Reasoning Engines for Rigorous System Engineering

Block 3: Quantified Boolean Formulas and DepQBF
1. Inside Search-based QBF Solvers

Uwe Egly    Florian Lonsing

Knowledge-Based Systems Group
Institute of Information Systems
Vienna University of Technology

This work is supported by the Austrian Science Fund (FWF) under grant S11409-N23.
Success Story of SAT Solving:

- Backtracking search, clause learning as systematic application of resolution: CDCL
- Broad field of research: solver technology, theory, applications.
- Solver development driven by applications and vice versa.

Quantified Boolean Formulae (QBF):

- Explicit quantifiers ($\forall, \exists$) over propositional variables.
- NP-completeness of SAT vs. PSPACE-completeness of QBF.
- Potentially more succinct QBF encodings of PSPACE-complete problems.

QBF Solving:

- QCDCL: inspired by CDCL for SAT.
- Alternative: variable elimination.
- After peak in 2006/2007, renewed interest in QBF.
Success Story of SAT Solving:
- Backtracking search, clause learning as systematic application of resolution: CDCL
- Broad field of research: solver technology, theory, applications.
- Solver development driven by applications and vice versa.

Quantified Boolean Formulae (QBF):
- Explicit quantifiers ($\forall, \exists$) over propositional variables.
- NP-completeness of SAT vs. PSPACE-completeness of QBF.
- Potentially more succinct QBF encodings of PSPACE-complete problems.

QBF Solving:
- QCDCL: inspired by CDCL for SAT.
- Alternative: variable elimination.
- After peak in 2006/2007, renewed interest in QBF.
Success Story of SAT Solving:
- Backtracking search, clause learning as systematic application of resolution: CDCL
- Broad field of research: solver technology, theory, applications.
- Solver development driven by applications and vice versa.

Quantified Boolean Formulae (QBF):
- Explicit quantifiers (∀, ∃) over propositional variables.
- NP-completeness of SAT vs. PSPACE-completeness of QBF.
- Potentially more succinct QBF encodings of PSPACE-complete problems.

QBF Solving:
- QCDCL: inspired by CDCL for SAT.
- Alternative: variable elimination.
- After peak in 2006/2007, renewed interest in QBF.
6 new formula sets, 150 formulas each.

- At least one solver is good for a set (but it is not always the same).

- [Link](http://www.kr.tuwien.ac.at/events/qbfgallery2013/)
Our Focus:
- Search-based QBF solving as a major approach (next to variable elimination).
- Bottom-up approach: from basic building blocks to general view.
- The role of Q-resolution in QBF solvers.

Lessons to be Learned:
- (Q)CDCL is not just a combination of backtracking search and clause learning.
- Implementation: QBF solvers are more complex than SAT solvers.
- Pitfalls when porting SAT solver technology to QBF.

Example: DepQBF
- Search-based QBF solver, under active development since 2010.
- Open source: http://lonsing.github.io/depqbf/
- Friday: API demo, examples of recent improvements (incremental solving).
Overview (2/2)

Our Focus:
- Search-based QBF solving as a major approach (next to variable elimination).
- Bottom-up approach: from basic building blocks to general view.
- The role of Q-resolution in QBF solvers.

Lessons to be Learned:
- (Q)CDCL is not just a combination of backtracking search and clause learning.
- Implementation: QBF solvers are more complex than SAT solvers.
- Pitfalls when porting SAT solver technology to QBF.

Example: DepQBF
- Search-based QBF solver, under active development since 2010.
- Open source: http://lonsing.github.io/depqbf/
- Friday: API demo, examples of recent improvements (incremental solving).
Our Focus:
- Search-based QBF solving as a major approach (next to variable elimination).
- Bottom-up approach: from basic building blocks to general view.
- The role of Q-resolution in QBF solvers.

Lessons to be Learned:
- (Q)CDCL is not just a combination of backtracking search and clause learning.
- Implementation: QBF solvers are more complex than SAT solvers.
- Pitfalls when porting SAT solver technology to QBF.

Example: DepQBF
- Search-based QBF solver, under active development since 2010.
- Open source: http://lonsing.github.io/depqbf/
- Friday: API demo, examples of recent improvements (incremental solving).
Syntax, Semantics, Notation
QBF Syntax

QBF in Prenex Conjunctive Normal Form:
- Given a Boolean formula $\phi(x_1, \ldots, x_m)$ in CNF.
- Quantifier prefix $Q_1 B_1 Q_2 B_2 \ldots Q_m B_m$.
- Quantifiers $Q_i \in \{\forall, \exists\}$.
- Quantifier block $B_i \subseteq \{x_1, \ldots, x_m\}$ containing variables.
- QBF in prenex CNF (PCNF): $Q_1 B_1 Q_2 B_2 \ldots Q_m B_m \phi(x_1, \ldots, x_m)$.
- $B_i \leq B_{i+1}$: quantifier blocks are linearly ordered (extended to variables, literals).

Example
- Given the CNF $\phi := (x_1 \lor \neg x_3) \land (x_1 \lor x_4) \land (\neg x_2 \lor \neg x_3) \land (\neg x_3 \lor x_4)$.
- Given the quantifier prefix $\forall x_1, x_2 \exists x_3, x_4$.
- Prenex CNF: $\forall x_1, x_2 \exists x_3, x_4. (x_1 \lor \neg x_3) \land (x_1 \lor x_4) \land (\neg x_2 \lor \neg x_3) \land (\neg x_3 \lor x_4)$. 
QBF Syntax

QBF in Prenex Conjunctive Normal Form:
- Given a Boolean formula $\phi(x_1, \ldots, x_m)$ in CNF.
- Quantifier prefix $Q_1 B_1 Q_2 B_2 \ldots Q_m B_m$.
- Quantifiers $Q_i \in \{\forall, \exists\}$.
- Quantifier block $B_i \subseteq \{x_1, \ldots, x_m\}$ containing variables.
- QBF in prenex CNF (PCNF): $Q_1 B_1 Q_2 B_2 \ldots Q_m B_m \phi(x_1, \ldots, x_m)$.
- $B_i \leq B_{i+1}$: quantifier blocks are linearly ordered (extended to variables, literals).

Example
- Given the CNF $\phi := (x_1 \lor \neg x_3) \land (x_1 \lor x_4) \land (\neg x_2 \lor \neg x_3) \land (\neg x_3 \lor x_4)$.
- Given the quantifier prefix $\forall x_1, x_2 \exists x_3, x_4$.
- Prenex CNF: $\forall x_1, x_2 \exists x_3, x_4. (x_1 \lor \neg x_3) \land (x_1 \lor x_4) \land (\neg x_2 \lor \neg x_3) \land (\neg x_3 \lor x_4)$.
Variable Assignments:

- Mapping $V \to \{\top, \bot\}$, variables $V = \{x_1, \ldots, x_m\}$, truth values $\top$ and $\bot$.
- Given a CNF $\phi(x_1, \ldots, x_i, \ldots, x_m)$, assigning $x_i$ to $v$, where $v \in \{\top, \bot\}$, produces the CNF $\phi(x_1, \ldots, x_m)[x_i/v]$.
- In $\phi(x_1, \ldots, x_m)[x_i/v]$, occurrences of $x_i$ are replaced by the value $v$.
- Standard simplifications by Boolean algebra:
  - $\phi' \lor \top \equiv \top$, $\phi' \lor \bot \equiv \phi'$, $\phi' \land \top \equiv \phi'$, $\phi' \land \bot \equiv \bot$.
- Write $\phi[x_i]$ for $\phi[x_i/\top]$ and $\phi[\neg x_i]$ for $\phi[x_i/\bot]$: literals denote assignments.

Example

- Given the CNF $\phi := (x_1 \lor \neg x_2) \land (x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3)$.
- $\phi[x_2/\top] = (x_1 \lor \neg \top) \land (\top \lor x_3) \land (\neg x_1 \lor \neg x_3)$.
- $\phi[x_2/\top] = (x_1) \land (\neg x_1 \lor \neg x_3)$.
Variable Assignments:

- Mapping $V \rightarrow \{\top, \bot\}$, variables $V = \{x_1, \ldots, x_m\}$, truth values $\top$ and $\bot$.
- Given a CNF $\phi(x_1, \ldots, x_i, \ldots, x_m)$, assigning $x_i$ to $v$, where $v \in \{\top, \bot\}$, produces the CNF $\phi(x_1, \ldots, x_m)[x_i/v]$.
- In $\phi(x_1, \ldots, x_m)[x_i/v]$, occurrences of $x_i$ are replaced by the value $v$.
- Standard simplifications by Boolean algebra:
  $\phi' \lor \top \equiv \top$, $\phi' \lor \bot \equiv \phi'$, $\phi' \land \top \equiv \phi'$, $\phi' \land \bot \equiv \bot$.
- Write $\phi[x_i]$ for $\phi[x_i/\top]$ and $\phi[\neg x_i]$ for $\phi[x_i/\bot]$: literals denote assignments.

Example

- Given the CNF $\phi := (x_1 \lor \neg x_2) \land (x_2 \lor x_3) \land (\neg x_1 \lor \neg x_3)$.
- $\phi[x_2/\top] = (x_1 \lor \neg \top) \land (\top \lor x_3) \land (\neg x_1 \lor \neg x_3)$.
- $\phi[x_2/\top] = (x_1) \land (\neg x_1 \lor \neg x_3)$. 
Recursive Definition:

- Recursively assign the variables in prefix order (from left to right).
- Base cases: the QBF $\top$ ($\bot$) is satisfiable (unsatisfiable).
- $\psi = \forall x \ldots \phi$ is satisfiable if $\psi[\neg x]$ and $\psi[x]$ are satisfiable.
- $\psi = \exists x \ldots \phi$ is satisfiable if $\psi[\neg x]$ or $\psi[x]$ is satisfiable.
- Prerequisite: every variable is quantified in the prefix (no free variables).
- Satisfiability-equivalence of two PCNFs $\psi$ and $\psi'$: $\psi \equiv \psi'$.

Example

The PCNF $\psi = \forall x \exists y. (x \lor \neg y) \land (\neg x \lor y)$ is satisfiable if

1. $\psi[x] = \exists y. (y)$ and
2. $\psi[\neg x] = \exists y. (\neg y)$ are satisfiable.

1. $\psi[x] = \exists y. (y)$ is satisfiable since $\psi[x, y] = \top$ is satisfiable.
2. $\psi[\neg x] = \exists y. (\neg y)$ is satisfiable since $\psi[\neg x, \neg y] = \top$ is satisfiable.
QBF Semantics (2/5)

Recursive Definition:

- Recursively assign the variables in prefix order (from left to right).
- Base cases: the QBF $\top$ ($\bot$) is satisfiable (unsatisfiable).
- $\psi = \forall x \ldots \phi$ is satisfiable if $\psi[\neg x]$ and $\psi[x]$ are satisfiable.
- $\psi = \exists x \ldots \phi$ is satisfiable if $\psi[\neg x]$ or $\psi[x]$ is satisfiable.
- Prerequisite: every variable is quantified in the prefix (no free variables).
- Satisfiability-equivalence of two PCNFs $\psi$ and $\psi'$: $\psi \equiv \psi'$.

Example

The PCNF $\psi = \forall x \exists y. (x \lor \neg y) \land (\neg x \lor y)$ is satisfiable if

1. $\psi[x] = \exists y.(y)$ and
2. $\psi[\neg x] = \exists y.(\neg y)$ are satisfiable.

1. $\psi[x] = \exists y.(y)$ is satisfiable since $\psi[x, y] = \top$ is satisfiable.
2. $\psi[\neg x] = \exists y.(\neg y)$ is satisfiable since $\psi[\neg x, \neg y] = \top$ is satisfiable.
Recursive Definition:

- Recursively assign the variables in prefix order (from left to right).
- Base cases: the QBF $\top$ ($\bot$) is satisfiable (unsatisfiable).
- $\psi = \forall x \ldots \phi$ is satisfiable if $\psi[\neg x]$ and $\psi[x]$ are satisfiable.
- $\psi = \exists x \ldots \phi$ is satisfiable if $\psi[\neg x]$ or $\psi[x]$ is satisfiable.
- Prerequisite: every variable is quantified in the prefix (no free variables).
- Satisfiability-equivalence of two PCNFs $\psi$ and $\psi'$: $\psi \equiv \psi'$.

Example

The PCNF $\psi = \forall x \exists y. (x \lor \neg y) \land (\neg x \lor y)$ is satisfiable if

1. $\psi[x] = \exists y.(y)$ and
2. $\psi[\neg x] = \exists y.(\neg y)$ are satisfiable.

(1) $\psi[x] = \exists y.(y)$ is satisfiable since $\psi[x, y] = \top$ is satisfiable.
(2) $\psi[\neg x] = \exists y.(\neg y)$ is satisfiable since $\psi[\neg x, \neg y] = \top$ is satisfiable.
**Recursive Definition:**

- Recursively assign the variables in prefix order (from left to right).
- Base cases: the QBF $\top$ ($\bot$) is satisfiable (unsatisfiable).
- $\psi = \forall x \ldots \phi$ is satisfiable if $\psi[\neg x]$ and $\psi[x]$ are satisfiable.
- $\psi = \exists x \ldots \phi$ is satisfiable if $\psi[\neg x]$ or $\psi[x]$ is satisfiable.
- Prerequisite: every variable is quantified in the prefix (no free variables).
- Satisfiability-equivalence of two PCNFs $\psi$ and $\psi'$: $\psi \equiv \psi'$.

**Example**

The PCNF $\psi = \forall x \exists y.(x \lor \neg y) \land (\neg x \lor y)$ is satisfiable if

1. $\psi[x] = \exists y.(y)$ and
2. $\psi[\neg x] = \exists y.(\neg y)$ are satisfiable.

(1) $\psi[x] = \exists y.(y)$ is satisfiable since $\psi[x, y] = \top$ is satisfiable.
(2) $\psi[\neg x] = \exists y.(\neg y)$ is satisfiable since $\psi[\neg x, \neg y] = \top$ is satisfiable.
Example

The PCNF $\psi = \exists y \forall x. (x \lor \neg y) \land (\neg x \lor y)$ is unsatisfiable because neither

1. $\psi[y] = \forall x. (x)$ nor
2. $\psi[\neg y] = \forall x. (\neg x)$ is satisfiable.

(1) $\psi[y] = \forall x. (x)$ is unsatisfiable since $\psi[y, \neg x]$ is unsatisfiable.
(2) $\psi[\neg y] = \forall x. (\neg x)$ is unsatisfiable since $\psi[\neg y, x]$ is unsatisfiable.
Example

The PCNF $\psi = \exists y \forall x. (x \lor \neg y) \land (\neg x \lor y)$ is unsatisfiable because neither

1. $\psi[y] = \forall x.(x)$ nor
2. $\psi[\neg y] = \forall x.(\neg x)$ is satisfiable.

(1) $\psi[y] = \forall x.(x)$ is unsatisfiable since $\psi[y, \neg x]$ is unsatisfiable.
(2) $\psi[\neg y] = \forall x.(\neg x)$ is unsatisfiable since $\psi[\neg y, x]$ is unsatisfiable.
Example

The PCNF $\psi = \exists y \forall x. (x \lor \neg y) \land (\neg x \lor y)$ is unsatisfiable because neither

1. $\psi[y] = \forall x.(x)$ nor
2. $\psi[\neg y] = \forall x.(\neg x)$ is satisfiable.

1. $\psi[y] = \forall x.(x)$ is unsatisfiable since $\psi[y, \neg x]$ is unsatisfiable.
2. $\psi[\neg y] = \forall x.(\neg x)$ is unsatisfiable since $\psi[\neg y, x]$ is unsatisfiable.
Assignment Trees of a PCNF $\psi$:

- Dedicated root node $r$.
- Each path from root $r$ to a leaf represents a variable assignment $A$.
- The assignment sequence along each path follows the prefix order.
- Leaf is labelled with $\top$ if the PCNF $\psi[A]$ is satisfiable.
- Leaf is labelled with $\bot$ if the PCNF $\psi[A]$ is unsatisfiable.

Example

- Satisfiable PCNF $\psi = \forall x \exists y. (x \lor \neg y) \land (\neg x \lor y)$.
  - The node “$r$” represents $\psi$.
  - The node “$\neg x$” represents $\psi[\neg x] = \exists y. (\neg y)$.
  - Leftmost path: $\psi[\neg x, \neg y]$ is satisfiable.
  - Rightmost path: $\psi[x, y]$ is satisfiable.
Assignment Trees of a PCNF $\psi$:

- Dedicated root node $r$.
- Each path from root $r$ to a leaf represents a variable assignment $A$.
- The assignment sequence along each path follows the prefix order.
- Leaf is labelled with $\top$ if the PCNF $\psi[A]$ is satisfiable.
- Leaf is labelled with $\bot$ if the PCNF $\psi[A]$ is unsatisfiable.

Example

- Satisfiable PCNF $\psi = \forall x \exists y. (x \lor \neg y) \land (\neg x \lor y)$.
- The node “$r$” represents $\psi$.
- The node “$\neg x$” represents $\psi[\neg x] = \exists y. (\neg y)$.
- Leftmost path: $\psi[\neg x, \neg y]$ is satisfiable.
- Rightmost path: $\psi[x, y]$ is satisfiable.
Example (continued)

- Satisfiable PCNF \( \psi = \forall x \exists y. (x \lor \neg y) \land (\neg x \lor y). \)
- The node “\( r \)” represents \( \psi \).
- The node “\( \neg x \)” represents \( \psi[^{\neg x}] = \exists y. (\neg y). \)
- Leftmost path: \( \psi[^{\neg x}, \neg y] \) is satisfiable.
- Rightmost path: \( \psi[^{x}, y] \) is satisfiable.

- Assignment trees visualize the structure of recursive semantical evaluation.
- \( \forall x. \psi \): both recursive subcases \( \psi[^{\neg x}] \) and \( \psi[^{x}] \) (children) must be satisfiable.
- \( \exists x. \psi \): one recursive subcase \( \psi[^{\neg x}] \) or \( \psi[^{x}] \) (child) must be satisfiable.
Basic Backtracking Search
Recursive QBF Semantics and Backtracking Search

- Application of semantic rules: “splitting”, “decision making”, “branching”.
- Backtracking: flip value of decision (wrt. quantifier type and base case).
- Early termination if one subcase of $\exists (\forall)$ is satisfiable (unsatisfiable).

```c
bool bt_search (PCNF Qxψ, Assignment A)
    /* 1. Simplify under given assignment. */
    ψ' := simplify(Qxψ[A]);
    /* 2. Check base cases. */
    if (ψ' == ⊥)
        return false;
    if (ψ' == ⊤)
        return true;
    /* 3. Decision making, backtracking. */
    if (Q == ∃)
        return bt_search (ψ', A ∪ {¬x}) ||
        bt_search (ψ', A ∪ {x});
    if (Q == ∀)
        return bt_search (ψ', A ∪ {¬x}) &&
        bt_search (ψ', A ∪ {x});
```


Application of semantic rules: “splitting”, “decision making”, “branching”.

Backtracking: flip value of decision (wrt. quantifier type and base case).

Early termination if one subcase of $\exists (\forall)$ is satisfiable (unsatisfiable).

```c
bool bt_search (PCNF $Qx\psi$, Assignment A)
/* 1. Simplify under given assignment. */
   $\psi' :=$ simplify($Qx\psi[A]$);
/* 2. Check base cases. */
   if ($\psi' == \bot$)
      return false;
   if ($\psi' == \top$)
      return true;
/* 3. Decision making, backtracking. */
   if ($Q == \exists$)
      return bt_search ($\psi'$, A $\cup \{\neg x\}$) $\|$
      bt_search ($\psi'$, A $\cup \{x\}$);
   if ($Q == \forall$)
      return bt_search ($\psi'$, A $\cup \{\neg x\}$) $\&\&$
      bt_search ($\psi'$, A $\cup \{x\}$);
```


Example

\[ \psi := \forall y \exists x_1, x_2. (x_2) \land (\neg y \lor \neg x_2) \land (\neg y \lor x_1). \]

- Leftmost path: check \( \forall \)-subcase \( \psi[\neg y] \).
  - \( \exists \)-subcase \( \psi[\neg y, x_2] = \top \) already, no need to try \( \psi[\neg y, \neg x_2] \).
    - Backtrack and check \( \psi[y] \).
  - \( \exists \)-subcase \( \psi[y, \neg x_1] = \bot \), flip \( x_1 \) and check \( \psi[y, x_1] \).
  - Both \( \exists \)-subcases \( \psi[y, x_1, \neg x_2] = \bot \) and \( \psi[y, x_1, x_2] = \bot \), hence \( \exists \)-subcase \( \psi[y, x_1] \) unsat.
  - Since \( \exists \)-subcases \( \psi[y, \neg x_1], \psi[y, x_1] \) unsat., also \( \forall \)-subcase \( \psi[y] \) and \( \psi \) unsat.

**Observation:** the clause \((x_2)\) in \( \psi \) can only be satisfied by setting \( x_2 \) to true.

- Every subcase \( \psi[\ldots, \neg x_2, \ldots] \) is unsatisfiable: consider \( \psi[\ldots, x_2, \ldots] \) instead.
- \( \forall \)-subcase \( \psi[x_2, y] = \bot \), hence \( \psi \) unsatisfiable: smaller assignment tree!
Backtracking Search: Example

Example

ψ := ∀y∃x₁, x₂.(x₂) ∧ (¬y ∨ ¬x₂) ∧ (¬y ∨ x₁).

- Leftmost path: check ∀-subcase ψ[¬y].
  - ∃-subcase ψ[¬y, x₂] = ⊤ already, no need to try ψ[¬y, ¬x₂].
  - Backtrack and check ψ[y].
- ∃-subcase ψ[y, ¬x₁] = ⊥, flip x₁ and check ψ[y, x₁].
- Both ∃-subcases ψ[y, x₁, ¬x₂] = ⊥ and ψ[y, x₁, x₂] = ⊥, hence ∃-subcase ψ[y, x₁] unsat.
- Since ∃-subcases ψ[y, ¬x₁], ψ[y, x₁] unsat., also ∀-subcase ψ[y] and ψ unsat.

Observation: the clause (x₂) in ψ can only be satisfied by setting x₂ to true.

- Every subcase ψ[⋯, ¬x₂, ⋯] is unsatisfiable: consider ψ[⋯, x₂, ⋯] instead.
- ∀-subcase ψ[x₂, y] = ⊥, hence ψ unsatisfiable: smaller assignment tree!
Backtracking Search: Example

Example

\[ \psi := \forall y \exists x_1, x_2. (x_2) \land (\neg y \lor \neg x_2) \land (\neg y \lor x_1). \]

- **Leftmost path:** check \( \forall \)-subcase \( \psi[\neg y] \).
  
  \( \exists \)-subcase \( \psi[\neg y, x_2] = \top \) already, no need to try \( \psi[\neg y, \neg x_2] \). Backtrack and check \( \psi[y] \).

- **\( \exists \)-subcase** \( \psi[y, \neg x_1] = \bot \), flip \( x_1 \) and check \( \psi[y, x_1] \).

- Both \( \exists \)-subcases \( \psi[y, x_1, \neg x_2] = \bot \) and \( \psi[y, x_1, x_2] = \bot \), hence \( \exists \)-subcase \( \psi[y, x_1] \) unsat.

- Since \( \exists \)-subcases \( \psi[y, \neg x_1], \psi[y, x_1] \) unsat., also \( \forall \)-subcase \( \psi[y] \) and \( \psi \) unsat.

**Observation:** the clause \((x_2)\) in \( \psi \) can only be satisfied by setting \( x_2 \) to true.

- Every subcase \( \psi[\ldots, \neg x_2, \ldots] \) is unsatisfiable: consider \( \psi[\ldots, x_2, \ldots] \) instead.

- \( \forall \)-subcase \( \psi[x_2, y] = \bot \), hence \( \psi \) unsatisfiable: smaller assignment tree!
Backtracking Search: Example

Example

\( \psi := \forall y \exists x_1, x_2. (x_2) \land (\neg y \lor \neg x_2) \land (\neg y \lor x_1). \)

- Leftmost path: check \( \forall \)-subcase \( \psi[\neg y] \).
  - \( \exists \)-subcase \( \psi[\neg y, x_2] = \top \) already, no need to try \( \psi[\neg y, \neg x_2] \).
    - Backtrack and check \( \psi[y] \).
  - \( \exists \)-subcase \( \psi[y, \neg x_1] = \bot \), flip \( x_1 \) and check \( \psi[y, x_1] \).
  - Both \( \exists \)-subcases \( \psi[y, x_1, \neg x_2] = \bot \) and \( \psi[y, x_1, x_2] = \bot \), hence \( \exists \)-subcase \( \psi[y, x_1] \) unsat.
- Since \( \exists \)-subcases \( \psi[y, \neg x_1], \psi[y, x_1] \) unsat., also \( \forall \)-subcase \( \psi[y] \) and \( \psi \) unsat.

Observation: the clause \( (x_2) \) in \( \psi \) can only be satisfied by setting \( x_2 \) to true.

- Every subcase \( \psi[\ldots, \neg x_2, \ldots] \) is unsatisfiable: consider \( \psi[\ldots, x_2, \ldots] \) instead.
- \( \forall \)-subcase \( \psi[x_2, y] = \bot \), hence \( \psi \) unsatisfiable: smaller assignment tree!
Backtracking Search: Example

Example

\[ \psi := \forall y \exists x_1, x_2. (x_2) \land (\neg y \lor \neg x_2) \land (\neg y \lor x_1). \]

- **Leftmost path:** check \( \forall \)-subcase \( \psi[\neg y] \).
  - \( \exists \)-subcase \( \psi[\neg y, x_2] = \top \) already, no need to try \( \psi[\neg y, \neg x_2] \).
    - Backtrack and check \( \psi[y] \).
  - \( \exists \)-subcase \( \psi[y, \neg x_1] = \bot \), flip \( x_1 \) and check \( \psi[y, x_1] \).
  - Both \( \exists \)-subcases \( \psi[y, x_1, \neg x_2] = \bot \) and \( \psi[y, x_1, x_2] = \bot \), hence \( \exists \)-subcase \( \psi[y, x_1] \) unsat.
  - Since \( \exists \)-subcases \( \psi[y, \neg x_1], \psi[y, x_1] \) unsat., also \( \forall \)-subcase \( \psi[y] \) and \( \psi \) unsat.

**Observation:** the clause \( (x_2) \) in \( \psi \) can only be satisfied by setting \( x_2 \) to true.

- Every subcase \( \psi[\ldots, \neg x_2, \ldots] \) is unsatisfiable: consider \( \psi[\ldots, x_2, \ldots] \) instead.
- \( \forall \)-subcase \( \psi[x_2, y] = \bot \), hence \( \psi \) unsatisfiable: smaller assignment tree!
Backtracking Search: Example

**Example**

\[ \psi := \forall y \exists x_1, x_2. (x_2) \land (\neg y \lor \neg x_2) \land (\neg y \lor x_1). \]

- **Leftmost path**: check \( \forall \)-subcase \( \psi[\neg y] \).
  - \( \exists \)-subcase \( \psi[\neg y, x_2] = \top \) already, no need to try \( \psi[\neg y, \neg x_2] \).
  - Backtrack and check \( \psi[y] \).
- \( \exists \)-subcase \( \psi[y, \neg x_1] = \bot \), flip \( x_1 \) and check \( \psi[y, x_1] \).
- Both \( \exists \)-subcases \( \psi[y, x_1, \neg x_2] = \bot \) and \( \psi[y, x_1, x_2] = \bot \), hence \( \exists \)-subcase \( \psi[y, x_1] \) unsat.
- Since \( \exists \)-subcases \( \psi[y, \neg x_1], \psi[y, x_1] \) unsat., also \( \forall \)-subcase \( \psi[y] \) and \( \psi \) unsat.

**Observation**: the clause \( (x_2) \) in \( \psi \) can only be satisfied by setting \( x_2 \) to true.
- Every subcase \( \psi[\ldots, \neg x_2, \ldots] \) is unsatisfiable: consider \( \psi[\ldots, x_2, \ldots] \) instead.
- \( \forall \)-subcase \( \psi[x_2, y] = \bot \), hence \( \psi \) unsatisfiable: smaller assignment tree!
Backtracking Search: Drawbacks

- Assignments by decisions only (i.e. must be flipped during backtracking).
- Generated assignment trees might contain irrelevant branches.
- Goal: add rules to make assignments other than decisions which can be ignored during backtracking.
- Avoid irrelevant branches resulting from “wrong” decisions.

Example (continued)

\[ \psi := \forall y \exists x_1, x_2. (x_2) \land (\neg y \lor \neg x_2) \land (\neg y \lor x_1). \]

\[
\begin{array}{c}
\neg \neg \neg y \\
\neg x_2 \\
\neg x_1 \\
x_2 \\
x_1 \\
x_1 \\
x_2 \\
x_1 \\
x_2 \\
x_2 \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \\
\bot \

Observe: \( \exists x_2 \) is not leftmost in prefix of \( \psi \).
Improvements to Backtracking Search: Assignment Generation
Definition (Unit Literal Detection)

- Given a QBF $\psi$, a clause $C \in \psi$ is unit if and only if $C = (l)$ and $q(l) = \exists$.
- The existential literal $l$ in $C$ is called a unit literal.
- Unit literal detection $UL(C) := \{l\}$ collects the assignment $\{l\}$ from the unit clause $C = (l)$.
- Unit literal detection on a QBF $\psi$: $UL(\psi) := \bigcup_{C \in \psi} UL(C)$.

Example (continued)

The clause $(x_2)$ in $\psi := \forall y \exists x_1, x_2. (x_2) \land (\neg y \lor \neg x_2) \land (\neg y \lor x_1)$ is unit: $UL(\psi) = \{x_2\}$

Definition (Unit Literal Detection)

- Given a QBF \( \psi \), a clause \( C \in \psi \) is unit if and only if \( C = (l) \) and \( q(l) = \exists \).
- The existential literal \( l \) in \( C \) is called a unit literal.
- Unit literal detection \( UL(C) := \{l\} \) collects the assignment \( \{l\} \) from the unit clause \( C = (l) \).
- Unit literal detection on a QBF \( \psi \): \( UL(\psi) := \bigcup_{C \in \psi} UL(C) \).

Example (continued)

The clause \( (x_2) \) in \( \psi := \forall y \exists x_1, x_2. (x_2 \land (\neg y \lor \neg x_2) \land (\neg y \lor x_1) \) is unit: \( UL(\psi) = \{x_2\} \)

Definition (Pure Literal Detection)

- A literal \( l \) is pure in a QBF \( \psi \) if there are clauses which contain \( l \) but no clauses which contain \( \neg l \).

- Pure literal detection \( PL(\psi) := \bigcup \{ l' \} \) collects the assignment \( \{ l' \} \) such that \( l \) is a pure literal in \( \psi \) and \( l' := l \) if \( q(l) = \exists \) and \( l' := \neg l \) if \( q(l) = \forall \).

- The variable of an existential (universal) pure literal is assigned so that clauses are satisfied (not satisfied) by that assignment.

Example (continued)

The universal literal \( \neg y \) in \( \psi := \forall y \exists x_1, x_2. (x_2) \land (\neg y \lor \neg x_2) \land (\neg y \lor x_1) \) is pure. \( PL(\psi) = \{ y \} \) and \( \psi[y] := \exists x_1, x_2. (x_2) \land (\neg x_2) \land (x_1) \).

Definition (Pure Literal Detection)

- A literal $l$ is pure in a QBF $\psi$ if there are clauses which contain $l$ but no clauses which contain $\neg l$.
- Pure literal detection $PL(\psi) := \bigcup\{l'\}$ collects the assignment $\{l'\}$ such that $l$ is a pure literal in $\psi$ and $l' := l$ if $q(l) = \exists$ and $l' := \neg l$ if $q(l) = \forall$.
- The variable of an existential (universal) pure literal is assigned so that clauses are satisfied (not satisfied) by that assignment.

Example (continued)

The universal literal $\neg y$ in $\psi := \forall y \exists x_1, x_2. (x_2) \land (\neg y \lor \neg x_2) \land (\neg y \lor x_1)$ is pure.
$PL(\psi) = \{y\}$ and $\psi[y] := \exists x_1, x_2. (x_2) \land (\neg x_2) \land (x_1)$.

Definition

Given a clause \( C \), \textit{universal reduction (UR)} on \( C \) produces the clause

\[
UR(C) := C \setminus \{ l \in C \mid q(l) = \forall \text{ and } \forall l' \in C \text{ with } q(l') = \exists : \text{var}(l') < \text{var}(l) \},
\]

where \( < \) is the linear variable ordering given by the quantifier prefix.

- UR deletes “trailing” universal literals from clauses.
- UR shortens clauses.

Example (continued)

Given \( \psi := \forall y \exists x_1, x_2. (x_2) \land (\neg y \lor \neg x_2) \land (\neg y \lor x_1) \).  

By UL: \( \psi[x_2] := \forall y \exists x_1. (\neg y) \land (\neg y \lor x_1) \).  

\( UR((\neg y)) = \emptyset \) in \( \psi[x_2] \).

**Definition**

Given a clause $C$, *universal reduction (UR)* on $C$ produces the clause

$$UR(C) := C \setminus \{l \in C \mid q(l) = \forall \text{ and } \forall l' \in C \text{ with } q(l') = \exists : \text{var}(l') < \text{var}(l)\},$$

where $<$ is the linear variable ordering given by the quantifier prefix.

- UR deletes “trailing” universal literals from clauses.
- UR shortens clauses.

**Example (continued)**

Given $\psi := \forall y \exists x_1, x_2. (x_2) \land (\neg y \lor \neg x_2) \land (\neg y \lor x_1)$.

By UL: $\psi[x_2] := \forall y \exists x_1. (\neg y) \land (\neg y \lor x_1)$.

$UR((-y)) = \emptyset \text{ in } \psi[x_2]$.

Definition

Boolean Constraint Propagation for QBF (QBCP):

- Given a PCNF $\psi$ and the empty assignment $A = \{\}$, i.e. $\psi[A] = \psi$.
  1. Apply universal reduction to $\psi[A]$.
  2. Apply unit literal detection (UL) to $\psi[A]$ to get new assignments by UL.
  3. Apply pure literal detection (PL) to $\psi[A]$ to find new assignments by PL.
- Add assignments found by UL and PL to $A$, repeat steps 1-3.
- Stop if $A$ does not change anymore or if $\psi[A] = \top$ or $\psi[A] = \bot$. 
Properties of QBCP:

- QBCP takes a PCNF $\psi$ and an assignment $A$ and produces an extended assignment $A'$ and a new PCNF $\psi' = \psi[A']$ by UL, PL, and UR.
- Soundness: $\psi \equiv \psi'$ (satisfiability-equivalence).
- No order restriction: QBCP assigns variables from any quantifier block.

QBCP in Practice:

- Combine decision making and QBCP.
- Successively apply QBCP starting with $A = \{x\}$ where $x$ is a decision.
- No need to flip assignments by UL and PL in QBCP during backtracking.
Properties of QBCP:

- QBCP takes a PCNF $\psi$ and an assignment $A$ and produces an extended assignment $A'$ and a new PCNF $\psi' = \psi[A']$ by UL, PL, and UR.
- Soundness: $\psi \equiv \psi'$ (satisfiability-equivalence).
- No order restriction: QBCP assigns variables from any quantifier block.

QBCP in Practice:

- Combine decision making and QBCP.
- Successively apply QBCP starting with $A = \{x\}$ where $x$ is a decision.
- No need to flip assignments by UL and PL in QBCP during backtracking.
Example

- $\psi = \forall y_5 \exists x_1 \forall y_2 \exists x_3, x_4. (\neg y_5 \lor x_4) \land (y_5 \lor \neg x_4) \land (x_1 \lor y_2 \lor \neg x_4) \land (\neg x_1 \lor x_3 \lor \neg x_4) \land (\neg y_2 \lor \neg x_3)$.

- No simplifications of $\psi$ by QBCP possible.

- Make decision: $A = \{y_5\}$.

- $\psi[y_5] = \exists x_1 \forall y_2 \exists x_3, x_4. (x_4) \land (x_1 \lor y_2 \lor \neg x_4) \land (\neg x_1 \lor x_3 \lor \neg x_4) \land (\neg y_2 \lor \neg x_3)$.

- By UL: $\psi[y_5, x_4] = \exists x_1 \forall y_2 \exists x_3. (x_1 \lor y_2) \land (\neg x_1 \lor x_3) \land (\neg y_2 \lor \neg x_3)$.

- By UR: $\psi[y_5, x_4] = \exists x_1 \forall y_2 \exists x_3. (x_1) \land (\neg x_1 \lor x_3) \land (\neg y_2 \lor \neg x_3)$.

- By PL: $\psi[y_5, x_4, y_2] = \exists x_1 \exists x_3. (x_1) \land (\neg x_1 \lor x_3) \land (\neg x_3)$.

- By UL: $\psi[y_5, x_4, y_2, x_1] = \exists x_3. (x_3) \land (\neg x_3)$.

- By UL: $\psi[y_5, x_4, y_2, x_1, x_3] = \bot$.

- By QBCP, we have shown: $\psi[y_5] \equiv \psi[y_5, x_4, y_2, x_1, x_3] \equiv \bot$.

- Since $y_5$ is a universal decision: $\psi[y_5] \equiv \bot \equiv \psi$, only one branch explored.

- Worst case: search tree has $2^5$ branches.
Example

- \( \psi = \forall y_5 \exists x_1 \forall y_2 \exists x_3, x_4. (\neg y_5 \lor x_4) \land (y_5 \lor \neg x_4) \land (x_1 \lor y_2 \lor \neg x_4) \land (\neg x_1 \lor x_3 \lor \neg x_4) \land (\neg y_2 \lor \neg x_3). \)
- No simplifications of \( \psi \) by QBCP possible.
- Make decision: \( A = \{ y_5 \}. \)
  - \( \psi | y_5 = \exists x_1 \forall y_2 \exists x_3, x_4. (x_4) \land (x_1 \lor y_2 \lor \neg x_4) \land (\neg x_1 \lor x_3 \lor \neg x_4) \land (\neg y_2 \lor \neg x_3). \)
  - By UL: \( \psi | y_5, x_4 = \exists x_1 \forall y_2 \exists x_3. (x_1 \lor y_2) \land (\neg x_1 \lor x_3) \land (\neg y_2 \lor \neg x_3). \)
  - By UR: \( \psi | y_5, x_4 = \exists x_1 \forall y_2 \exists x_3. (x_1) \land (\neg x_1 \lor x_3) \land (\neg y_2 \lor \neg x_3). \)
  - By PL: \( \psi | y_5, x_4, y_2 = \exists x_1 \exists x_3. (x_1) \land (\neg x_1 \lor x_3) \land (\neg y_2 \lor \neg x_3). \)
  - By UL: \( \psi | y_5, x_4, y_2, x_1 = \exists x_3. (x_3) \land (\neg x_3). \)
  - By UL: \( \psi | y_5, x_4, y_2, x_1, x_3 = \bot. \)
  - By QBCP, we have shown: \( \psi | y_5 \equiv \psi | y_5, x_4, y_2, x_1, x_3 \equiv \bot. \)
  - Since \( y_5 \) is a universal decision: \( \psi | y_5 \equiv \bot \equiv \psi, \) only one branch explored.
- Worst case: search tree has \( 2^{5} \) branches.
QBCP Example

Example

- $\psi = \forall y_5 \exists x_1 \forall y_2 \exists x_3, x_4. (\neg y_5 \lor x_4) \land (y_5 \lor \neg x_4) \land (x_1 \lor y_2 \lor \neg x_4) \land (\neg x_1 \lor x_3 \lor \neg x_4) \land (\neg y_2 \lor \neg x_3)$.
- No simplifications of $\psi$ by QBCP possible.
- Make decision: $A = \{y_5\}$.
- $\psi[y_5] = \exists x_1 \forall y_2 \exists x_3, x_4. (x_4) \land (x_1 \lor y_2 \lor \neg x_4) \land (\neg x_1 \lor x_3 \lor \neg x_4) \land (\neg y_2 \lor \neg x_3)$.
- By UL: $\psi[y_5, x_4] = \exists x_1 \forall y_2 \exists x_3. (x_1 \lor y_2) \land (\neg x_1 \lor x_3) \land (\neg y_2 \lor \neg x_3)$.
- By UR: $\psi[y_5, x_4] = \exists x_1 \forall y_2 \exists x_3. (x_1) \land (\neg x_1 \lor x_3) \land (\neg y_2 \lor \neg x_3)$.
- By PL: $\psi[y_5, x_4, y_2] = \exists x_1 \exists x_3. (x_1) \land (\neg x_1 \lor x_3) \land (\neg x_3)$.
- By UL: $\psi[y_5, x_4, y_2, x_1] = \exists x_3. (x_3) \land (\neg x_3)$.
- By UL: $\psi[y_5, x_4, y_2, x_1, x_3] = \bot$.
- By QBCP, we have shown: $\psi[y_5] \equiv \psi[y_5, x_4, y_2, x_1, x_3] \equiv \bot$.
- Since $y_5$ is a universal decision: $\psi[y_5] \equiv \bot \equiv \psi$, only one branch explored.
- Worst case: search tree has $2^5$ branches.
Example

- \( \psi = \forall y_5 \exists x_1 \forall y_2 \exists x_3, x_4. (\neg y_5 \lor x_4) \land (y_5 \lor \neg x_4) \land (x_1 \lor y_2 \lor \neg x_4) \land (\neg x_1 \lor x_3 \lor \neg x_4) \land (\neg y_2 \lor \neg x_3). \)
- No simplifications of \( \psi \) by QBCP possible.
- Make decision: \( A = \{y_5\} \).
- \( \psi[y_5] = \exists x_1 \forall y_2 \exists x_3, x_4. (x_4) \land (x_1 \lor y_2 \lor \neg x_4) \land (\neg x_1 \lor x_3 \lor \neg x_4) \land (\neg y_2 \lor \neg x_3). \)
- By UL: \( \psi[y_5, x_4] = \exists x_1 \forall y_2 \exists x_3. (x_1 \lor y_2) \land (\neg x_1 \lor x_3) \land (\neg y_2 \lor \neg x_3). \)
- By UR: \( \psi[y_5, x_4] = \exists x_1 \forall y_2 \exists x_3. (x_1) \land (\neg x_1 \lor x_3) \land (\neg y_2 \lor \neg x_3). \)
- By PL: \( \psi[y_5, x_4, y_2] = \exists x_1 \exists x_3. (x_1) \land (\neg x_1 \lor x_3) \land (\neg x_3). \)
- By UL: \( \psi[y_5, x_4, y_2, x_1] = \exists x_3. (x_3) \land (\neg x_3). \)
- By UL: \( \psi[y_5, x_4, y_2, x_1, x_3] = \bot. \)
- By QBCP, we have shown: \( \psi[y_5] \equiv \psi[y_5, x_4, y_2, x_1, x_3] \equiv \bot. \)
- Since \( y_5 \) is a universal decision: \( \psi[y_5] \equiv \bot \equiv \psi \), only one branch explored.
- Worst case: search tree has \( 2^5 \) branches.
Example

- $\psi = \forall y_5 \exists x_1 \forall y_2 \exists x_3, x_4. (\neg y_5 \vee x_4) \wedge (y_5 \vee \neg x_4) \wedge (x_1 \vee y_2 \vee \neg x_4) \wedge (\neg x_1 \vee x_3 \vee \neg x_4) \wedge (\neg y_2 \vee \neg x_3)$.

- No simplifications of $\psi$ by QBCP possible.

- Make decision: $A = \{y_5\}$.

- $\psi[y_5] = \exists x_1 \forall y_2 \exists x_3, x_4. (x_4) \wedge (x_1 \vee y_2 \vee \neg x_4) \wedge (\neg x_1 \vee x_3 \vee \neg x_4) \wedge (\neg y_2 \vee \neg x_3)$.

- By UL: $\psi[y_5, x_4] = \exists x_1 \forall y_2 \exists x_3. (x_1 \vee y_2) \wedge (\neg x_1 \vee x_3) \wedge (\neg y_2 \vee \neg x_3)$.

- By UR: $\psi[y_5, x_4] = \exists x_1 \forall y_2 \exists x_3. (x_1) \wedge (\neg x_1 \vee x_3) \wedge (\neg y_2 \vee \neg x_3)$.

- By PL: $\psi[y_5, x_4, y_2] = \exists x_1 \exists x_3. (x_1) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_3)$.

- By UL: $\psi[y_5, x_4, y_2, x_1] = \exists x_3. (x_3) \wedge (\neg x_3)$.  

- By UL: $\psi[y_5, x_4, y_2, x_1, x_3] = \bot$.

- By QBCP, we have shown: $\psi[y_5] \equiv \psi[y_5, x_4, y_2, x_1, x_3] \equiv \bot$.

- Since $y_5$ is a universal decision: $\psi[y_5] \equiv \bot \equiv \psi$, only one branch explored.

- Worst case: search tree has $2^5$ branches.
Example

- $\psi = \forall y_5 \exists x_1 \forall y_2 \exists x_3 , x_4 . (\neg y_5 \lor x_4) \land (y_5 \lor \neg x_4) \land (x_1 \lor y_2 \lor \neg x_4) \land (\neg x_1 \lor x_3 \lor \neg x_4) \land (\neg y_2 \lor \neg x_3)$.  
- No simplifications of $\psi$ by QBCP possible.
- Make decision: $A = \{ y_5 \}$.
- $\psi[y_5] = \exists x_1 \forall y_2 \exists x_3 , x_4 . (x_4) \land (x_1 \lor y_2 \lor \neg x_4) \land (\neg x_1 \lor x_3 \lor \neg x_4) \land (\neg y_2 \lor \neg x_3)$.  
- By UL: $\psi[y_5 , x_4] = \exists x_1 \forall y_2 \exists x_3 . (x_1 \lor y_2) \land (\neg x_1 \lor x_3) \land (\neg y_2 \lor \neg x_3)$.  
- By UR: $\psi[y_5 , x_4] = \exists x_1 \forall y_2 \exists x_3 . (x_1) \land (\neg x_1 \lor x_3) \land (\neg y_2 \lor \neg x_3)$.  
- By PL: $\psi[y_5 , x_4 , y_2] = \exists x_1 \exists x_3 . (x_1) \land (\neg x_1 \lor x_3) \land (\neg x_3)$.  
- By UL: $\psi[y_5 , x_4 , y_2 , x_1] = \exists x_3 . (x_3) \land (\neg x_3)$.  
- By UL: $\psi[y_5 , x_4 , y_2 , x_1 , x_3] = \bot$.  
- By QBCP, we have shown: $\psi[y_5] \equiv \psi[y_5 , x_4 , y_2 , x_1 , x_3] \equiv \bot$.  
- Since $y_5$ is a universal decision: $\psi[y_5] \equiv \bot \equiv \psi$, only one branch explored.  
- Worst case: search tree has $2^5$ branches.
Example

- $\psi = \forall y_5 \exists x_1 \forall y_2 \exists x_3, x_4. (\neg y_5 \lor x_4) \land (y_5 \lor \neg x_4) \land (x_1 \lor y_2 \lor \neg x_4) \land (\neg x_1 \lor x_3 \lor \neg x_4) \land (\neg y_2 \lor \neg x_3)$.

- No simplifications of $\psi$ by QBCP possible.

- Make decision: $A = \{y_5\}$.

- $\psi[y_5] = \exists x_1 \forall y_2 \exists x_3, x_4. (x_4) \land (x_1 \lor y_2 \lor \neg x_4) \land (\neg x_1 \lor x_3 \lor \neg x_4) \land (\neg y_2 \lor \neg x_3)$.

- By UL: $\psi[y_5, x_4] = \exists x_1 \forall y_2 \exists x_3. (x_1 \lor y_2) \land (\neg x_1 \lor x_3) \land (\neg y_2 \lor \neg x_3)$.

- By UR: $\psi[y_5, x_4] = \exists x_1 \forall y_2 \exists x_3. (x_1) \land (\neg x_1 \lor x_3) \land (\neg y_2 \lor \neg x_3)$.

- By PL: $\psi[y_5, x_4, y_2] = \exists x_1 \exists x_3. (x_1) \land (\neg x_1 \lor x_3) \land (\neg x_3)$.

- By UL: $\psi[y_5, x_4, y_2, x_1] = \exists x_3. (x_3) \land (\neg x_3)$.

- By UL: $\psi[y_5, x_4, y_2, x_1, x_3] = \bot$.

- By QBCP, we have shown: $\psi[y_5] \equiv \psi[y_5, x_4, y_2, x_1, x_3] \equiv \bot$.

- Since $y_5$ is a universal decision: $\psi[y_5] \equiv \bot \equiv \psi$, only one branch explored.

- Worst case: search tree has $2^5$ branches.
**Example**

- \( \psi = \forall y_5 \exists x_1 \forall y_2 \exists x_3, x_4. (\neg y_5 \lor x_4) \land (y_5 \lor \neg x_4) \land (x_1 \lor y_2 \lor \neg x_4) \land (\neg x_1 \lor x_3 \lor \neg x_4) \land (\neg y_2 \lor \neg x_3). \)
- No simplifications of \( \psi \) by QBCP possible.
- Make decision: \( A = \{y_5\}. \)
- \( \psi[y_5] = \exists x_1 \forall y_2 \exists x_3, x_4. (x_4) \land (x_1 \lor y_2 \lor \neg x_4) \land (\neg x_1 \lor x_3 \lor \neg x_4) \land (\neg y_2 \lor \neg x_3). \)
- By UL: \( \psi[y_5, x_4] = \exists x_1 \forall y_2 \exists x_3. (x_1 \lor y_2) \land (\neg x_1 \lor x_3) \land (\neg y_2 \lor \neg x_3). \)
- By UR: \( \psi[y_5, x_4] = \exists x_1 \forall y_2 \exists x_3. (x_1) \land (\neg x_1 \lor x_3) \land (\neg y_2 \lor \neg x_3). \)
- By PL: \( \psi[y_5, x_4, y_2] = \exists x_1 \exists x_3. (x_1) \land (\neg x_1 \lor x_3) \land (\neg x_3). \)
- By UL: \( \psi[y_5, x_4, y_2, x_1] = \exists x_3. (x_3) \land (\neg x_3). \)
- By UL: \( \psi[y_5, x_4, y_2, x_1, x_3] = \bot. \)
- By QBCP, we have shown: \( \psi[y_5] \equiv \psi[y_5, x_4, y_2, x_1, x_3] \equiv \bot. \)
- Since \( y_5 \) is a universal decision: \( \psi[y_5] \equiv \bot \equiv \psi, \) only one branch explored.
- Worst case: search tree has \( 2^5 \) branches.
Example

- \( \psi = \forall y_5 \exists x_1 \forall y_2 \exists x_3, x_4. (\neg y_5 \lor x_4) \land (y_5 \lor \neg x_4) \land (x_1 \lor y_2 \lor \neg x_4) \land (\neg x_1 \lor x_3 \lor \neg x_4) \land (\neg y_2 \lor \neg x_3) \).
- No simplifications of \( \psi \) by QBCP possible.
- Make decision: \( A = \{ y_5 \} \).
- \( \psi[y_5] = \exists x_1 \forall y_2 \exists x_3, x_4. (x_4) \land (x_1 \lor y_2 \lor \neg x_4) \land (\neg x_1 \lor x_3 \lor \neg x_4) \land (\neg y_2 \lor \neg x_3) \).
- By UL: \( \psi[y_5, x_4] = \exists x_1 \forall y_2 \exists x_3. (x_1 \lor y_2) \land (\neg x_1 \lor x_3) \land (\neg y_2 \lor \neg x_3) \).
- By UR: \( \psi[y_5, x_4] = \exists x_1 \forall y_2 \exists x_3. (x_1) \land (\neg x_1 \lor x_3) \land (\neg y_2 \lor \neg x_3) \).
- By PL: \( \psi[y_5, x_4, y_2] = \exists x_1 \exists x_3. (x_1) \land (\neg x_1 \lor x_3) \land (\neg x_3) \).
- By UL: \( \psi[y_5, x_4, y_2, x_1] = \exists x_3. (x_3) \land (\neg x_3) \).
- By UL: \( \psi[y_5, x_4, y_2, x_1, x_3] = \bot \).
- By QBCP, we have shown: \( \psi[y_5] \equiv \psi[y_5, x_4, y_2, x_1, x_3] \equiv \bot \).
- Since \( y_5 \) is a universal decision: \( \psi[y_5] \equiv \bot \equiv \psi \), only one branch explored.
- Worst case: search tree has \( 2^5 \) branches.
Iterative Search-Based QBF Solving (QDPLL)

QDPLL:
- QBF-specific variant of the DPLL algorithm for propositional logic [DLL62].
- Original descriptions [GNT01, CGS98] both recursive and iterative.
- Start with empty assignment.
- Decisions open a new $\exists/\forall$-subcase.
- Function $\text{qbcp}$ applies UL, PL, UR and simplifications to extend the assignment corresponding to the current $\exists/\forall$-subcase.
- Function $\text{analyze}$: retraction of assignments, flipping a decision variable by backtracking.

Result $\text{qdpll} (\text{PCNF } f)$

Result $r = \text{UNDEF}$;
Assignment $a = \{\}$;
while (true)
  /* Simplify. */
  $(r, a) = \text{qbcp} (f, a);$  
  if ($r == \text{UNDET}$)
    /* Decision making. */
    $a = \text{assign_dec_var} (f, a);$  
  else
    /* Backtracking. */
    /* $r == \text{UNSAT} \text{ or } r == \text{SAT} */
    $\text{btlevel} = \text{analyze} (r, a);$  
    if ($\text{btlevel} == \text{INVALID}$)
      return $r$;
    else
      $a = \text{backtrack} (\text{btlevel})$;

Search-Based QBF Solving: Iterative vs. Recursive

Result qdpl1 (PCNF f)
Result r = UNDEF;
Assignment a = {};
while (true)
/* Simplify. */
(r,a) = qbcp (f,a);
if (r == UNDET)
/* Decision making. */
a = assign_dec_var (f,a);
else
/* Backtracking. */
/* r == UNSAT or r == SAT */
btlevel = analyze (r,a);
if (btlevel == INVALID)
    return r;
else
    a = backtrack (btlevel);

bool bt_search (PCNF Qxψ, Assignment A)
/* 1. Simplify under given assignment. */
ψ' := simplify(Qxψ[A]);
/* 2. Check base cases. */
if (ψ' == ⊥)
    return false;
if (ψ' == ⊤)
    return true;
/* 3. Decision making, backtracking. */
if (Q == ∃)
    return bt_search (ψ', A ∪ {¬x}) ||
    bt_search (ψ', A ∪ {x});
if (Q == ∀)
    return bt_search (ψ', A ∪ {¬x}) &&
    bt_search (ψ', A ∪ {x});

Comparison:
- bt_search very close to recursive semantics.
- qdpl1 explicitly enumerates paths (i.e. assignments) in assignment trees.
- QBCP makes the difference between qdpl1 and bt_search.
- Structure of qdpl1 is close to implementations of modern QBF solvers.
Search-Based QBF Solving: Iterative vs. Recursive

Result qdpll (PCNF f)

Result r = UNDEF;
Assignment a = {};
while (true)
    /* Simplify. */
    (r,a) = qbcp (f,a);
    if (r == UNDET)
        /* Decision making. */
        a = assign_dec_var (f,a);
    else
        /* Backtracking. */
        /* r == UNSAT or r == SAT */
        btlevel = analyze (r,a);
        if (btlevel == INVALID)
            return r;
        else
            a = backtrack (btlevel);

bool bt_search (PCNF Qxψ, Assignment A)

    /* 1. Simplify under given assignment. */
    ψ' := simplify(Qxψ[A]);
    /* 2. Check base cases. */
    if (ψ' == ⊥)
        return false;
    if (ψ' == ⊤)
        return true;
    /* 3. Decision making, backtracking. */
    if (Q == ∃)
        return bt_search (ψ', A ∪ {¬x}) ||
        bt_search (ψ', A ∪ {x});
    if (Q == ∀)
        return bt_search (ψ', A ∪ {¬x}) &&
        bt_search (ψ', A ∪ {x});

Comparison:

- bt_search very close to recursive semantics.
- qdpll explicitly enumerates paths (i.e. assignments) in assignment trees.
- QBCP makes the difference between qdpll and bt_search.
- Structure of qdpll is close to implementations of modern QBF solvers.
Search-Based QBF Solving: Iterative vs. Recursive

Result qdpl1 (PCNF f)
Result r = UNDEF;
Assignment a = {}; 
while (true)
    /* Simplify. */
    (r,a) = qbcp (f,a);
    if (r == UNDET)
        /* Decision making. */
        a = assign_dec_var (f,a);
    else
        /* Backtracking. */
        /* r == UNSAT or r == SAT */
        btlevel = analyze (r,a);
        if (btlevel == INVALID)
            return r;
        else
            a = backtrack (btlevel);

bool bt_search (PCNF Qxψ, Assignment A)
    /* 1. Simplify under given assignment. */
    ψ' := simplify(Qxψ[A]);
    /* 2. Check base cases. */
    if (ψ' == ⊥)
        return false;
    if (ψ' == ⊤)
        return true;
    /* 3. Decision making, backtracking. */
    if (Q == ∃)
        return bt_search (ψ', A ∪ {¬x}) || bt_search (ψ', A ∪ {x});
    if (Q == ∀)
        return bt_search (ψ', A ∪ {¬x}) && bt_search (ψ', A ∪ {x});

Comparison:

- bt_search very close to recursive semantics.
- qdpl1 explicitly enumerates paths (i.e. assignments) in assignment trees.
- QBCP makes the difference between qdpl1 and bt_search.
- Structure of qdpl1 is close to implementations of modern QBF solvers.
Search-Based QBF Solving: Iterative vs. Recursive

Result qdpll (PCNF f)
Result r = UNDEF;
Assignment a = {};
while (true)
    /* Simplify. */
    (r,a) = qbcp (f,a);
    if (r == UNDET)
        /* Decision making. */
        a = assign_dec_var (f,a);
    else
        /* Backtracking. */
        /* r == UNSAT or r == SAT */
        btlevel = analyze (r,a);
        if (btlevel == INVALID)
            return r;
        else
            a = backtrack (btlevel);

bool bt_search (PCNF Qxψ, Assignment A)
/* 1. Simplify under given assignment. */
ψ′ := simplify(Qxψ[A]);
/* 2. Check base cases. */
if (ψ′ == ⊥)
    return false;
if (ψ′ == ⊤)
    return true;
/* 3. Decision making, backtracking. */
if (Q == ∃)
    return bt_search (ψ′, A ∪ {¬x}) ||
            bt_search (ψ′, A ∪ {x});
if (Q == ∀)
    return bt_search (ψ′, A ∪ {¬x}) &&
            bt_search (ψ′, A ∪ {x});

Comparison:
- bt_search very close to recursive semantics.
- qdpll explicitly enumerates paths (i.e. assignments) in assignment trees.
- QBCP makes the difference between qdpll and bt_search.
- Structure of qdpll is close to implementations of modern QBF solvers.
Search-Based QBF Solving: Iterative vs. Recursive

Result qdpll (PCNF f)
Result r = UNDEF;
Assignment a = {};
while (true)
  /* Simplify. */
  (r,a) = qbcp (f,a);
  if (r == UNDET)
    /* Decision making. */
    a = assign_dec_var (f,a);
  else
    /* Backtracking. */
    /* r == UNSAT or r == SAT */
    btlevel = analyze (r,a);
    if (btlevel == INVALID)
      return r;
    else
      a = backtrack (btlevel);

bool bt_search (PCNF Qxψ, Assignment A)
  /* 1. Simplify under given assignment. */
  ψ’ := simplify(Qxψ[A]);
  /* 2. Check base cases. */
  if (ψ' == ⊥)
    return false;
  if (ψ' == ⊤)
    return true;
  /* 3. Decision making, backtracking. */
  if (Q == ∃)
    return bt_search (ψ’, A ∪ {¬x}) ||
            bt_search (ψ’, A ∪ {x});
  if (Q == ∀)
    return bt_search (ψ’, A ∪ {¬x}) &&
            bt_search (ψ’, A ∪ {x});

Comparison:
- bt_search very close to recursive semantics.
- qdpll explicitly enumerates paths (i.e. assignments) in assignment trees.
- QBCP makes the difference between qdpll and bt_search.
- Structure of qdpll is close to implementations of modern QBF solvers.
Improvements to Backtracking Search: Backtracking is not optimal
Assignments:

- Represented as sequence $A = \{l_1, l_2, \ldots, l_n\}$ of literals.
- Assignments due to decisions and QBCP (UL, PL).
- Literals $l_i$ are ordered chronologically as they were assigned.
- Conflict: assignment $A$ such that $\psi[A] = \bot$.
- Solution: assignment $A$ such that $\psi[A] = \top$. 
Chronological Backtracking:

- Given a conflict $A = \{\ldots, d, \ldots, l_n\}$ where $d$ is the most-recent *unflipped* existential decision.
- Given a solution $A = \{\ldots, d, \ldots, l_n\}$ where $d$ is the most-recent *unflipped* universal decision.
- No such $d$: formula solved.
- Retract decision $d$ and later assignments: $A' = A \setminus \{d, \ldots, l_n\}$.
- Set the variable of $d$ to the opposite value (flip): $A' = A' \cup \{\neg d\}$.
- Continue with $A = A'$.

Snippet of qdpl1:
```c
/* Backtracking. */
/* r == UNSAT or r == SAT */
btlevel = analyze (r,a);
if (btlevel == INVALID)
    return r;
else
    a = backtrack (btlevel);
```
Example

\[ \psi = \exists x_1, x_2, x_3, x_4 \forall y_5 \exists x_6. (\neg x_3 \lor x_4) \land (x_3 \lor x_4) \land (\neg x_4 \lor x_6) \land (\neg x_1 \lor y_5 \lor \neg x_6) \land \phi. \]

1. Assume that \( \phi \) contains further clauses.
2. Decision on \( x_1 \): \( A = A \cup \{x_1\} \).
3. Decision on \( x_2 \): \( A = A \cup \{x_2\} \).
4. Decision on \( x_3 \): \( A = A \cup \{x_3\} \).
5. \( \psi[x_1, x_2, x_3] = \exists x_4 \forall y_5 \exists x_6. (x_4) \land (\neg x_4 \lor x_6) \land (y_5 \lor \neg x_6) \).
6. By QBCP (UL): \( A = A \cup \{x_4, x_6\} \).
7. By QBCP (UR): conflict \( A = \{x_1, x_2, x_3, x_4, x_6\} \), \( \psi[A] = \bot \).
8. Flip \( x_3 \), get conflict \( A = \{x_1, x_2, \neg x_3, x_4, x_6\} \), where again \( x_4, x_6 \) by UL.
9. Flip \( x_2 \), assume that no conflict/solution is found with \( A = \{x_1, \neg x_2\} \).
10. Continue with a decision on \( x_3 \): \( A = \{x_1, \neg x_2, x_3\} \) or \( A = \{x_1, \neg x_2, \neg x_3\} \).
11. In any case, get a conflict by \( \{x_1, \neg x_2, x_3, x_4, x_6\} \) and \( \{x_1, \neg x_2, \neg x_3, x_4, x_6\} \).
12. Repeated subassignments \( \{x_3, x_4, x_6\} \), \( \{\neg x_3, x_4, x_6\} \) of conflicts (steps 7,8).
13. Flipping \( x_2 \) did not resolve the conflict, redundant work in steps 9-11.
QDPLL with Chronological Backtracking (1/2)

Example

\[
\psi = \exists x_1, x_2, x_3, x_4 \forall y_5 \exists x_6. (\neg x_3 \lor x_4) \land (x_3 \lor x_4) \land (\neg x_4 \lor x_6) \land (\neg x_1 \lor y_5 \lor \neg x_6) \land \phi.
\]

1. Assume that \( \phi \) contains further clauses.

2. Decision on \( x_1 \): \( A = A \cup \{ x_1 \} \).

3. Decision on \( x_2 \): \( A = A \cup \{ x_2 \} \).

4. Decision on \( x_3 \): \( A = A \cup \{ x_3 \} \).

5. \( \psi[x_1, x_2, x_3] = \exists x_4 \forall y_5 \exists x_6. (x_4) \land (\neg x_4 \lor x_6) \land (y_5 \lor \neg x_6) \).

6. By QBCP (UL): \( A = A \cup \{ x_4, x_6 \} \).

7. By QBCP (UR): conflict \( A = \{ x_1, x_2, x_3, x_4, x_6 \} \), \( \psi[A] = \bot \).

8. Flip \( x_3 \), get conflict \( A = \{ x_1, x_2, \neg x_3, x_4, x_6 \} \), where again \( x_4, x_6 \) by UL.

9. Flip \( x_2 \), assume that no conflict/solution is found with \( A = \{ x_1, \neg x_2 \} \).

10. Continue with a decision on \( x_3 \): \( A = \{ x_1, \neg x_2, x_3 \} \) or \( A = \{ x_1, \neg x_2, \neg x_3 \} \).

11. In any case, get a conflict by \( \{ x_1, \neg x_2, x_3, x_4, x_6 \} \) and \( \{ x_1, \neg x_2, \neg x_3, x_4, x_6 \} \).

12. Repeated subassignments \( \{ x_3, x_4, x_6 \}, \{ \neg x_3, x_4, x_6 \} \) of conflicts (steps 7,8).

13. Flipping \( x_2 \) did not resolve the conflict, redundant work in steps 9-11.
Example

\[ \psi = \exists x_1, x_2, x_3, x_4 \forall y_5 \exists x_6. (\neg x_3 \lor x_4) \land (x_3 \lor x_4) \land (\neg x_4 \lor x_6) \land (\neg x_1 \lor y_5 \lor \neg x_6) \land \phi. \]

1. Assume that \( \phi \) contains further clauses.
2. Decision on \( x_1 \): \( A = A \cup \{x_1\} \).
3. Decision on \( x_2 \): \( A = A \cup \{x_2\} \).
4. Decision on \( x_3 \): \( A = A \cup \{x_3\} \).
5. \( \psi[x_1, x_2, x_3] = \exists x_4 \forall y_5 \exists x_6. (x_4) \land (\neg x_4 \lor x_6) \land (y_5 \lor \neg x_6) \).
6. By QBCP (UL): \( A = A \cup \{x_4, x_6\} \).
7. By QBCP (UR): conflict \( A = \{x_1, x_2, x_3, x_4, x_6\}, \psi[A] = \bot \).
8. Flip \( x_3 \), get conflict \( A = \{x_1, x_2, \neg x_3, x_4, x_6\} \), where again \( x_4, x_6 \) by UL.
9. Flip \( x_2 \), assume that no conflict/solution is found with \( A = \{x_1, \neg x_2\} \).
10. Continue with a decision on \( x_3 \): \( A = \{x_1, \neg x_2, x_3\} \) or \( A = \{x_1, \neg x_2, \neg x_3\} \).
11. In any case, get a conflict by \( \{x_1, \neg x_2, x_3, x_4, x_6\} \) and \( \{x_1, \neg x_2, \neg x_3, x_4, x_6\} \).
12. Repeated subassignments \( \{x_3, x_4, x_6\}, \{\neg x_3, x_4, x_6\} \) of conflicts (steps 7,8).
13. Flipping \( x_2 \) did not resolve the conflict, redundant work in steps 9-11.
Example

$$\psi = \exists x_1, x_2, x_3, x_4 \forall y_5 \exists x_6. (\neg x_3 \lor x_4) \land (x_3 \lor x_4) \land (\neg x_4 \lor x_6) \land (\neg x_1 \lor y_5 \lor \neg x_6) \land \phi.$$  

1. Assume that $\phi$ contains further clauses.
2. Decision on $x_1$: $A = A \cup \{x_1\}$.
3. Decision on $x_2$: $A = A \cup \{x_2\}$.
4. Decision on $x_3$: $A = A \cup \{x_3\}$.
5. $\psi[x_1, x_2, x_3] = \exists x_4 \forall y_5 \exists x_6. (x_4) \land (\neg x_4 \lor x_6) \land (y_5 \lor \neg x_6)$.
6. By QBCP (UL): $A = A \cup \{x_4, x_6\}$.
7. By QBCP (UR): conflict $A = \{x_1, x_2, x_3, x_4, x_6\}$, $\psi[A] = \bot$.
8. Flip $x_3$, get conflict $A = \{x_1, x_2, \neg x_3, x_4, x_6\}$, where again $x_4, x_6$ by UL.
9. Flip $x_2$, assume that no conflict/solution is found with $A = \{x_1, \neg x_2\}$.
10. Continue with a decision on $x_3$: $A = \{x_1, \neg x_2, x_3\}$ or $A = \{x_1, \neg x_2, \neg x_3\}$.
11. In any case, get a conflict by $\{x_1, \neg x_2, x_3, x_4, x_6\}$ and $\{x_1, \neg x_2, \neg x_3, x_4, x_6\}$.
12. Repeated subassignments $\{x_3, x_4, x_6\}, \{\neg x_3, x_4, x_6\}$ of conflicts (steps 7,8).
13. Flipping $x_2$ did not resolve the conflict, redundant work in steps 9-11.
QDPLL with Chronological Backtracking (1/2)

Example

\[ \psi = \exists x_1, x_2, x_3, x_4 \forall y_5 \exists x_6 . (\neg x_3 \lor x_4) \land (x_3 \lor x_4) \land (\neg x_4 \lor x_6) \land (\neg x_1 \lor y_5 \lor \neg x_6) \land \phi. \]

1. Assume that \( \phi \) contains further clauses.
2. Decision on \( x_1 \): \( A = A \cup \{x_1\} \).
3. Decision on \( x_2 \): \( A = A \cup \{x_2\} \).
4. Decision on \( x_3 \): \( A = A \cup \{x_3\} \).
5. \( \psi[x_1, x_2, x_3] = \exists x_4 \forall y_5 \exists x_6 . (x_4) \land (\neg x_4 \lor x_6) \land (y_5 \lor \neg x_6) \).
6. By QBCP (UL): \( A = A \cup \{x_4, x_6\} \).
7. By QBCP (UR): conflict \( A = \{x_1, x_2, x_3, x_4, x_6\} \), \( \psi[A] = \bot \).
8. Flip \( x_3 \), get conflict \( A = \{x_1, x_2, \neg x_3, x_4, x_6\} \), where again \( x_4, x_6 \) by UL.
9. Flip \( x_2 \), assume that no conflict/solution is found with \( A = \{x_1, \neg x_2\} \).
10. Continue with a decision on \( x_3 \): \( A = \{x_1, \neg x_2, x_3\} \) or \( A = \{x_1, \neg x_2, \neg x_3\} \).
11. In any case, get a conflict by \( \{x_1, \neg x_2, x_3, x_4, x_6\} \) and \( \{x_1, \neg x_2, \neg x_3, x_4, x_6\} \).
12. Repeated subassignments \( \{x_3, x_4, x_6\} \), \( \{\neg x_3, x_4, x_6\} \) of conflicts (steps 7,8).
13. Flipping \( x_2 \) did not resolve the conflict, redundant work in steps 9-11.
**Example**

$\psi = \exists x_1, x_2, x_3, x_4 \forall y_5 \exists x_6. (\neg x_3 \lor x_4) \land (x_3 \lor x_4) \land (\neg x_4 \lor x_6) \land (\neg x_1 \lor y_5 \lor \neg x_6) \land \phi$.

1. Assume that $\phi$ contains further clauses.
2. Decision on $x_1$: $A = A \cup \{x_1\}$.
3. Decision on $x_2$: $A = A \cup \{x_2\}$.
4. Decision on $x_3$: $A = A \cup \{x_3\}$.
5. $\psi[x_1, x_2, x_3] = \exists x_4 \forall y_5 \exists x_6. (x_4) \land (\neg x_4 \lor x_6) \land (y_5 \lor \neg x_6)$.
6. By QBCP (UL): $A = A \cup \{x_4, x_6\}$.
7. By QBCP (UR): conflict $A = \{x_1, x_2, x_3, x_4, x_6\}$, $\psi[A] = \bot$.
8. Flip $x_3$, get conflict $A = \{x_1, x_2, \neg x_3, x_4, x_6\}$, where again $x_4, x_6$ by UL.
9. Flip $x_2$, assume that no conflict/solution is found with $A = \{x_1, \neg x_2\}$.
10. Continue with a decision on $x_3$: $A = \{x_1, \neg x_2, x_3\}$ or $A = \{x_1, \neg x_2, \neg x_3\}$.
11. In any case, get a conflict by $\{x_1, \neg x_2, x_3, x_4, x_6\}$ and $\{x_1, \neg x_2, \neg x_3, x_4, x_6\}$.
12. Repeated subassignments $\{x_3, x_4, x_6\}$, $\{\neg x_3, x_4, x_6\}$ of conflicts (steps 7,8).
13. Flipping $x_2$ did not resolve the conflict, redundant work in steps 9-11.
QDPLL with Chronological Backtracking (1/2)

Example

\[ \psi = \exists x_1, x_2, x_3, x_4 \forall y_5 \exists x_6. (\neg x_3 \lor x_4) \land (x_3 \lor x_4) \land (\neg x_4 \lor x_6) \land (\neg x_1 \lor y_5 \lor \neg x_6) \land \phi. \]

1. Assume that \( \phi \) contains further clauses.
2. Decision on \( x_1 \): \( A = A \cup \{x_1\} \).
3. Decision on \( x_2 \): \( A = A \cup \{x_2\} \).
4. Decision on \( x_3 \): \( A = A \cup \{x_3\} \).
5. \( \psi[x_1, x_2, x_3] = \exists x_4 \forall y_5 \exists x_6. (x_4) \land (\neg x_4 \lor x_6) \land (y_5 \lor \neg x_6) \).
6. By QBCP (UL): \( A = A \cup \{x_4, x_6\} \).
7. By QBCP (UR): conflict \( A = \{x_1, x_2, x_3, x_4, x_6\} \), \( \psi[A] = \bot \).
8. Flip \( x_3 \), get conflict \( A = \{x_1, x_2, \neg x_3, x_4, x_6\} \), where again \( x_4, x_6 \) by UL.
9. Flip \( x_2 \), assume that no conflict/solution is found with \( A = \{x_1, \neg x_2\} \).
10. Continue with a decision on \( x_3 \): \( A = \{x_1, \neg x_2, x_3\} \) or \( A = \{x_1, \neg x_2, \neg x_3\} \).
11. In any case, get a conflict by \( \{x_1, \neg x_2, x_3, x_4, x_6\} \) and \( \{x_1, \neg x_2, \neg x_3, x_4, x_6\} \).
12. Repeated subassignments \( \{x_3, x_4, x_6\}, \{\neg x_3, x_4, x_6\} \) of conflicts (steps 7,8).
13. Flipping \( x_2 \) did not resolve the conflict, redundant work in steps 9-11.
Example

\[ \psi = \exists x_1, x_2, x_3, x_4 \forall y_5 \exists x_6. (\neg x_3 \lor x_4) \land (x_3 \lor x_4) \land (\neg x_4 \lor x_6) \land (\neg x_1 \lor y_5 \lor \neg x_6) \land \phi. \]

1. Assume that \( \phi \) contains further clauses.
2. Decision on \( x_1 \): \( A = A \cup \{x_1\} \).
3. Decision on \( x_2 \): \( A = A \cup \{x_2\} \).
4. Decision on \( x_3 \): \( A = A \cup \{x_3\} \).
5. \( \psi[x_1, x_2, x_3] = \exists x_4 \forall y_5 \exists x_6. (x_4) \land (\neg x_4 \lor x_6) \land (y_5 \lor \neg x_6) \).
6. By QBCP (UL): \( A = A \cup \{x_4, x_6\} \).
7. By QBCP (UR): conflict \( A = \{x_1, x_2, x_3, x_4, x_6\} \), \( \psi[A] = \bot \).
8. Flip \( x_3 \), get conflict \( A = \{x_1, x_2, \neg x_3, x_4, x_6\} \), where again \( x_4, x_6 \) by UL.
9. Flip \( x_2 \), assume that no conflict/solution is found with \( A = \{x_1, \neg x_2\} \).
10. Continue with a decision on \( x_3 \): \( A = \{x_1, \neg x_2, x_3\} \) or \( A = \{x_1, \neg x_2, \neg x_3\} \).
11. In any case, get a conflict by \( \{x_1, \neg x_2, x_3, x_4, x_6\} \) and \( \{x_1, \neg x_2, \neg x_3, x_4, x_6\} \).
12. Repeated subassignments \( \{x_3, x_4, x_6\} \), \( \{\neg x_3, x_4, x_6\} \) of conflicts (steps 7,8).
13. Flipping \( x_2 \) did not resolve the conflict, redundant work in steps 9-11.
QDPLL with Chronological Backtracking (1/2)

Example

\[ \psi = \exists x_1, x_2, x_3, x_4 \forall y_5 \exists x_6. (\neg x_3 \lor x_4) \land (x_3 \lor x_4) \land (\neg x_4 \lor x_6) \land (\neg x_1 \lor y_5 \lor \neg x_6) \land \phi. \]

1. Assume that \( \phi \) contains further clauses.
2. Decision on \( x_1 \): \( A = A \cup \{x_1\} \).
3. Decision on \( x_2 \): \( A = A \cup \{x_2\} \).
4. Decision on \( x_3 \): \( A = A \cup \{x_3\} \).
5. \( \psi[x_1, x_2, x_3] = \exists x_4 \forall y_5 \exists x_6. (x_4) \land (\neg x_4 \lor x_6) \land (y_5 \lor \neg x_6) \).
6. By QBCP (UL): \( A = A \cup \{x_4, x_6\} \).
7. By QBCP (UR): conflict \( A = \{x_1, x_2, x_3, x_4, x_6\} \), \( \psi[A] = \bot \).
8. Flip \( x_3 \), get conflict \( A = \{x_1, x_2, \neg x_3, x_4, x_6\} \), where again \( x_4, x_6 \) by UL.
9. Flip \( x_2 \), assume that no conflict/solution is found with \( A = \{x_1, \neg x_2\} \).
10. Continue with a decision on \( x_3 \): \( A = \{x_1, \neg x_2, x_3\} \) or \( A = \{x_1, \neg x_2, \neg x_3\} \).
11. In any case, get a conflict by \( \{x_1, \neg x_2, x_3, x_4, x_6\} \) and \( \{x_1, \neg x_2, \neg x_3, x_4, x_6\} \).
12. Repeated subassignments \( \{x_3, x_4, x_6\} \), \( \{\neg x_3, x_4, x_6\} \) of conflicts (steps 7,8).
13. Flipping \( x_2 \) did not resolve the conflict, redundant work in steps 9-11.
Example

\[\psi = \exists x_1, x_2, x_3, x_4 \forall y_5 \exists x_6. (\neg x_3 \lor x_4) \land (x_3 \lor x_4) \land (\neg x_4 \lor x_6) \land (\neg x_1 \lor y_5 \lor \neg x_6) \land \phi.\]

1. Assume that \( \phi \) contains further clauses.

2. Decision on \( x_1 \): \( A = A \cup \{x_1\} \).

3. Decision on \( x_2 \): \( A = A \cup \{x_2\} \).

4. Decision on \( x_3 \): \( A = A \cup \{x_3\} \).

5. \( \psi[x_1, x_2, x_3] = \exists x_4 \forall y_5 \exists x_6. (x_4) \land (\neg x_4 \lor x_6) \land (y_5 \lor \neg x_6) \).

6. By QBCP (UL): \( A = A \cup \{x_4, x_6\} \).

7. By QBCP (UR): conflict \( A = \{x_1, x_2, x_3, x_4, x_6\} \), \( \psi[A] = \bot \).

8. Flip \( x_3 \), get conflict \( A = \{x_1, x_2, \neg x_3, x_4, x_6\} \), where again \( x_4, x_6 \) by UL.

9. Flip \( x_2 \), assume that no conflict/solution is found with \( A = \{x_1, \neg x_2\} \).

10. Continue with a decision on \( x_3 \): \( A = \{x_1, \neg x_2, x_3\} \) or \( A = \{x_1, \neg x_2, \neg x_3\} \).

11. In any case, get a conflict by \( \{x_1, \neg x_2, x_3, x_4, x_6\} \) and \( \{x_1, \neg x_2, \neg x_3, x_4, x_6\} \).

12. Repeated subassignments \( \{x_3, x_4, x_6\}, \{\neg x_3, x_4, x_6\} \) of conflicts (steps 7,8).

13. Flipping \( x_2 \) did not resolve the conflict, redundant work in steps 9-11.
Example (continued)

\[ \psi = \exists x_1, x_2, x_3, x_4 \forall y_5 \exists x_6. (\neg x_3 \lor x_4) \land (x_3 \lor x_4) \land (\neg x_4 \lor x_6) \land (\neg x_1 \lor y_5 \lor \neg x_6) \land \phi. \]

Conflicts generated by QDPLL:

- \( A = \{x_1, x_2, x_3, x_4, x_6\} \).
- \( A = \{x_1, x_2, \neg x_3, x_4, x_6\} \).
- \( A = \{x_1, \neg x_2, x_3, x_4, x_6\} \).
- \( A = \{x_1, \neg x_2, \neg x_3, x_4, x_6\} \).

Same conflicting subtrees after flipping \( x_2 \).
- Decision \( x_2 \) is irrelevant in this context.

Drawback of Chronological Backtracking:

- Flipping variables which are irrelevant for the current conflict/solution.
- Repeating subassignments of previous conflicts: redundant work, needless branches.
Example (continued)

\[ \psi = \exists x_1, x_2, x_3, x_4 \forall y_5 \exists x_6. (\neg x_3 \lor x_4) \land (x_3 \lor x_4) \land (\neg x_4 \lor x_6) \land (\neg x_1 \lor y_5 \lor \neg x_6) \land \phi. \]

Conflicts generated by QDPLL:

- \( A = \{x_1, x_2, x_3, x_4, x_6\} \).
- \( A = \{x_1, x_2, \neg x_3, x_4, x_6\} \).
- \( A = \{x_1, \neg x_2, x_3, x_4, x_6\} \).
- \( A = \{x_1, \neg x_2, \neg x_3, x_4, x_6\} \).
- Same conflicting subtrees after flipping \( x_2 \).
- Decision \( x_2 \) is irrelevant in this context.

Drawback of Chronological Backtracking:

- Flipping variables which are irrelevant for the current conflict/solution.
- Repeating subassignments of previous conflicts: redundant work, needless branches.
QBCP and Implication Graphs

Definition (implication graph as a levelized graph)

- Vertices: literals in $A$ (variable assignments), special vertex $\emptyset$ denoting a clause $C \in \psi$ such that $C[A] = \bot$ (conflicting clause).
- For assignments $\{l\}$ by UL from a unit clause $C[A]$: the clause $ante(l) := C$ is the antecedent clause of the assignment $\{l\}$.
- Define $ante(\emptyset) = C$, for a clause $C \in \psi$ such that $C[A] = \bot$.
- Edges: $(x, y) \in E$ if $y$ assigned by UL and literal $\neg x \in ante(y)$.

Example (continued)

$\psi = \exists x_1, x_2, x_3, x_4 \forall y_5 \exists x_6. (\neg x_3 \lor x_4) \land (x_3 \lor x_4) \land (\neg x_4 \lor x_6) \land (\neg x_1 \lor y_5 \lor \neg x_6) \land \phi$.

Implication graph for conflict

$A = \{x_1, x_2, x_3, x_4, x_6\}$

where $x_1, x_2,$ and $x_3$ are decisions.

Note: UR applied to get $\emptyset$.

- Implication graph is implicitly constructed during QBCP.
- Similar to BCP in SAT solvers, but QBCP includes additional rules PL, UR.
QBCP and Implication Graphs

Definition (implication graph as a levelized graph)

- Vertices: literals in $A$ (variable assignments), special vertex $\emptyset$ denoting a clause $C \in \psi$ such that $C[A] = \bot$ (conflicting clause).
- For assignments $\{l\}$ by UL from a unit clause $C[A]$: the clause $\text{ante}(l) := C$ is the antecedent clause of the assignment $\{l\}$.
- Define $\text{ante}(\emptyset) = C$, for a clause $C \in \psi$ such that $C[A] = \bot$.
- Edges: $(x, y) \in E$ if $y$ assigned by UL and literal $\neg x \in \text{ante}(y)$.

Example (continued)

$\psi = \exists x_1, x_2, x_3, x_4 \forall y_5 \exists x_6. (\neg x_3 \lor x_4) \land (x_3 \lor x_4) \land (\neg x_4 \lor x_6) \land (\neg x_1 \lor y_5 \lor \neg x_6) \land \phi$.

Implication graph for conflict

$A = \{x_1, x_2, x_3, x_4, x_6\}$

where $x_1, x_2$, and $x_3$ are decisions.

Note: UR applied to get $\emptyset$.

- Implication graph is implicitly constructed during QBCP.
- Similar to BCP in SAT solvers, but QBCP includes additional rules PL, UR.
Non-Chronological Backtracking — Backjumping (1/3)

Idea: given a conflict $A$ and the implication graph.

1. Start at the conflicting clause $\emptyset$ and traverse the implication graph backwards.
2. Collect all decisions reachable from $\emptyset$: conflict set.
3. Retract all assignments made after the second most recent existential decision in the conflict set.
4. Flip the most recent unflipped existential decision in the conflict set.

Example (continued)

$\psi = \exists x_1,x_2,x_3,x_4 \forall y_5 \exists x_6. (\neg x_3 \lor x_4) \land (x_3 \lor x_4) \land (\neg x_4 \lor x_6) \land (\neg x_1 \lor y_5 \lor \neg x_6) \land \phi.$

- Conflict $A = \{x_1, x_2, x_3, x_4, x_6\}$, decisions $x_1, x_2, x_3$.
- Steps 1,2: conflict set $\{x_1, x_3\}$.
- Step 3: retract $\{x_2, x_3, x_4, x_6\}$ from $A$.
- Step 4: flip $x_3$, $A \cup \{\neg x_3\} = \{x_1, \neg x_3\}$.
- “Jump over” irrelevant, non-reachable $x_2$. 
Idea: given a conflict $A$ and the implication graph.

1. Start at the conflicting clause $\emptyset$ and traverse the implication graph backwards.
2. Collect all decisions reachable from $\emptyset$: conflict set.
3. Retract all assignments made after the second most recent existential decision in the conflict set.
4. Flip the most recent unflipped existential decision in the conflict set.

Example (continued)

$$\psi = \exists x_1, x_2, x_3, x_4 \forall y_5 \exists x_6. (\neg x_3 \lor x_4) \land (x_3 \lor x_4) \land (\neg x_4 \lor x_6) \land (\neg x_1 \lor y_5 \lor \neg x_6) \land \phi.$$ 

- Conflict $A = \{ x_1, x_2, x_3, x_4, x_6 \}$, decisions $x_1, x_2, x_3$.
- Steps 1,2: conflict set $\{ x_1, x_3 \}$.
- Step 3: retract $\{ x_2, x_3, x_4, x_6 \}$ from $A$.
- Step 4: flip $x_3$, $A \cup \{ \neg x_3 \} = \{ x_1, \neg x_3 \}$.
- “Jump over” irrelevant, non-reachable $x_2$. 
Example (continued)

\[ \psi = \exists x_1, x_2, x_3, x_4 \forall y_5 \exists x_6. (\neg x_3 \lor x_4) \land (x_3 \lor x_4) \land (\neg x_4 \lor x_6) \land (\neg x_1 \lor y_5 \lor \neg x_6) \land \phi. \]

- After flipping \( x_3 \), get conflict \( A = \{x_1, \neg x_3, x_4, x_6\} \), decisions \( x_1, \neg x_3 \).
- Conflict set \( \{x_1, \neg x_3\} \).
- Retract \( \{\neg x_3, x_4, x_6\} \) from \( A \).
- Flip \( x_1 \), \( A \cup \{\neg x_1\} = \{\neg x_1\} \).

Chronological backtracking:

Non-chronological backtracking:
Properties of Backjumping:

- "backtracking" = "chronological backtracking".
- "backjumping" = "non-chronological backtracking".
- Potential retraction of irrelevant decisions, exponential reduction of branches in assignment trees.
- Children of nodes in assignment trees might have different labels: \( x_2, \neg x_3 \) in the example.
- Similar approaches to backjump from solutions, i.e. \( \psi[A] = \top \).
- Fundamentally different from traditional recursive backtracking search.

Example (continued)

Snippet of `bt_search`:

```c
/* 3. Decision making, backtracking. */
if (Q == ∃)
    return bt_search (ψ', A ∪ \{¬x\}) ||
    bt_search (ψ', A ∪ \{x\});
if (Q == ∀)
    return bt_search (ψ', A ∪ \{¬x\}) &&
    bt_search (ψ', A ∪ \{x\});
```
Backtracking and Backjumping in QDPLL

Implementation:

- Function `analyze` must be adapted.
- Stop if there is no decision to be flipped in the conflict set.
- Think of backtracking like a variant of backjumping where the conflict set always contains all decisions made.

```plaintext
Result qdpll (PCNF f)
Result r = UNDEF;
Assignment a = {};
while (true)
    /* Simplify. */
    (r,a) = qbcp (f,a);
    if (r == UNDET)
        /* Decision making. */
        a = assign_dec_var (f,a);
    else
        /* Backtracking. */
        /* r == UNSAT or r == SAT */
        btlevel = analyze (r,a);
        if (btlevel == INVALID)
            return r;
        else
            a = backtrack (btlevel);
```
Example (continued)

\[ \psi = \exists x_1, x_2, x_3, x_4 \forall y_5 \exists x_6. \neg x_3 \lor x_4 \land x_3 \lor x_4 \land \neg x_4 \lor x_6 \land \neg x_1 \lor y_5 \lor \neg x_6 \land \phi. \]

- Assume that the assignment tree on the right is a subtree of a bigger tree.
- Observe: every assignment \( A \) with \( \{x_1, x_4\} \subseteq A \) is a conflict (under QBCP).
- QBCP extends \( \{x_1, x_4\} \) to \( \{x_1, x_4, x_6\} \) by UL.
- Clause \( (\neg x_1 \lor y_5 \lor \neg x_6)[x_1, x_4, x_6] = \bot \).
- The subassignment \( \{x_1, x_4\} \) can be repeated in other branches, the same clause is conflicting.
- Backjumping cannot avoid this problem.
Improvements to Backtracking Search: Backjumping is not optimal
Idea:

- A clause \((l_1 \lor l_2 \lor \ldots \lor l_k)\) is conflicting under the assignment \(\{\neg l_1, \ldots, \neg l_k\}\).
- QDPLL tries to satisfy clauses by unit literal detection in QBCP.
- Clauses in a PCNF guide QDPLL away from conflicts.
- Intuition(!): if a subassignment \(A = \{l_1, l_2, \ldots, l_k\}\) is responsible for a conflict then add the clause \((\neg l_1 \lor \neg l_2 \lor \ldots \lor \neg l_k)\) to the PCNF.
- QDPLL tries to satisfy the added clause by assigning \(\neg l_i\) for at least one \(l_i\).
- QDPLL will not enumerate assignments \(A'\) such that \(A \subseteq A'\).
Example (continued)
\[ \psi = \exists x_1, x_2, x_3, x_4 \forall y_5 \exists x_6. (\neg x_3 \lor x_4) \land (x_3 \lor x_4) \land (\neg x_4 \lor x_6) \land (\neg x_1 \lor y_5 \lor \neg x_6) \land \phi. \]

- Every assignment \( A \) with \( \{x_1, x_4\} \subseteq A \) is a conflict (under QBCP).
- Adding the clause \( (\neg x_1 \lor \neg x_4) \) to \( \psi \) prevents QDPLL from repeating the subassignment \( \{x_1, x_4\} \) in other branches.
- Assigning \( x_1 (x_4) \) triggers the assignment of \( \neg x_4 (\neg x_1) \) by unit literal detection in QBCP.
Properties:
- “clause learning” = adding clauses obtained from analyzing a conflict.
- “learned clause” = added clause.
- In general, adding arbitrary clauses to a PCNF $\psi$ can make $\psi$ unsatisfiable.
- Correctness of clause learning: $\psi \equiv \psi \land C$.

In Practice:
- Checking if $\psi \equiv \psi \land C$ is PSPACE-complete.
- How to efficiently find clauses $C$ which can safely be added to $\psi$?

Resolution:
- Given the PCNF $\psi' = \psi \land C_1 \land C_2$, the resolution operation produces a new clause $C_r$ (resolvent) from $C_1$ and $C_2$ such that $\psi' \equiv \psi' \land C_r$.
- By construction, a resolvent can safely be added to a PCNF.
- Idea: use resolution to produce learned clauses.

Clause Learning (3/9)

Properties:
- “clause learning” = adding clauses obtained from analyzing a conflict.
- “learned clause” = added clause.
- In general, adding arbitrary clauses to a PCNF $\psi$ can make $\psi$ unsatisfiable.
- Correctness of clause learning: $\psi \equiv \psi \land C$.

In Practice:
- Checking if $\psi \equiv \psi \land C$ is PSPACE-complete.
- How to efficiently find clauses $C$ which can safely be added to $\psi$?

Resolution:
- Given the PCNF $\psi' = \psi \land C_1 \land C_2$, the resolution operation produces a new clause $C_r$ (resolvent) from $C_1$ and $C_2$ such that $\psi' \equiv \psi' \land C_r$.
- By construction, a resolvent can safely be added to a PCNF.
- Idea: use resolution to produce learned clauses.

Clause Learning (3/9)

Properties:
- “clause learning” = adding clauses obtained from analyzing a conflict.
- “learned clause” = added clause.
- In general, adding arbitrary clauses to a PCNF $\psi$ can make $\psi$ unsatisfiable.
- Correctness of clause learning: $\psi \equiv \psi \land C$.

In Practice:
- Checking if $\psi \equiv \psi \land C$ is PSPACE-complete.
- How to efficiently find clauses $C$ which can safely be added to $\psi$?

Resolution:
- Given the PCNF $\psi' = \psi \land C_1 \land C_2$, the resolution operation produces a new clause $C_r$ (resolvent) from $C_1$ and $C_2$ such that $\psi' \equiv \psi' \land C_r$.
- By construction, a resolvent can safely be added to a PCNF.
- Idea: use resolution to produce learned clauses.

Q-Resolution:
- Combination of universal reduction and resolution for propositional logic.
- Q-resolvents $C$ can safely be added to a PCNF because $\psi \equiv \psi \land C$.

Definition (Q-Resolution)
- Let $C_1, C_2$ be non-tautological clauses where $v \in C_1, \neg v \in C_2$ for an $\exists$-variable $v$.
- Variable $v$ is the pivot of the Q-resolution step.
- Tentative Q-resolvent of $C_1$ and $C_2$: $C_1 \otimes C_2 := (UR(C_1) \setminus \{v\}) \cup (UR(C_2) \setminus \{\neg v\})$.
- If $\{x, \neg x\} \subseteq C_1 \otimes C_2$ for some variable $x$, then no Q-resolvent exists.
- Otherwise, the non-tautological Q-resolvent is $C := UR(C_1 \otimes C_2)$.
Q-Resolution:

- Combination of universal reduction and resolution for propositional logic.
- Q-resolvents $C$ can safely be added to a PCNF because $\psi \equiv \psi \land C$.

**Definition (Q-Resolution)**

- Let $C_1$, $C_2$ be *non-tautological* clauses where $v \in C_1$, $\neg v \in C_2$ for an $\exists$-variable $v$.
- Variable $v$ is the *pivot* of the Q-resolution step.
- *Tentative Q-resolvent* of $C_1$ and $C_2$: $C_1 \otimes C_2 := (UR(C_1) \setminus \{v\}) \cup (UR(C_2) \setminus \{\neg v\})$.
- If $\{x, \neg x\} \subseteq C_1 \otimes C_2$ for some variable $x$, then no Q-resolvent exists.
- Otherwise, the non-tautological *Q-resolvent* is $C := UR(C_1 \otimes C_2)$. 
Example (continued)

\[ \psi = \exists x_1, x_2, x_3, x_4 \forall y_5 \exists x_6. (\neg x_3 \lor x_4) \land (x_3 \lor x_4) \land (\neg x_4 \lor x_6) \land (\neg x_1 \lor y_5 \lor \neg x_6) \land \phi. \]

- **Conflict** \( A = \{x_1, x_2, x_3, x_4, x_6\} \),
  decisions \( x_1, x_2, x_3 \).

- **Idea**: consider antecedent clauses by unit literal detection and the conflicting clause for possible Q-resolutions, in reverse assignment order.

- Resolve \( ante(\emptyset) = (\neg x_1 \lor y_5 \lor \neg x_6) \) and
  \( ante(x_6) = (\neg x_4 \lor x_6) \), get tentative Q-resolvent
  \( (\neg x_1 \lor \neg x_4 \lor y_5) \) and finally the Q-resolvent
  \( (\neg x_1 \lor \neg x_4) \) by UR.

- **Add** \( (\neg x_1 \lor \neg x_4) \) as a learned clause.
Example (continued)

\[ \psi = \exists x_1, x_2, x_3, x_4 \forall y_5 \exists x_6. (\neg x_3 \lor x_4) \land (x_3 \lor x_4) \land (\neg x_4 \lor x_6) \land (\neg x_1 \lor y_5 \lor \neg x_6) \land \phi. \]

- Add \((\neg x_1 \lor \neg x_4)\) as a learned clause.
- Retract \(\{ x_2, x_3, x_4, x_6 \} \), continue with \(A = \{ x_1 \}\).
- By QBCP the learned clause \((\neg x_1 \lor \neg x_4)\) is unit and \(A = \{ x_1, \neg x_4 \}\).
- Further, clause \((x_3 \lor x_4)\) is unit and \(A = \{ x_1, \neg x_4, x_3 \}\).
- Conflict \(A = \{ x_1, \neg x_4, x_3 \}\), clause \((\neg x_3 \lor x_4)[A] = \bot\) conflicting.
- Resolve \(ante(\emptyset) = (\neg x_3 \lor x_4)\) and \(ante(x_3) = (x_3 \lor x_4)\), get Q-resolvent \((x_4)\).

After learning \((\neg x_1 \lor \neg x_4)\), continue with \(A = \{ x_1 \}\):
\( \psi = \exists x_1, x_2, x_3, x_4 \forall y_5 \exists x_6. (\neg x_3 \lor x_4) \land (x_3 \lor x_4) \land (\neg x_4 \lor x_6) \land (\neg x_1 \lor y_5 \lor \neg x_6) \land \phi. \)

- Resolve \( \text{ante}(\emptyset) = (\neg x_3 \lor x_4) \) and \( \text{ante}(x_3) = (x_3 \lor x_4) \), get Q-resolvent \( x_4 \).
- Add \( x_4 \) as a learned clause.
- Retract \( \{x_1, \neg x_4, x_3\} \), continue with \( A = \{\} \).
- By QBCP, get \( A = \{x_4, \neg x_1, x_6\} \) since the two learned clauses became unit.

...
Example (continued)

\[ \psi = \exists x_1, x_2, x_3, x_4 \forall y_5 \exists x_6. (\neg x_3 \lor x_4) \land (x_3 \lor x_4) \land (\neg x_4 \lor x_6) \land (\neg x_1 \lor y_5 \lor \neg x_6) \land \phi. \]

- Only three decisions in left branch:
  \[ x_1, x_2, x_3. \]

- Other branches due to learned clauses which become unit after backtracking.

- Right branch: assignments \[ x_4, \neg x_1, x_6 \]
  by unit literal detection due to learned clauses \textit{without} decisions.

- Note: we never flipped decision variables explicitly.
Properties:

- Decisions are not explicitly flipped (unlike in backjumping).
- Our focus: learned clauses always become unit in QBCP after retracting assignments.
- Fundamentally different from backjumping and traditional backtracking.
- More powerful than backjumping: learned clauses prune search space.
- QDPLL learns the empty clause if and only if $\psi$ is unsatisfiable.

Novel View on Search-Based Solving with Clause Learning:

- Assignment-driven engines searching for a Q-resolution proof (of unsatisfiable QBFs).
- Traditional backtracking view does not fit any more (also applies to SAT solvers).
- More appropriate name: conflict-driven clause learning (CDCL) for QBF (QCDCL).
Properties:

- Decisions are not explicitly flipped (unlike in backjumping).
- Our focus: learned clauses always become unit in QBCP after retracting assignments.
- Fundamentally different from backjumping and traditional backtracking.
- More powerful than backjumping: learned clauses prune search space.
- QDPLL learns the empty clause if and only if $\psi$ is unsatisfiable.

Novel View on Search-Based Solving with Clause Learning:

- Assignment-driven engines searching for a Q-resolution proof (of unsatisfiable QBFs).
- Traditional backtracking view does not fit any more (also applies to SAT solvers).
- More appropriate name: conflict-driven clause learning (CDCL) for QBF (QCDCL).
Modern Search-Based QBF Solving:

- Implementation: analyze must be adapted.
- Clause learning in analyze.
- Backtracking based on learned clause.
- No explicit flipping of decisions.
- Challenges: efficient implementation.

```python
Result qdplll (PCNF f)
Result r = UNDEF;
Assignment a = {};
while (true)
    /* Simplify. */
    (r,a) = qbcp (f,a);
    if (r == UNDET)
        /* Decision making. */
        a = assign_dec_var (f,a);
    else
        /* Backtracking. */
        /* r == UNSAT or r == SAT */
        btlevel = analyze (r,a);
        if (btlevel == INVALID)
            return r;
        else
            a = backtrack (btlevel);
```
In reverse assignment order, resolve on existential variables which were assigned as unit literals, using clauses (i.e. antecedents) which became unit during QBCP.

- Tautological resolvents by universal literals might occur but must be avoided: deviate from strict reverse assignment order [GNT06].
- Worst case exponential number of intermediate resolvents [VG12].

Example

Assignment $A = \{x_1, x_2, x_3, x_4\}$

Assignment order: $x_1, x_2, x_3, x_4$

Can we resolve in reverse assignment order?

Clause $(\neg x_3 \lor \neg x_4)$ conflicting:

$x_1 \rightarrow x_2 \rightarrow x_3 \rightarrow \emptyset$

$x_4 \rightarrow x_1$
Example

\[ \exists x_1, x_3, x_4 \forall y_5 \exists x_2 \]
\[ (\neg x_1 \lor x_2) \land \]
\[ (x_3 \lor y_5 \lor \neg x_2) \land \]
\[ (x_4 \lor \neg y_5 \lor \neg x_2) \land \]
\[ (\neg x_3 \lor \neg x_4) \]

Assignment \( A = \{ x_1, x_2, x_3, x_4 \} \)
Assignment order: \( x_1, x_2, x_3, x_4 \)
Resolve on: \( x_4, x_2 (!), x_3, x_2 \)

Derivation of learned clause \( (\neg x_1) \):\
\[ (\neg x_3 \lor \neg x_4) \]
\[ (x_4 \lor \neg y_5 \lor \neg x_2) \]
\[ (\neg x_1 \lor x_2) \]
\[ (\neg x_3 \lor \neg y_5 \lor \neg x_2) \]
\[ (\neg x_1 \lor x_2) \]
\[ (\neg x_1 \lor \neg x_3) \]
\[ (x_3 \lor y_5 \lor \neg x_2) \]
\[ (\neg x_1 \lor \neg x_3) \]
\[ (\neg x_1 \lor x_2) \]
\[ (\neg x_1 \lor y_5 \lor \neg x_2) \]

Clause \( (\neg x_3 \lor \neg x_4) \) conflicting:

\[ x_1 \rightarrow x_2 \rightarrow x_3 \rightarrow \emptyset \]
\[ x_4 \]

- Linear-time procedure: resolve in assignment order [LEVG13].
Towards a Proof System for PCNFs
**Cube Learning:**

- Given a solution $A$, i.e. $\psi[A] = T$.
- Solution $A$ is a branch in the assignment tree with a $T$-leaf.
- All clauses in $\psi$ are satisfied under $A$.
- Idea: record $A$ as a conjunction of literals: *learned cube*.
- Dual to clauses, learned cubes become unit in QBCP after backtracking and prevent the solver from enumerating the same subassignment.

**Example**

- Satisfiable PCNF $\psi = \forall x \exists y. (x \lor \neg y) \land (\neg x \lor y)$.

$$\begin{array}{c}
\neg r \\
\neg x \\
\neg y \\
\bot \\
x \\
\neg y \\
\bot \\
y \\
T
\end{array}$$
Learning from Solutions (2/4)

Definition (model generation rule [GNT06])

Given a PCNF $\psi := \hat{Q}.\phi$ and a solution $A$, i.e. $\psi[A] = \top$. An initial cube $C = (\bigwedge_{l_i \in A} l_i)$ is a conjunction over the literals of a solution $A$.

Example

$\psi := \exists x_1 \forall y_8 \exists x_5, x_2, x_6, x_4. (y_8 \lor \neg x_5) \land (x_2 \lor \neg x_6) \land (\neg x_1 \lor x_4) \land (\neg y_8 \lor \neg x_4) \land (x_1 \lor x_6) \land (x_4 \lor x_5)$.

Solution $A_1 := \{x_6, x_2, \neg y_8, \neg x_5, x_4\}$, initial cube $C_1 := (x_6 \land x_2 \land \neg y_8 \land \neg x_5 \land x_4)$.

Solution $A_2 := \{y_8, \neg x_4, \neg x_1, x_5, x_6, x_2\}$, initial cube $C_2 := (y_8 \land \neg x_4 \land \neg x_1 \land x_5 \land x_6 \land x_2)$.
Learning from Solutions (3/4)

Definition

Given a cube $C$, existential reduction (ER) on $C$ produces the cube

$$ER(C) := C \setminus \{ l \in C \mid q(l) = \exists \text{ and } \forall l' \in C \text{ with } q(l') = \forall : \text{var}(l') < \text{var}(l)\},$$

where $<$ is the linear variable ordering given by the quantifier prefix.

- ER is dual to universal reduction, deletes “trailing” existential literals from cubes.
- ER shortens cubes.

Example (continued)

$$\psi := \exists x_1 \forall y_8 \exists x_5, x_2, x_6, x_4. (y_8 \lor \neg x_5) \land (x_2 \lor \neg x_6) \land (\neg x_1 \lor x_4) \land (\neg y_8 \lor \neg x_4) \land (x_1 \lor x_6) \land (x_4 \lor x_5).$$

Initial cube $C_1 := (x_6 \land x_2 \land \neg y_8 \land \neg x_5 \land x_4)$.

$C_3 := ER(C_1) = (\neg y_8)$

Initial cube $C_2 := (y_8 \land \neg x_4 \land \neg x_1 \land x_5 \land x_6 \land x_2)$.

$C_4 := ER(C_2) = (y_8 \land \neg x_1)$
Learning from Solutions (4/4)

**Definition (cube resolution [GNT06, ZM02])**

Given two non-contradictory cubes $C_1$ and $C_2$, *cube resolution* is defined analogously to Q-resolution for clauses, except:

- existential reduction.
- universal variables as pivots.

The cube resolvent of $C_1$ and $C_2$ (if it exists) is denoted by $C := C_1 \otimes C_2$.

**Example (continued)**

$$\psi := \exists x_1 \forall y_8 \exists x_5, x_2, x_6, x_4. (y_8 \lor \neg x_5) \land (x_2 \lor \neg x_6) \land (\neg x_1 \lor x_4) \land (\neg y_8 \lor \neg x_4) \land (x_1 \lor x_6) \land (x_4 \lor x_5).$$

Initial cube $C_1 := (x_6 \land x_2 \land \neg y_8 \land \neg x_5 \land x_4)$.

$C_3 := ER(C_1) = (\neg y_8)$

Initial cube $C_2 := (y_8 \land \neg x_4 \land \neg x_1 \land x_5 \land x_6 \land x_2)$.

$C_4 := ER(C_2) = (y_8 \land \neg x_1)$

$C_5 := C_3 \otimes C_4 = (\neg x_1)$, $ER(C_5) = \emptyset$. 
QCDCL as a Proof System

Cube Learning:
- Model generation, existential reduction, cube resolution.
- The empty cube is derived if and only if the PCNF satisfiable.
- Dual to clause learning: driven by assignment generation, implication graphs.

Clause Learning:
- Universal reduction, Q-resolution.
- The empty clause is derived if and only if the PCNF unsatisfiable.

Definition

- \( \text{PCNF } \psi := \hat{Q}. \phi \) with quantifier prefix \( \hat{Q} \) and CNF \( \phi \).
- \( \text{Augmented CNF of } \psi: \psi' := \hat{Q}. (\phi \land \theta \lor \gamma) \).
- Original clauses \( \phi \).
- Learned clauses \( \theta \), filled during clause learning.
- Learned cubes \( \gamma \), filled during cube learning.
- Properties: \( \hat{Q}. \phi \equiv \hat{Q}. (\phi \land \theta) \) and \( \hat{Q}. \phi \equiv \hat{Q}. (\phi \lor \gamma) \).
QCDCL generates proofs.

Proof generation is driven by assignments.

QCDCL does not flip decision variables explicitly.

Backtracking is driven by learned clauses and cubes.

Problem: CNF is bad for cube learning.

Result qdpll (PCNF f)
Result r = UNDEF;
Assignment a = {};
while (true)
/* Simplify. */
(r,a) = qbcp (f,a);
if (r == UNDET)
/* Decision making. */
a = assign_dec_var (f,a);
else
/* Backtracking. */
/* r == UNSAT or r == SAT */
btlevel = analyze (r,a);
if (btlevel == INVALID)
return r;
else
a = backtrack (btlevel);
Optimizations

Inspired by efficient solvers for propositional logic (SAT).

Restarts:
- Periodically retract all assignments and start over with $A = \{}$.
- Makes the solver incomplete unless restart period grows sufficiently large.
- Idea: getting out of “bad” regions in the search space.

Assignment Caching:
- Store assigned values other than decisions in a per-variable cache.
- If variable $x$ is selected to make a decision, then assign cached value of $x$.
- Idea: re-use previous assignments in similar parts of the formula.

Deletion of Learned Clauses and Cubes:
- Formula grows steadily by addition of learned clauses and cubes.
- QBCP will be slowed down.
- Idea: heuristically discard unimportant clauses and cubes.
Optimizations

Inspired by efficient solvers for propositional logic (SAT).

**Restarts:**
- Periodically retract all assignments and start over with $A = \emptyset$.
- Makes the solver incomplete unless restart period grows sufficiently large.
- Idea: getting out of “bad” regions in the search space.

**Assignment Caching:**
- Store assigned values other than decisions in a per-variable cache.
- If variable $x$ is selected to make a decision, then assign cached value of $x$.
- Idea: re-use previous assignments in similar parts of the formula.

**Deletion of Learned Clauses and Cubes:**
- Formula grows steadily by addition of learned clauses and cubes.
- QBCP will be slowed down.
- Idea: heuristically discard unimportant clauses and cubes.
Optimizations

Inspired by efficient solvers for propositional logic (SAT).

**Restarts:**
- Periodically retract all assignments and start over with \( A = \{ \} \).
- Makes the solver incomplete unless restart period grows sufficiently large.
- Idea: getting out of “bad” regions in the search space.

**Assignment Caching:**
- Store assigned values other than decisions in a per-variable cache.
- If variable \( x \) is selected to make a decision, then assign cached value of \( x \).
- Idea: re-use previous assignments in similar parts of the formula.

**Deletion of Learned Clauses and Cubes:**
- Formula grows steadily by addition of learned clauses and cubes.
- QBCP will be slowed down.
- Idea: heuristically discard unimportant clauses and cubes.
Optimizations

Inspired by efficient solvers for propositional logic (SAT).

**Restarts:**
- Periodically retract all assignments and start over with $A = \{\}$.  
- Makes the solver incomplete unless restart period grows sufficiently large.  
- Idea: getting out of “bad” regions in the search space.

**Assignment Caching:**
- Store assigned values other than decisions in a per-variable cache.  
- If variable $x$ is selected to make a decision, then assign cached value of $x$.  
- Idea: re-use previous assignments in similar parts of the formula.

**Deletion of Learned Clauses and Cubes:**
- Formula grows steadily by addition of learned clauses and cubes.  
- QBCP will be slowed down.  
- Idea: heuristically discard unimportant clauses and cubes.
Challenge: Variants of Clause/Cube Learning

**Traditional Q-Resolution:**
- Tautological resolvents $C$, i.e. where $\{v, \neg v\} \subseteq C$, are disallowed.
- In general, tautological resolvents might produce unsound results.

**Long-Distance (LD) Resolution:**
- Allow to produce certain tautological resolvents: soundness.
- Can produce exponentially shorter proofs than Q-resolution.
- Implemented in yQuaffle, DepQBF for clause learning.

**QU-Resolution:**
- Allow to resolve over universally quantified variables.
- Can produce exponentially shorter proofs than Q-resolution.

**Future Work:**
- How to integrate QU-resolution systematically into QCDCL?
Challenge: Variants of Clause/Cube Learning

Traditional Q-Resolution:
- Tautological resolvents $C$, i.e. where $\{v, \neg v\} \subseteq C$, are disallowed.
- In general, tautological resolvents might produce unsound results.

Long-Distance (LD) Resolution:
- Allow to produce *certain* tautological resolvents: soundness.
- Can produce exponentially shorter proofs than Q-resolution.
- Implemented in yQuaffle, DepQBF for clause learning.

QU-Resolution:
- Allow to resolve over universally quantified variables.
- Can produce exponentially shorter proofs than Q-resolution.

Future Work:
- How to integrate QU-resolution systematically into QCDCL?
Challenge: Variants of Clause/Cube Learning

**Traditional Q-Resolution:**
- Tautological resolvents $C$, i.e. where $\{v, \neg v\} \subseteq C$, are disallowed.
- In general, tautological resolvents might produce unsound results.

**Long-Distance (LD) Resolution:**
- Allow to produce *certain* tautological resolvents: soundness.
- Can produce exponentially shorter proofs than Q-resolution.
- Implemented in yQuaffle, DepQBF for clause learning.

**QU-Resolution:**
- Allow to resolve over universally quantified variables.
- Can produce exponentially shorter proofs than Q-resolution.

**Future Work:**
- How to integrate QU-resolution systematically into QCDCL?
Challenge: Variants of Clause/Cube Learning

Traditional Q-Resolution:
- Tautological resolvents $C$, i.e. where $\{v, \neg v\} \subseteq C$, are disallowed.
- In general, tautological resolvents might produce unsound results.

Long-Distance (LD) Resolution:
- Allow to produce *certain* tautological resolvents: soundness.
- Can produce exponentially shorter proofs than Q-resolution.
- Implemented in yQuaffle, DepQBF for clause learning.

QU-Resolution:
- Allow to resolve over universally quantified variables.
- Can produce exponentially shorter proofs than Q-resolution.

Future Work:
- How to integrate QU-resolution systematically into QCDCL?
Order of Assignments:

- Given the PCNF \(Q_1B_1Q_2B_2\ldots Q_mB_m.\phi\), QDPLL in general must only assign variables as decisions starting from \(B_1\) to ensure soundness.

Example

The PCNF \(\psi = \forall x \exists y.(x \lor \neg y) \land (\neg x \lor y)\) is satisfiable.
The PCNF \(\psi = \exists y \forall x.(x \lor \neg y) \land (\neg x \lor y)\) is unsatisfiable.

- This linear ordering limits the freedom to select decision variables.

Example

\(\psi = Q_1B_1\ldots Q_mB_n.\phi \land \phi'(B_n)\).

- \(\phi\): hard formula.
- \(\phi'(B_n)\): easy formula over variables in \(B_n\).
- QDPLL tends to assign variables in \(B_n\) late (except in QBCP).
- QDPLL attempts to solve hard \(\phi\) first.
Dependency Analysis:

- Are there variables which can be moved to the left end in the quantifier prefix without changing the satisfiability of $\psi$?
- Related work: quantifier shifting / miniscoping in theorem proving.
- PSPACE-complete problem.

Dependency Schemes:

- Binary relation $D \subseteq V \times V$ over variables in a PCNF.
- If $(x, y) \in D$ then must assign $x$ before $y$ to ensure soundness.
- If $(x, y) \notin D$ then can assign $x$ before $y$ or vice versa.
- $D$ computed by a syntactic analysis of the PCNF.
- Tradeoff: efficiency of computation and precision.
- Dependency Schemes in DepQBF: efficient integration as compact graphs.
Challenge: Variable Dependencies (2/2)

Dependency Analysis:
- Are there variables which can be moved to the left end in the quantifier prefix without changing the satisfiability of $\psi$?
- Related work: quantifier shifting / miniscoping in theorem proving.
- PSPACE-complete problem.

Dependency Schemes:
- Binary relation $D \subseteq V \times V$ over variables in a PCNF.
- If $(x, y) \in D$ then must assign $x$ before $y$ to ensure soundness.
- If $(x, y) \notin D$ then can assign $x$ before $y$ or vice versa.
- $D$ computed by a syntactic analysis of the PCNF.
- Tradeoff: efficiency of computation and precision.
- Dependency Schemes in DepQBF: efficient integration as compact graphs.
Recent Trends

**Preprocessing:**
- Impressive reduction in formula size and solving time.
- Can be harmful for solvers relying on formula structure.
- Applications: preprocessing can destroy the original encoding (certificates).

**CNF-based Solving and Structure Reconstruction:**
- Dedicated non-PCNF solvers operating on e.g. circuit structure.
- Recent focus on CNF data structures due to efficiency.
- Structure reconstruction to extract circuit information from a CNF.
- Can improve cube learning: shorter cubes, exponential gap.
Recent Trends

Preprocessing:
- Impressive reduction in formula size and solving time.
- Can be harmful for solvers relying on formula structure.
- Applications: preprocessing can destroy the original encoding (certificates).

CNF-based Solving and Structure Reconstruction:
- Dedicated non-PCNF solvers operating on e.g. circuit structure.
- Recent focus on CNF data structures due to efficiency.
- Structure reconstruction to extract circuit information from a CNF.
- Can improve cube learning: shorter cubes, exponential gap.
Summary

Search-Based QBF Solving (QDPLL):
- Originates from backtracking search (1960s), like SAT solvers.
- Modern implementations: fundamentally different.

QCDCL: Assignment Generation + Clause/Cube Learning:
- More powerful than backtracking/backjumping.
- No explicit flipping of decision variables.
- Generation of resolution proofs guided by assignments.
- State-of-the-art approach, crucial implementation details.
- Future work: safely relax the quantifier ordering.
- Future work: integrate preprocessing, certificate generation.

http://lonsing.github.io/depqbf/
M. Cadoli, A. Giovanardi, and M. Schaerf.
An Algorithm to Evaluate Quantified Boolean Formulae.

M. Davis, G. Logemann, and D. Loveland.
A Machine Program for Theorem-proving.

E. Giunchiglia, M. Narizzano, and A. Tacchella.
QUBE: A System for Deciding Quantified Boolean Formulas Satisfiability.

E. Giunchiglia, M. Narizzano, and A. Tacchella.
Clause/Term Resolution and Learning in the Evaluation of Quantified Boolean Formulas.

Efficient Clause Learning for Quantified Boolean Formulas via QBF Pseudo Unit Propagation.

Allen Van Gelder.