example



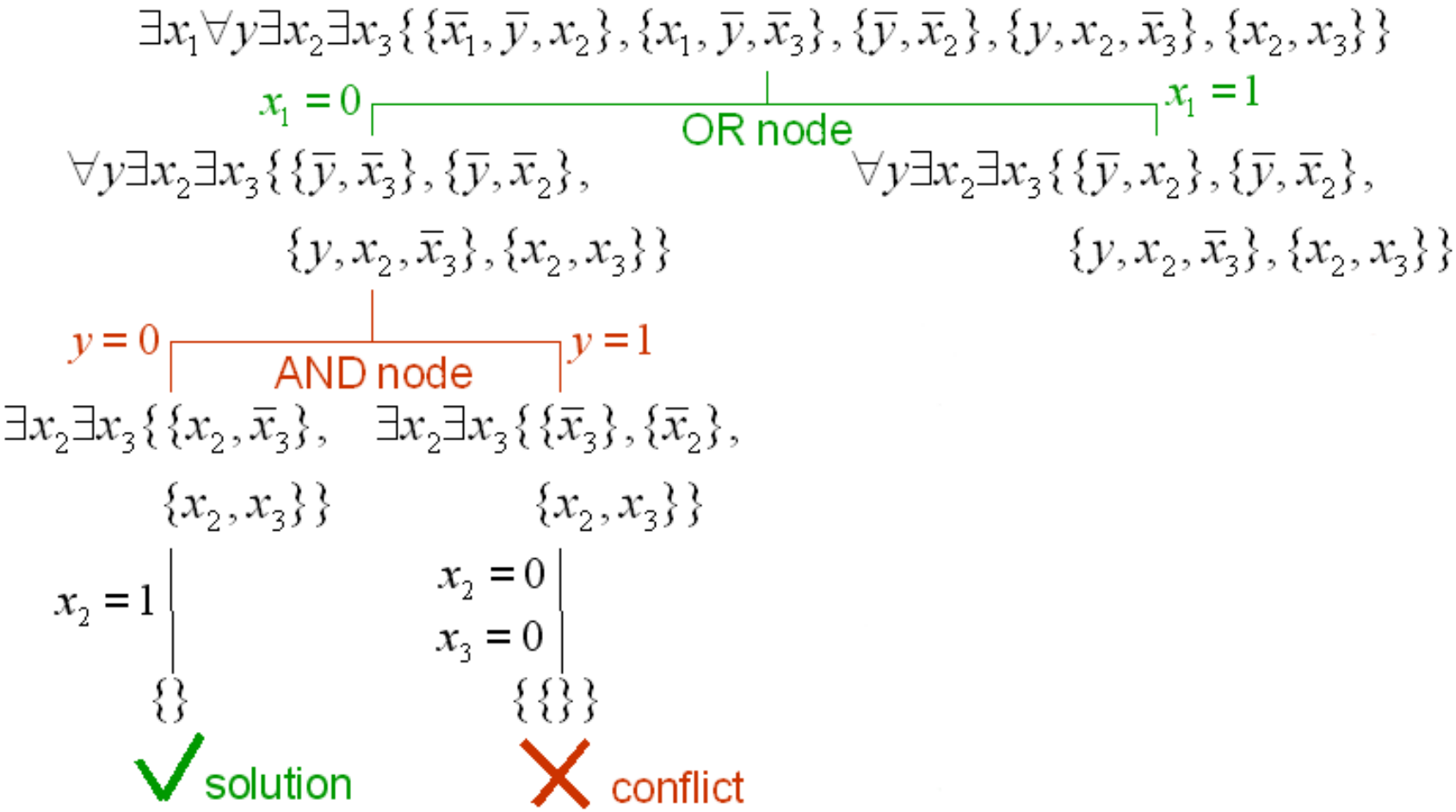
Search: an example



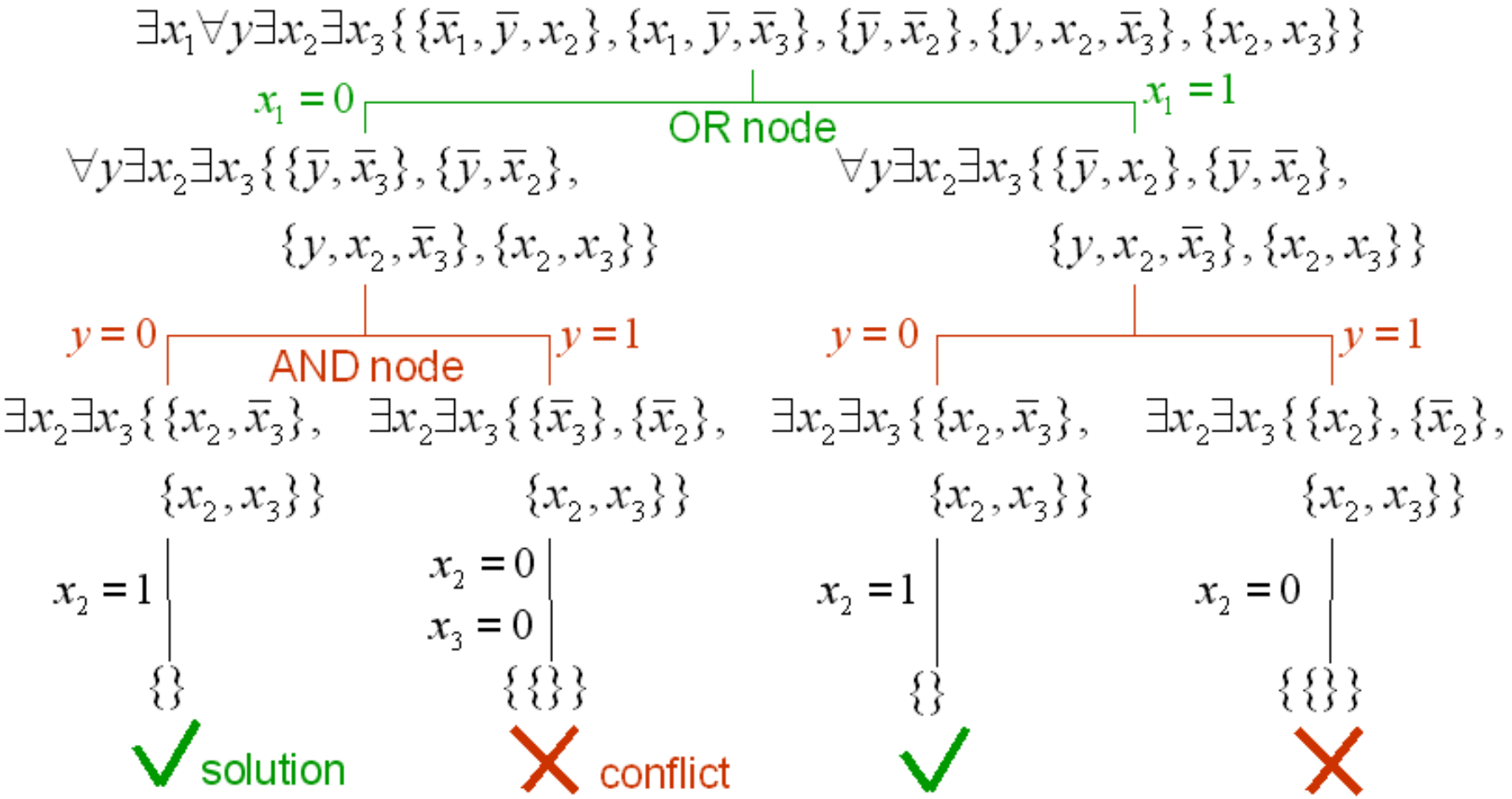
Search: an example



Search: an example



Search: an example



Variable elimination, search and treewidth

Variable elimination

- Can generate an **exponential number** of resolvents!
- Given a QBF φ the number of resolvents is $O(2^{tw_p(\varphi)})$.

Variable elimination, search and treewidth

Variable elimination

- Can generate an **exponential number** of resolvents!
- Given a QBF φ the number of resolvents is $O(2^{tw_p(\varphi)})$.

Backtracking search

- Can explore an **exponentially large** tree! (without storing it)
- Given a QBF with n variables, the time to explore the search tree is $O(2^n)$.

Variable elimination, search and treewidth

Variable elimination

- Can generate an **exponential number** of resolvents!
- Given a QBF φ the number of resolvents is $O(2^{tw_p(\varphi)})$.

Backtracking search

- Can explore an **exponentially large** tree! (without storing it)
- Given a QBF with n variables, the time to explore the search tree is $O(2^n)$.

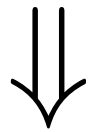
A structural approach

A sequence $\varphi_1, \varphi_2, \ldots$ where $\varphi_1 = \varphi$, and φ_{i+1} is obtained from φ_i by

- variable elimination (one step) whenever $tw_p(\varphi_i)$ is "small", and
- backtracking search (one step) otherwise.

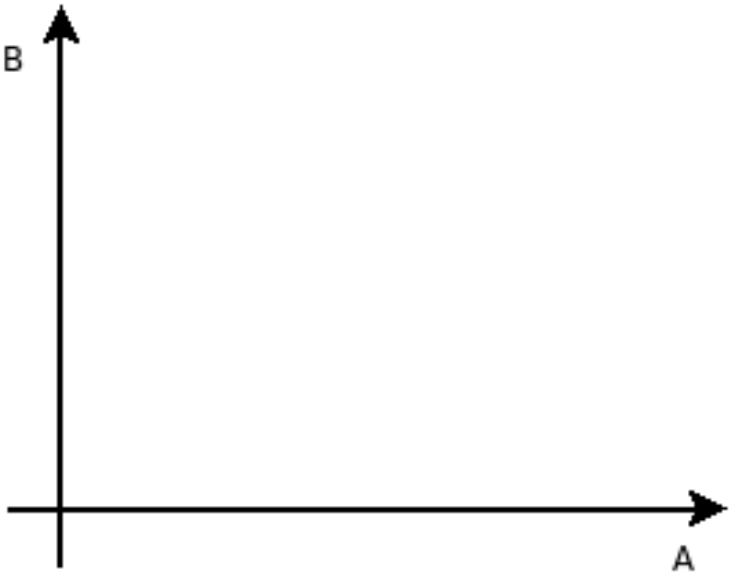
But there is a catch...

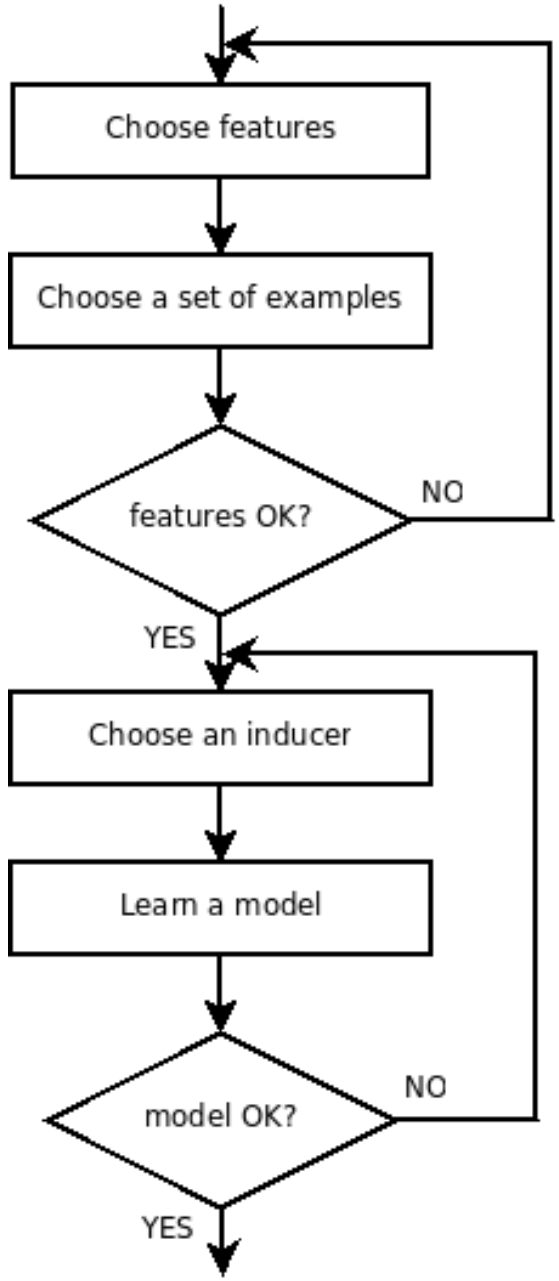
- Computing treewidth is an **NP-hard problem**!
- **Non trivial lower/upper bounds** of treewidth can be computed in **polynomial time** however
  - algorithms are **way too slow** for graphs of the size we are interested in: $\approx$**1K vertices or more**;
  - they **do not guarantee** the quality of the approximation.



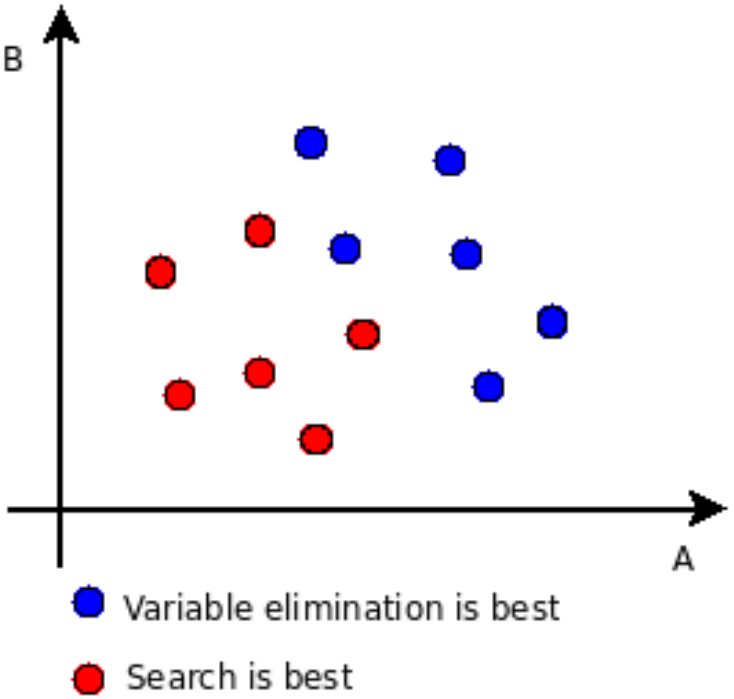
We need a **computationally efficient** yet **effective** way of choosing between variable elimination and search.

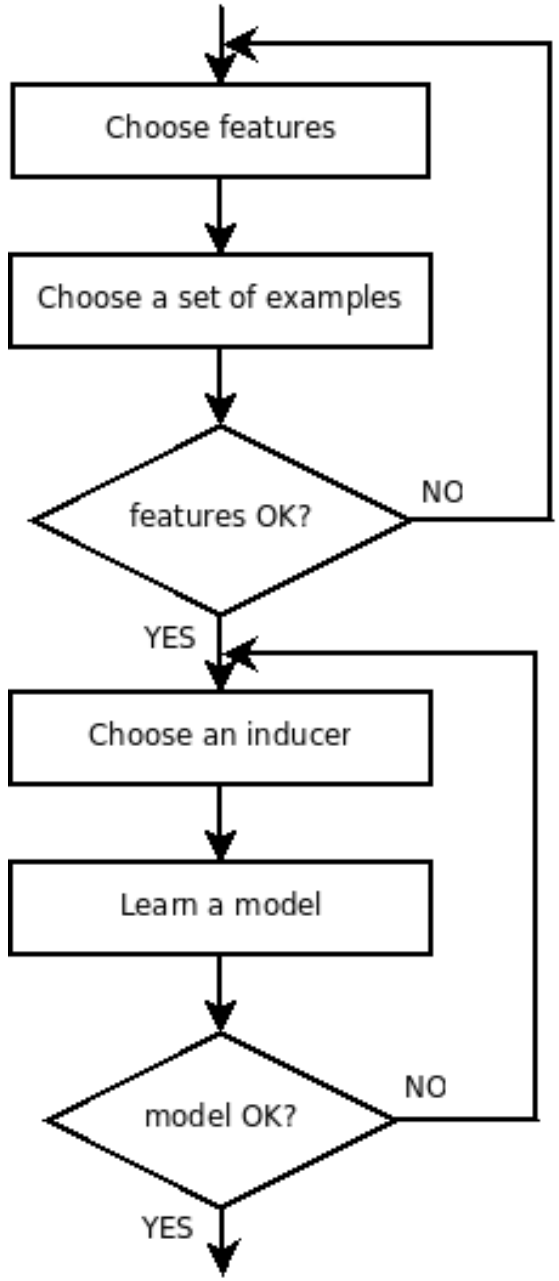
Our solution: learn to choose!



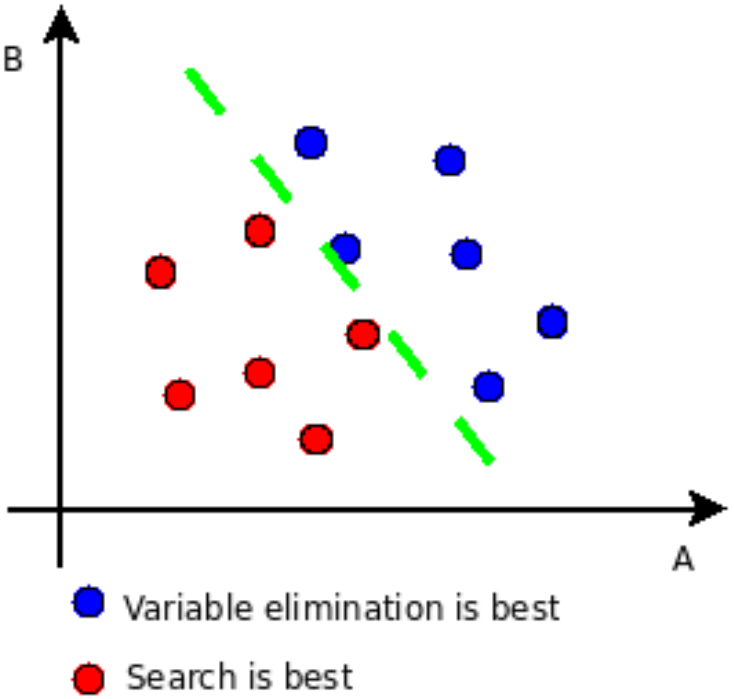


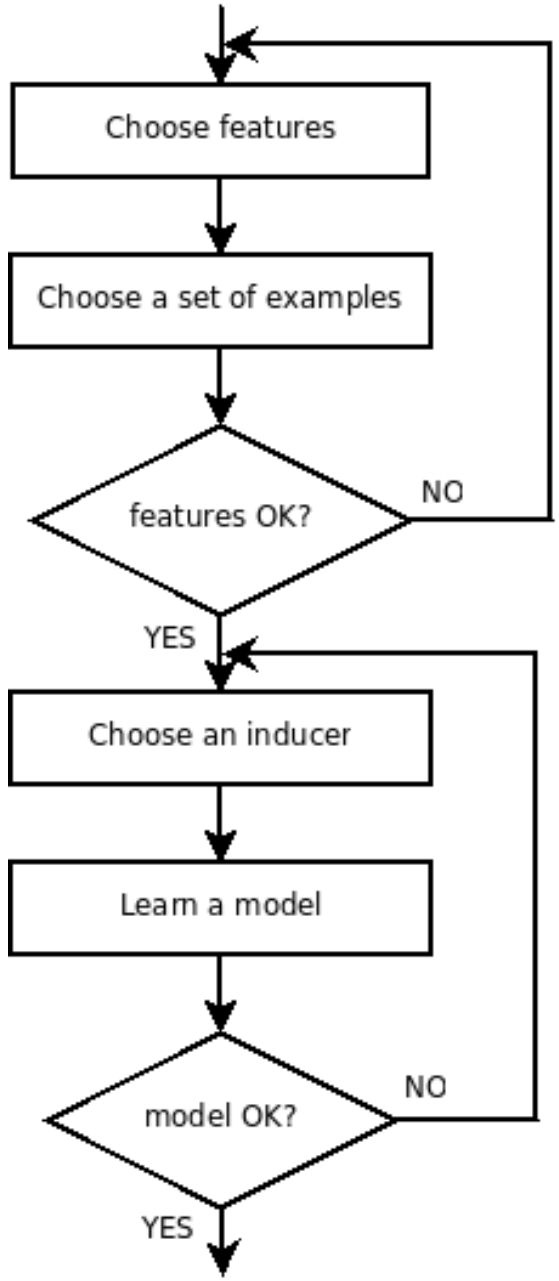
Our solution: learn to choose!



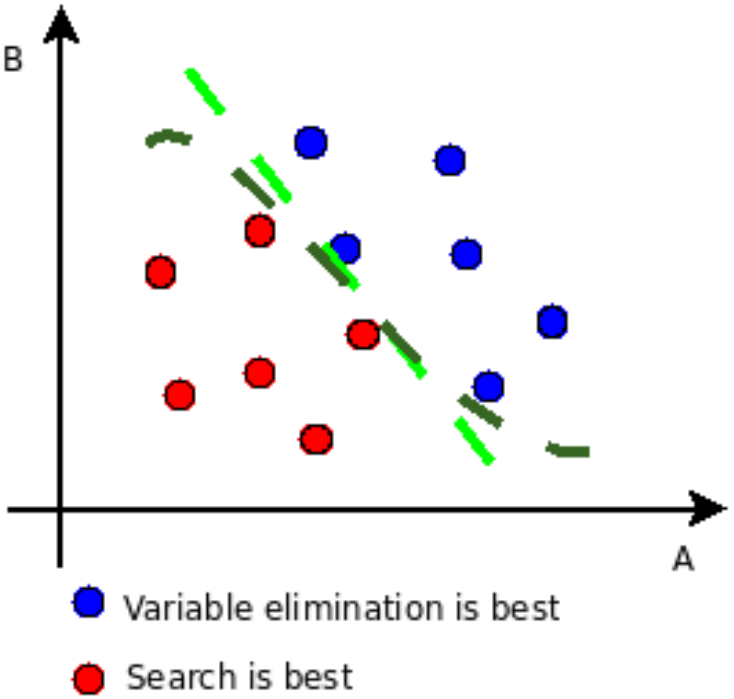


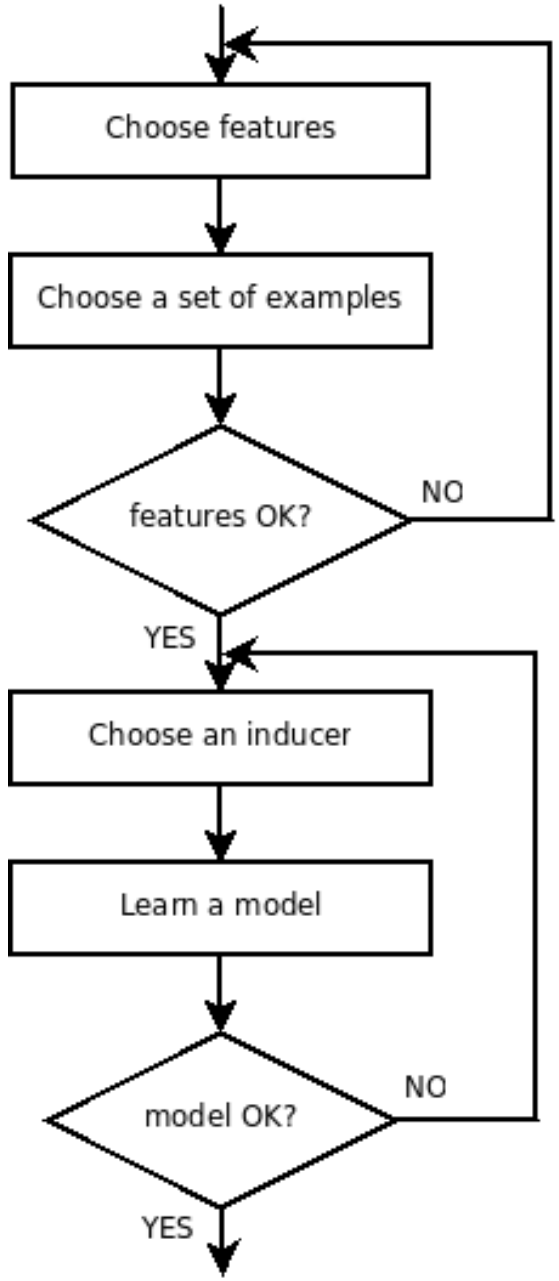
Our solution: learn to choose!



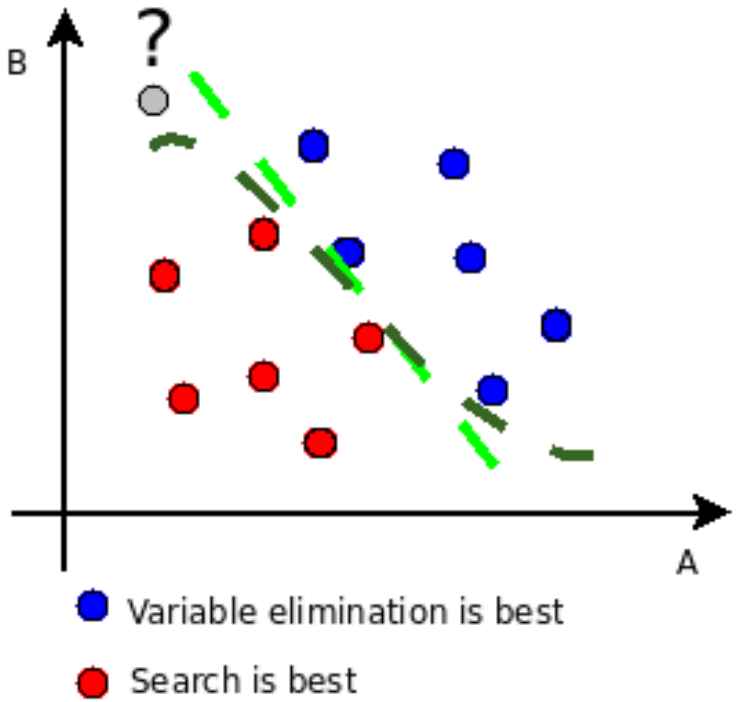


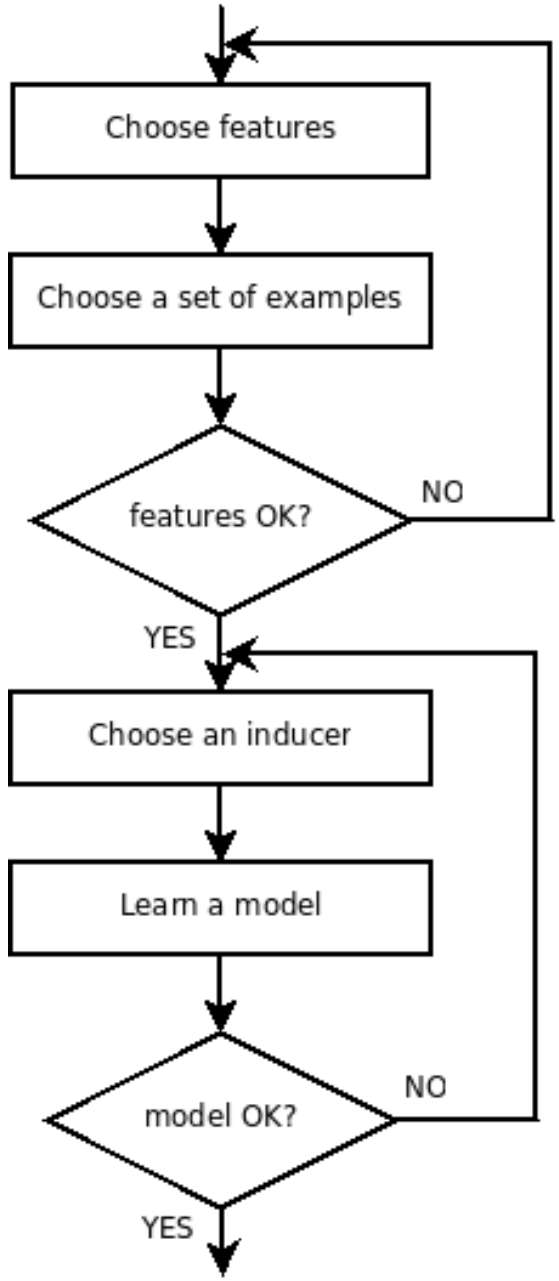
Our solution: learn to choose!





Our solution: learn to choose!





Choosing features

If x is any variable that qualifies for elimination,

- n_x # of clauses where x occurs, and
- l_x sum of the # of literals in clauses where x occurs.

We consider:

- The **number of occurrences** of x, denoted as $occs(x)$ and computed as $occs(x) = n_x + n_{\neg x}$.
- The **diversity** of x, denoted as $div(x)$ and computed as $div(x) = n_x \cdot n_{\neg x}$.
- The **companion literals** of x, denoted as $lits(x)$ and computed as $lits(x) = l_x \cdot l_{\neg x}$.

Are the values of such features discriminative enough?

Are features ok?

Two **basic versions** of QuReS:

- QuReS-bj, implements search plus intelligent backtracking.
- QuReS-ve, implements variable elimination.

A **relevant** set of examples (QBFEVAL'08-unique):

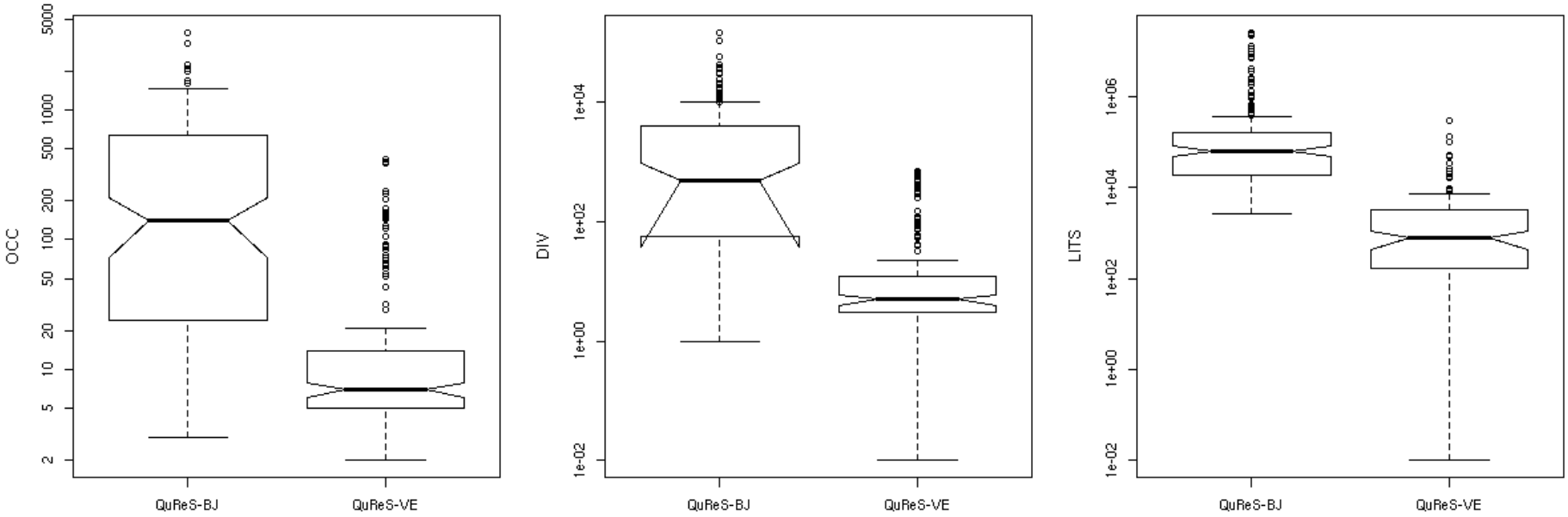
- Subset of 2008 QBF solvers competition dataset.
- Formulas solved either by QuReS or by QuReS-ve (not both!)

A plan for **validating features**:

- Run QuReS-ve on the QBFEVAL'08-unique subset.
- Collect the values of $occs(x)$, $div(x)$ and $lits(x)$ for each variable x eliminated by QuReS-ve.
- Compare the **distributions** between problems solved by QuReS-ve and those solved by QuReS-bj.

Discriminating QuReS-BJ and QuReS-VE

occs(*x*) *div*(*x*) *lits*(*x*)



The center (and also the spread for *occs*(*x*) and *div*(*x*)) of the distributions differs in a statistically significant way between QuReS-VE and QuReS-BJ.

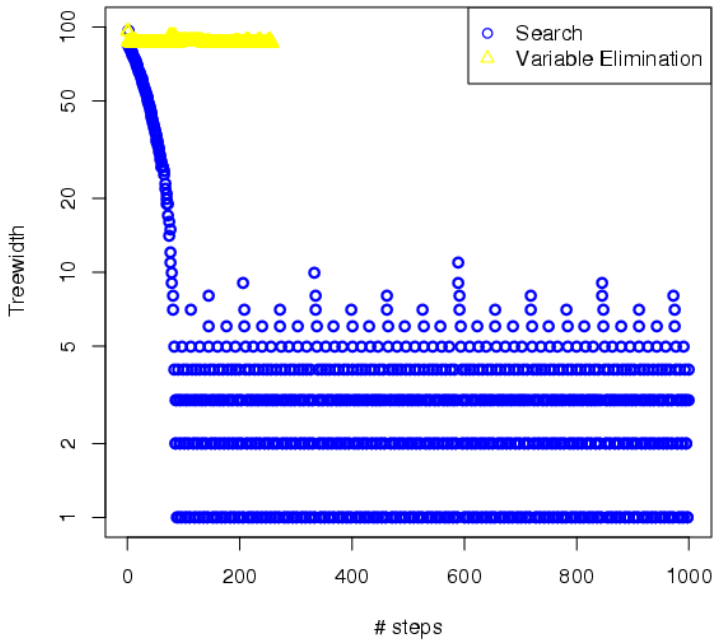
QuReS versions

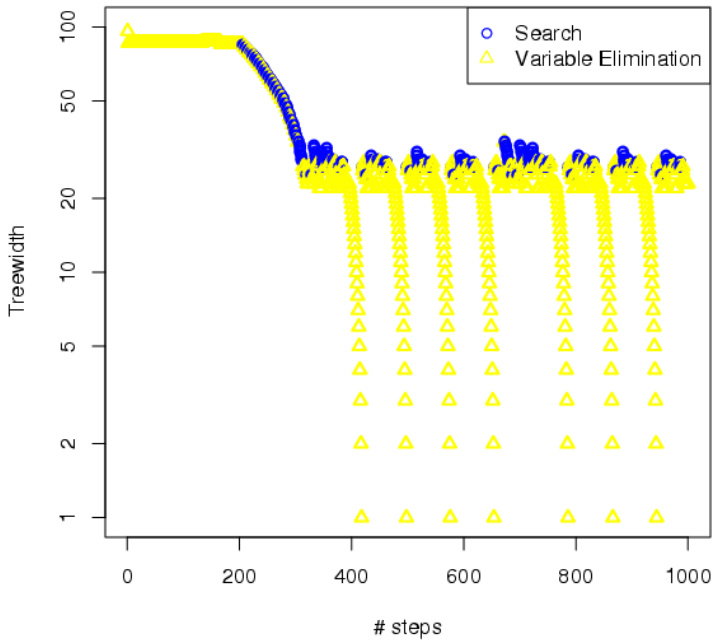
- QuReS-c4.5 featuring a selection strategy learned with C4.5 decision trees (WEKA's implementation J48)
- QuReS-svm featuring a selection strategy learned with SVMs (libSVM implementation)
- "Control" versions:
  - QuReS-hm featuring a "hand-made" strategy proposed in previous literature.
  - QuReS-rnd featuring a random selection strategy
- The basic algorithms QuReS-ve and QuReS-bj.

QuReS C++ source code is available from

www.mind-lab.it/projects/

QuRes in action: smallest "hard" instance





- QuRes-ve (yellow-left) decreases tw_p (96 $\rightarrow$ 86), but it exhausts memory
- QuRes-bj (blue-left) gets "trapped" and times out in $\sim 9 \times 10^6$ steps.
- QuRes changes its behavior according to the current value of tw_p:
  - First 206 steps are variable elimination only (yellow).
  - Search (blue) yields structurally simpler subproblems.

  QuRes takes less than 4×10^5 steps (25% search).

QuReS vs. QuReS

Solver	Solved		True		False	
	#	Time	#	Time	#	Time
QuReS-c4.5	1282	53812	569	27547	713	26265
QuReS-svm	1249	68423	548	30592	702	37831
QuReS-hm	883	64062	382	32989	501	31073
QuReS-rnd	670	24640	260	8428	410	16212
QuReS-bj	614	31543	208	13099	406	18444
QuReS-ve	528	12834	228	6384	300	6450

600s of CPU time, 3GB of main memory

QuRes vs. state-of-the-art solvers

Solver	Solved		True		False	
	#	Time	#	Time	#	Time
AQME	2434	43987	977	19747	1457	24240
QuBE6.1	2144	32414	828	18248	1316	14166
sKizzo	1887	40864	631	17550	1256	23314
QuRes-c4.5	1282	53812	569	27547	713	26265
QuBE3.0	1077	16700	406	6536	671	10164
nenofex	985	22360	459	13853	526	8507
quantor	972	15718	485	10418	487	5300
ssolve	965	23059	450	9866	515	13193
yQuaffle	948	16708	389	9058	559	7650
2clsQ	780	21287	391	13234	389	8053
QMRes	704	13576	360	7722	344	5853

- AQME includes QuBE and sKizzo as engines.
- Both AQME and sKizzo are far more sophisticated than QuRes (tens of thousands vs. hundreds LOCs).

Current limitations of QuReS

- The algorithm **selection strategy** is somehow **suboptimal**:
  - QuReS does not solve a superset of the instances solved by either StruQS[bj] and StruQS[ve].
  - the same is true of StruQS[bt,ve] vs. StruQS[bt] and StruQS[ve].
- Use of **forward subsumption** (FS) to reduce the number of resolvents:
  - without FS, the number of formulas solved by StruQS[bj,ve] **drops by 15%**.
  - FS accounts for **8%** of StruQS[bj,ve] time on **solved formulas**, but for **20%** of StruQS[bj,ve] time on **unsolved formulas**.
  - A time limit of 1200s allows StruQS[bj,ve] to solve about **10% additional problems**.

Future work

- Improve on the algorithm selection strategy
  - Look at corner cases for insight
  - Investigate further on features
- Improve on the algorithmics
  - Current data structure is “biased” towards the search part
  - Forward subsumption can probably be made more efficient

Take home message

Learning can aid the integration of algorithms when the “frontier” is not practically computable, e.g., treewidth of QBFs.

Some advertising...

- www.qbflib.org contains the largest publicly available collection of QBFs in standard format, and more (links to developers, relevant QBF papers, ...)
- www.qbfeval.org is the home of QBFEVAL, an international event seeking to compare state-of-the-art QBF solvers.
- QBFEVAL'10 (seventh event in the series) is coming soon. Results will be presented at the annual SAT conference (see www.satisfiability.org)

Thank you for your attention!