Witness function generation from resolution proofs for QBFs

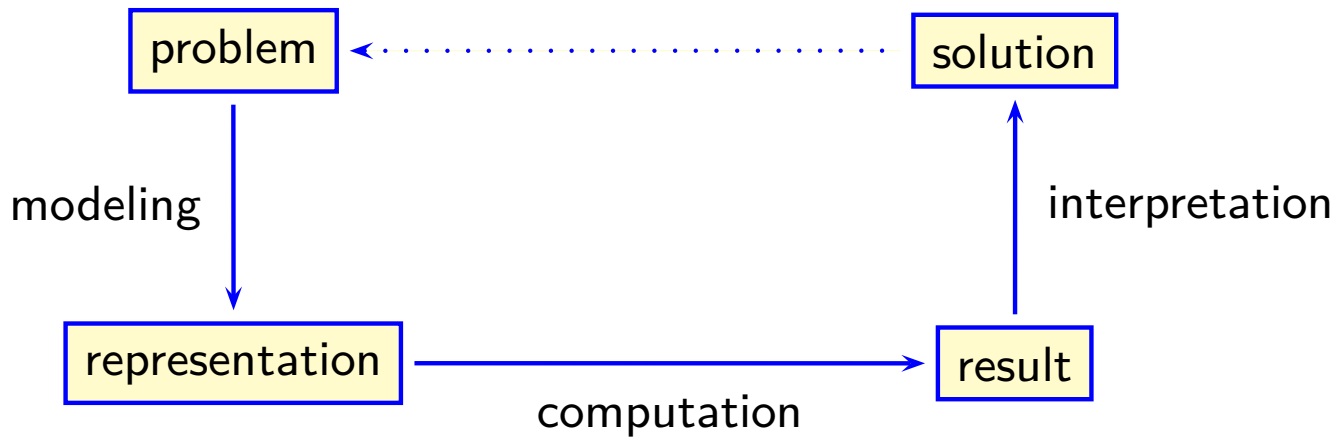
Uwe Egly

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


Supported by the Austrian Science Fund (FWF) under grant S11409-N23 (“RiSE”)

The lazy programmer's approach

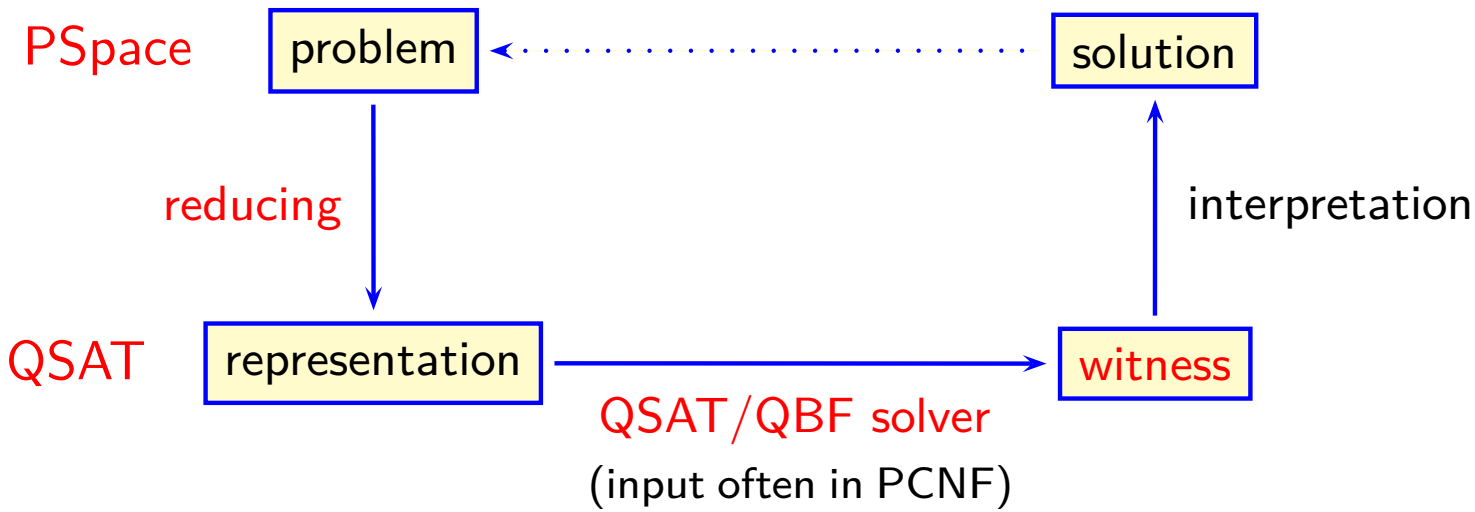
Allows us to implement problem solving programs rapidly



We want to model a problem by compiling it into a suitable representation s.t. the result of the compiled problem can be interpreted as a solution to the original problem.

The lazy programmer's approach

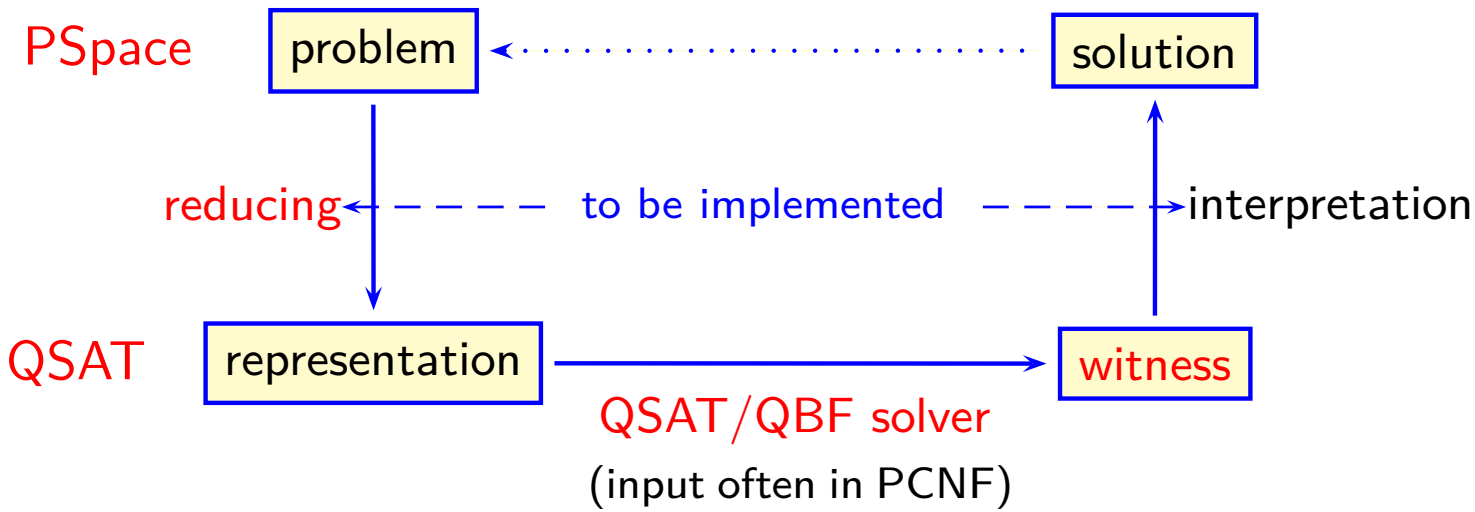
Allows us to implement problem solving programs rapidly



We want to model a problem by compiling it into a suitable representation s.t. the result of the compiled problem can be interpreted as a solution to the original problem.

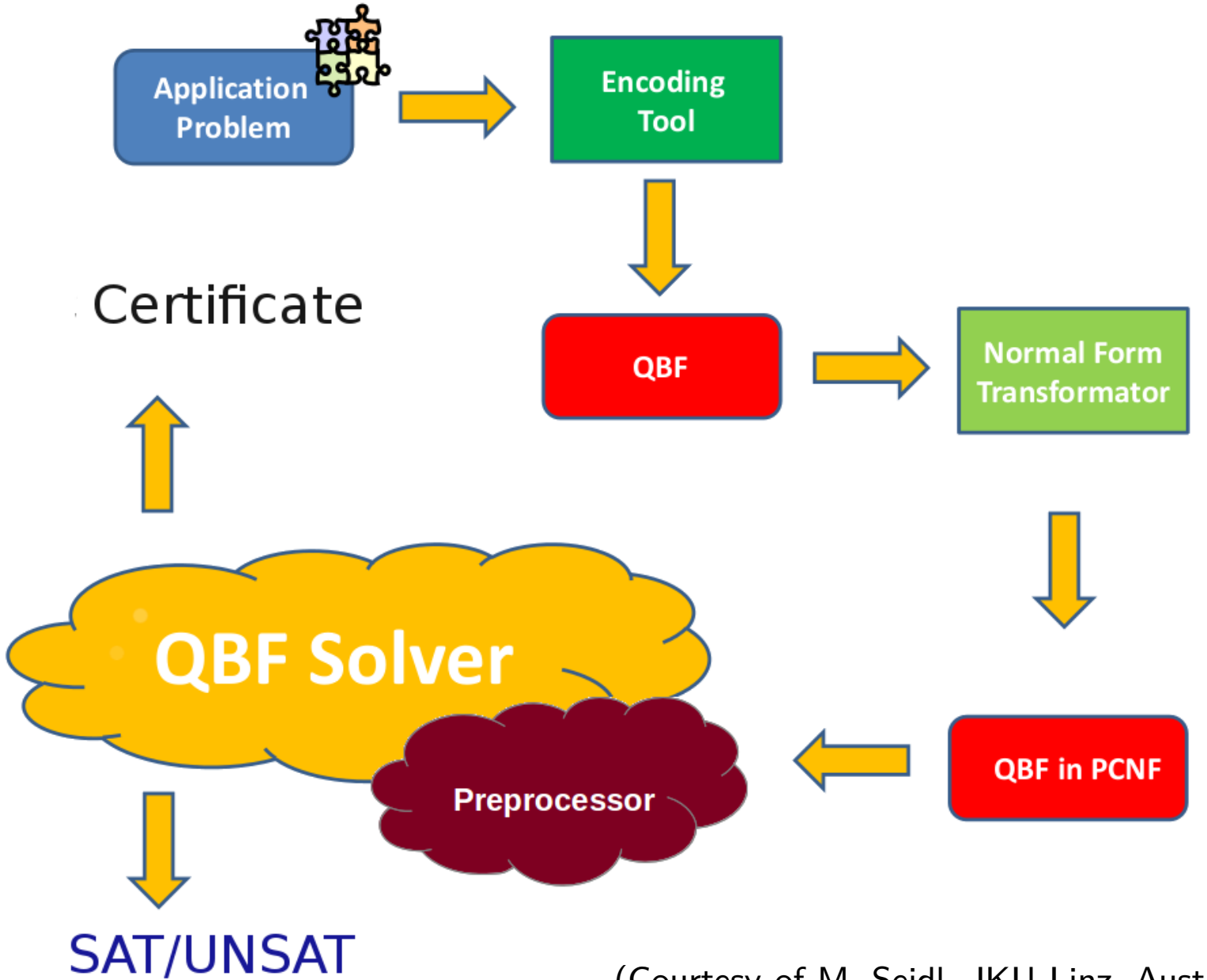
The lazy programmer's approach

Allows us to implement problem solving programs rapidly



We want to model a problem by compiling it into a suitable representation s.t. the result of the compiled problem can be interpreted as a solution to the original problem.

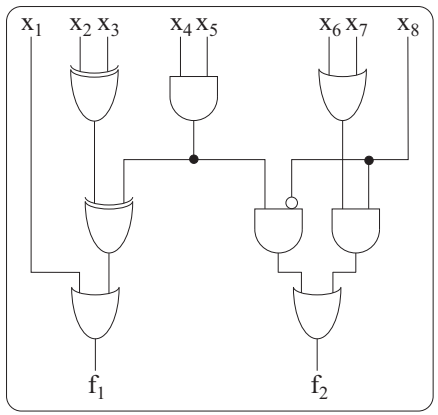
The work-flow



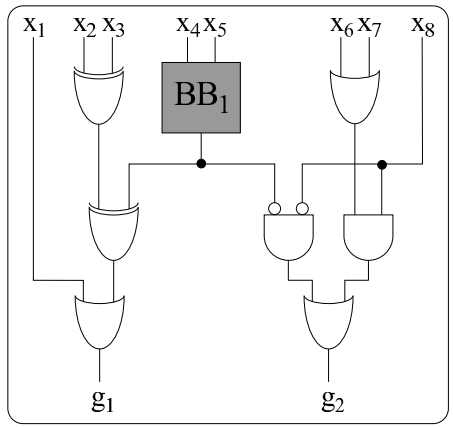
(Courtesy of M. Seidl, JKU Linz, Austria)

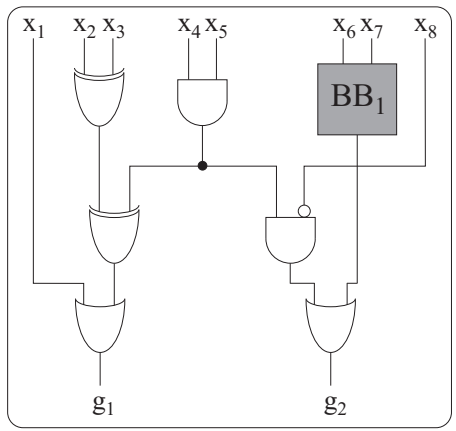
An example application

specification



partial implementations





Are the partial implementations on a good way, i.e.:

- can they be completed to an implementation satisfying the spec?

➥ The questions can be answered using a QBF solver!

➥ If the answer is yes, the black-box circuit can be synthesized as a witness function from a Q-resolution proof constructed by the solver!

Example taken from C. Scholl, B. Becker: Checking Equivalence for Partial Implementations, Proc. DAC, pp. 238-243, 2001.

Outline

PCNF: The input format for solvers

Prenex normal form (PNF), prefix, matrix, PCNF, closed

In the following, $Q_i \in \{\forall, \exists\}$ and p_i is a propositional variable.

Let $\Phi = Q_1 p_1 \ldots Q_n p_n \, \psi$ be a QBF.

- Φ is in prenex (normal) form (PNF) if ψ is purely propositional
- $Q_1 p_1 Q_2 p_2 \cdots Q_n p_n$ is the prefix of Φ; ψ is the matrix of Φ.
- Φ is in PCNF if Φ is in PNF and ψ is in CNF
- A CNF is a conjunction of clauses (= disjunctions of literals)
- Φ is closed if the variables in ψ are in $\{p_1, \ldots, p_n\}$

The semantics of QBFs

- Based on interpretations I represented as sets of atoms
- An atom p is true under I iff $p \in I$

Inductive definition of the truth value, $\nu_I(\Phi)$, of a QBF Φ under an interpretation I:

1. if $\Phi = \top$, then $\nu_I(\Phi) = 1$;
2. if $\Phi = p \in \mathcal{P}$, then $\nu_I(\Phi) = 1$ if $p \in I$, and $\nu_I(\Phi) = 0$ otherwise;
3. if $\Phi = \neg\Psi$, then $\nu_I(\Phi) = 1 - \nu_I(\Psi)$;
4. if $\Phi = (\Phi_1 \wedge \Phi_2)$, then $\nu_I(\Phi) = min(\{\nu_I(\Phi_1), \nu_I(\Phi_2)\})$;
5. if $\Phi = \forall p\, \Psi$, then $\nu_I(\Phi) = \nu_I(\Psi[p/\top] \ \wedge \ \Psi[p/\bot])$;
6. if $\Phi = \exists p\, \Psi$, then $\nu_I(\Phi) = \nu_I(\Psi[p/\top] \ \vee \ \Psi[p/\bot])$.

Truth conditions for $\bot$, $\vee$, $\rightarrow$, $\leftrightarrow$ follow from the above "as usual"

The semantics of QBFs (cont'd)

Notations

- Φ is true under I iff $\nu_I(\Phi) = 1$, otherwise Φ is false under I
- If $\nu_I(\Phi) = 1$, then I is a model of Φ (and Φ is satisfiable)
- If Φ is true under any interpretation, then Φ is valid
- Two sets of QBFs (or ordinary Boolean formulas) are logically equivalent iff they possess the same models

Observations

- A closed QBF is either valid or unsatisfiable, because it is either true under each interpretation I or false under each I.
- Hence, for closed QBFs, there is no need to refer to particular interpretations.

Certificates for QBFs

A certificate provides evidence of (un)satisfiability of a QBF

- One possibility to certify the truth of a closed QBF:
  Witness functions/formulas (WFs) for existential quantifiers which depend on (some) dominating universal quantifiers

  Example: $\forall x_1 \forall x_2 \exists y \, (x_1 \vee x_2 \vee \neg y) \wedge (\neg x_1 \vee y)$

☞ Take $y = x_1$: $\forall x_1 \forall x_2 \, (x_1 \vee x_2 \vee \neg x_1) \wedge (\neg x_1 \vee x_1)$ becomes true

☞ This can be checked with a validity checker for propositional logic

- WFs are sometimes the constructed solution to a problem

For a broader discussion, see V. Balabanov, J.-H. R. Jiang: Resolution proofs and Skolem functions in QBF evaluation and applications. CAV 2011. [link]

Outline

Resolution for QBF

Universal reduction (UR)

Definition (Universal reduction (UR))

Given a clause C, UR on C produces the clause

$$UR(C) := C \setminus \{\ell \in L_\forall(C) \mid \forall \ell' \in L_\exists(C) : var(\ell') < var(\ell)\},$$

where

1. $L_\forall(C)$ denotes the set of all universal literals in C,
2. $L_\exists(C)$ denotes the set of all existential literals in C, and
3. $var(\ell') < var(\ell)\}$ iff $var(\ell')$ occurs left of $var(\ell)\}$ in the quantifier prefix.

Examples for universal reductions

$$\Phi: \quad \exists a \forall x \exists b \left[\underbrace{(a \vee x)}_{C} \wedge \underbrace{(a \vee \neg x \vee b)}_{D} \wedge \psi(a, x, b) \right]$$

- Applying UR to C results the clause a because there is no $\exists$ literal $\ell \in C$, s.t. $\exists var(\ell)$ occurs behind $\forall x$
  ☞ x is tailing in C and can be deleted by UR
- Applying UR to D has no effect, because $\exists b$ (with b a literal in D) occurs after $\forall x$ and blocks the deletion of $\neg x$.

Resolution for QBF

Q-resolvent

Definition (Q-resolvent)

- Let C_1, C_2 be non-tautological clauses with $v \in C_1, \neg v \in C_2$ for an existential pivot variable v.
- *Tentative Q-resolvent* of C_1 and C_2:

$$C_1 \otimes C_2 := (UR(C_1) \cup UR(C_2)) \setminus \{v, \neg v\}.$$

- If $\{x, \neg x\} \subseteq C_1 \otimes C_2$ for some variable x, then no Q-resolvent exists.
- Otherwise, the *Q-resolvent* of C_1 and C_2 is

$$C := QRES(C_1, C_2) := UR(C_1 \otimes C_2).$$

Example of Q-resolvents

$$\Phi: \quad \exists a \forall x \exists b \forall y \exists c \left[\underbrace{(a \vee x \vee b)}_{C} \wedge \underbrace{(a \vee \neg b \vee y \vee c)}_{D} \wedge \underbrace{(\neg x \vee \neg b \vee c)}_{E} \right]$$

Is there a Q-resolvent between C and D?

- Only possibility: Use the $\exists$ variable b as a pivot variable
- Construct $C \otimes D = (UR(C) \cup UR(D)) \setminus \{b, \overline{b}\} = a \vee x \vee y \vee c$
- This is also the Q-resolvent
  - ➥ No complementary literals in $C \otimes D$ and UR-reduced

Is there a Q-resolvent between C and E?

- Again, the only possibility for the pivot variable is the $\exists$ variable b
- Construct $C \otimes E = (UR(C) \cup UR(E)) \setminus \{b, \overline{b}\} = a \vee x \vee \neg x \vee c$
- Since $\{x, \overline{x}\} \subseteq C \otimes E$, no Q-resolvent exists

Q-resolution: Derivation and refutation

Γ: $C_1, C_2, \ldots, C_n$ is called a QRES derivation of a clause C from a closed PCNF formula Φ if $C_n = C$ and, for all $i = 1, \ldots, n$,

1. C_i is a clause from the matrix of Φ or
2. C_i is a Q-resolvent of C_k and C_l with $k, l < i$.

Γ is called a QRES refutation of Φ if C is the empty clause $\square$.

Theorem (Kleine Büning et al., 1995)

A closed QBF Φ in PCNF is false iff there is a QRES refutation of Φ.

Remark

A similar procedure is possible for true closed QBFs in prenex disjunctive normal form. The corresponding resolution concept is called term resolution.

Where do we get the Q-refutations from?

At least 2 possible answers to this question:

1. From a solver based on Q-resolution
2. From a solver based on clause learning (often dubbed CDCL solver)

Propaganda

If you are looking for a QBF solver, try depQBF of F. Lonsing. It is available at

`http://lonsing.github.com/depqbf/`

Outline

Certificates

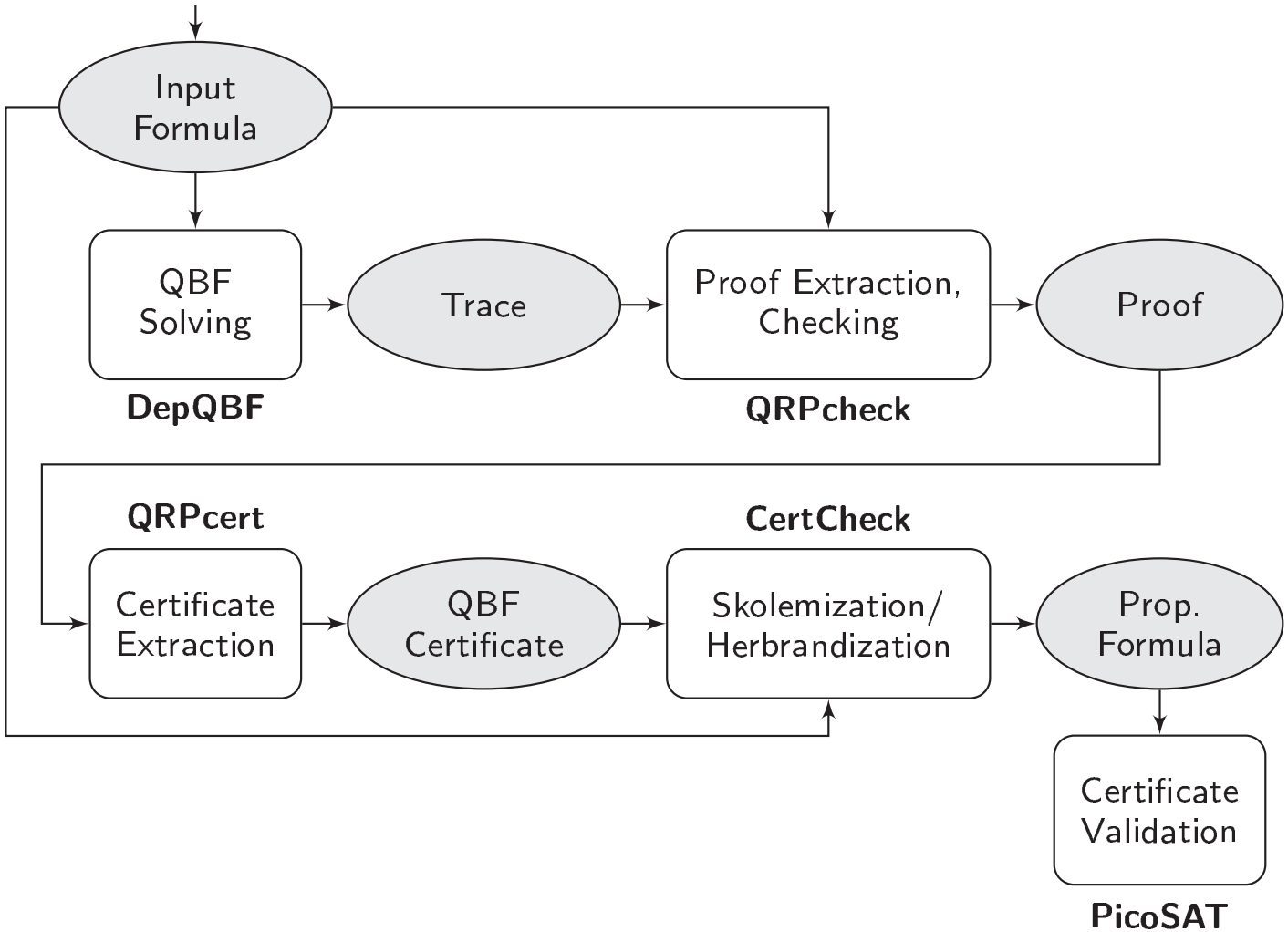
Certificates are required ...

- to verify the correctness of the result provided by a QBF solver
- as a concrete solution to a given (synthesis) problem like, e.g.,
  - a winning strategy in a game-oriented problem formalization,
  - a counter example to some correctness statement or
  - or simply the solution of a synthesis problem like before

☞ Here: Certificates are (sets of) Boolean functions

☞ They can be generated from cube or resolution proofs
(For the details, see V. Balabanov, J.-H. R. Jiang op. cit.)

The work-flow for certificate extraction



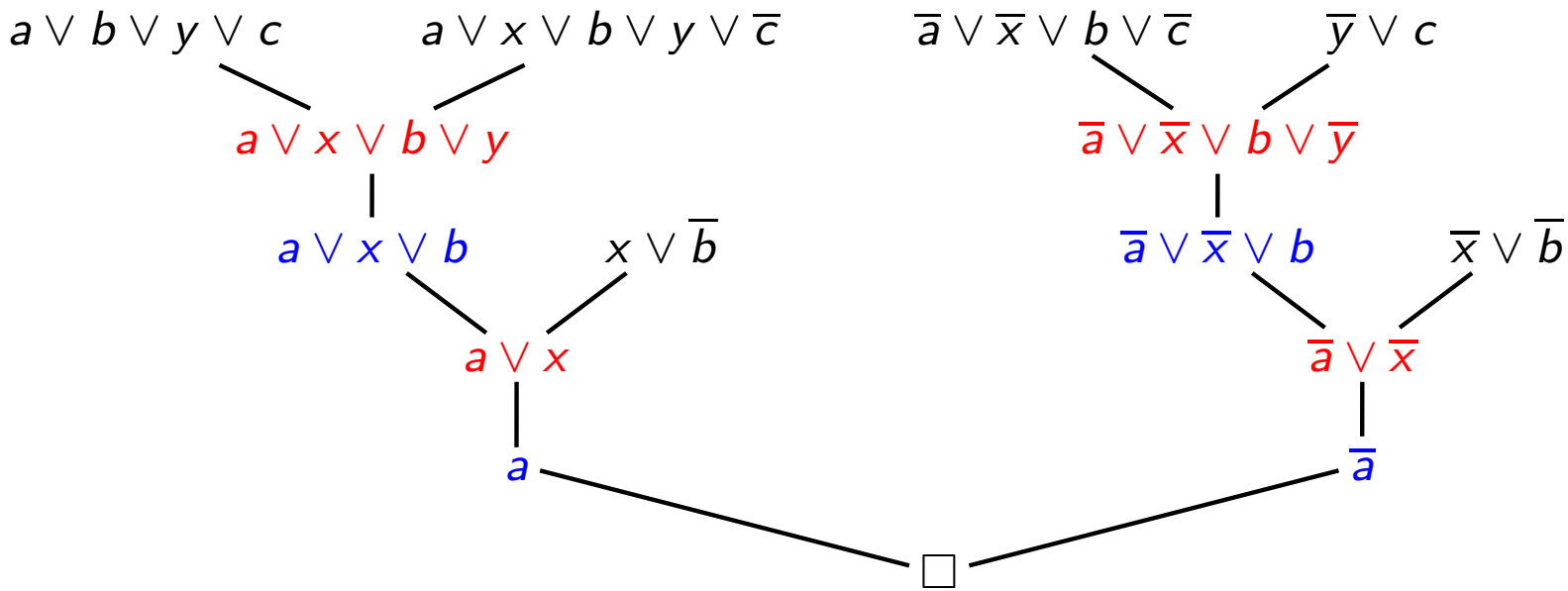
For further details, see Niemetz et al., Proc. SAT 2012

(Courtesy of M. Seidl, JKU Linz, Austria)

Certificate extraction: An example

Given a PCNF with the prefix $\exists a \forall x \exists b \forall y \exists c$ and the clauses

$(a \vee b \vee y \vee c)$ $(a \vee x \vee b \vee y \vee \neg c)$ $(x \vee \neg b)$ $(\neg y \vee c)$ $(\neg a \vee \neg x \vee b \vee \neg c)$ $(\neg x \vee \neg b)$



Basic idea: Map universal variables to the "rest of the clause"
$f_x(a) = a$ and $f_y(a, b) = (a \vee b) \wedge (a \wedge \neg b)$. Details will follow.

We use $\overline{\ell}$ instead of $\neg \ell$ for space reasons.

The structure of the functions

We use formulas with a single hole (indicated by $\bullet$) in the construction

$$\mathit{fwh} ::= \bullet \mid \mathit{clause} \wedge (\mathit{fwh}) \mid \mathit{cube} \vee (\mathit{fwh})$$

clause means a disjunction, *cube* means a conjunction of literals

We construct deeper and deeper nested formulas by replacing $\bullet$ by more complicated formulas containing a single dot. For example:

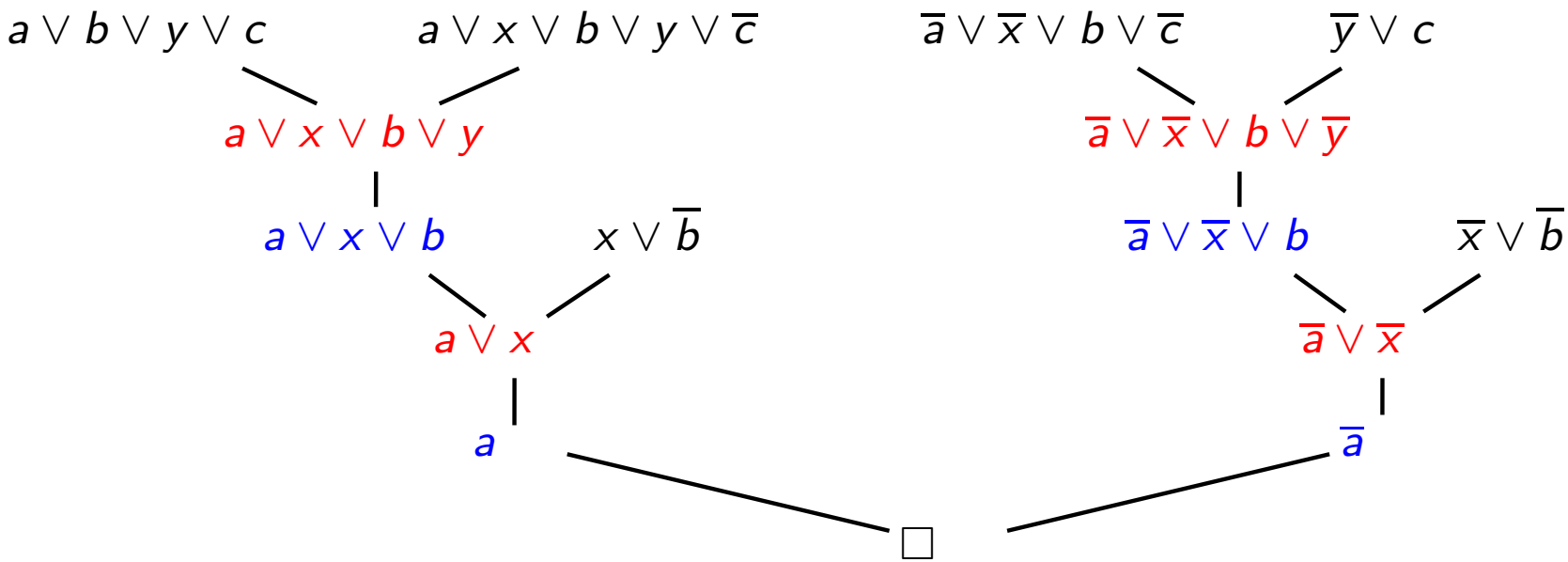
$$\bullet, \quad (c \vee d) \wedge (\bullet), \quad (c \vee d) \wedge ((e \vee f) \wedge (\bullet))$$

Eventually, formulas with $\bullet$ are reduced to formulas:

$$\mathit{red}(F) = \begin{cases} G(\mathit{Cube}) & \text{if } F = G(\mathit{Cube} \vee (\bullet)) \ ; \\ G(\mathit{Clause}) & \text{if } F = G(\mathit{Clause} \wedge (\bullet)) \ . \end{cases}$$

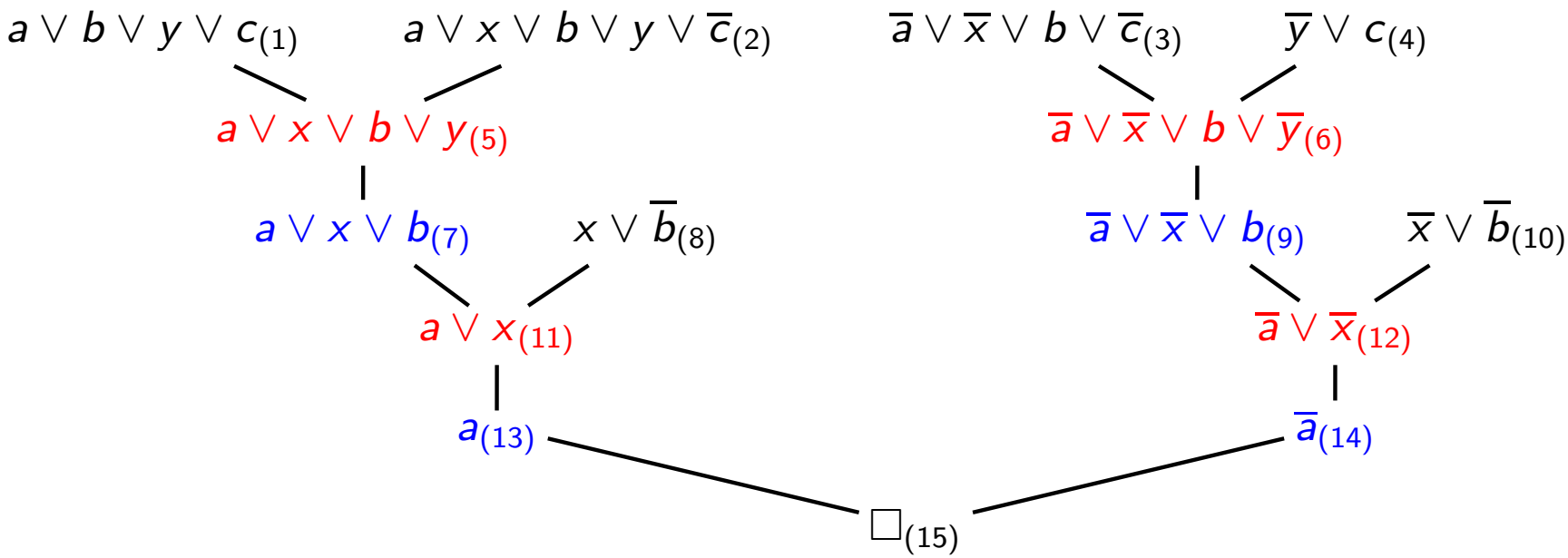
Certificate extraction: The details

$\exists a \forall x \exists b \forall y \exists c$



Certificate extraction: The details

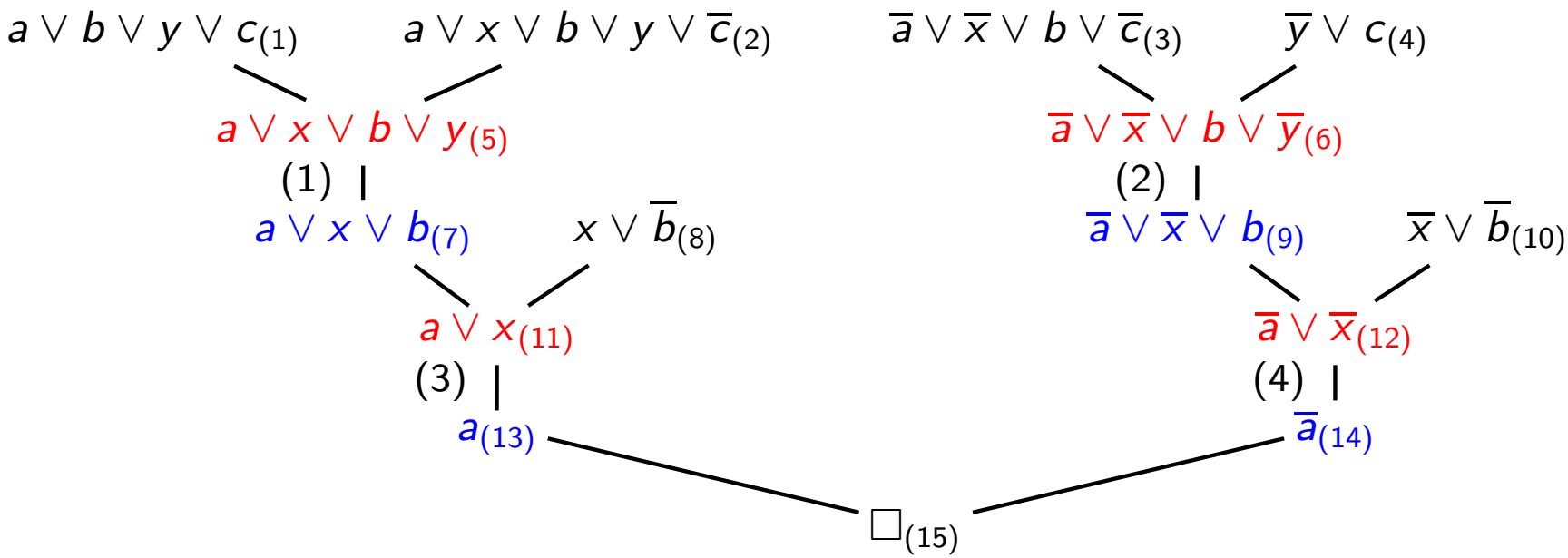
$\exists a \forall x \exists b \forall y \exists c$



Fix topological ordering on vertices s.t. premises are smaller than conclusions

Certificate extraction: The details

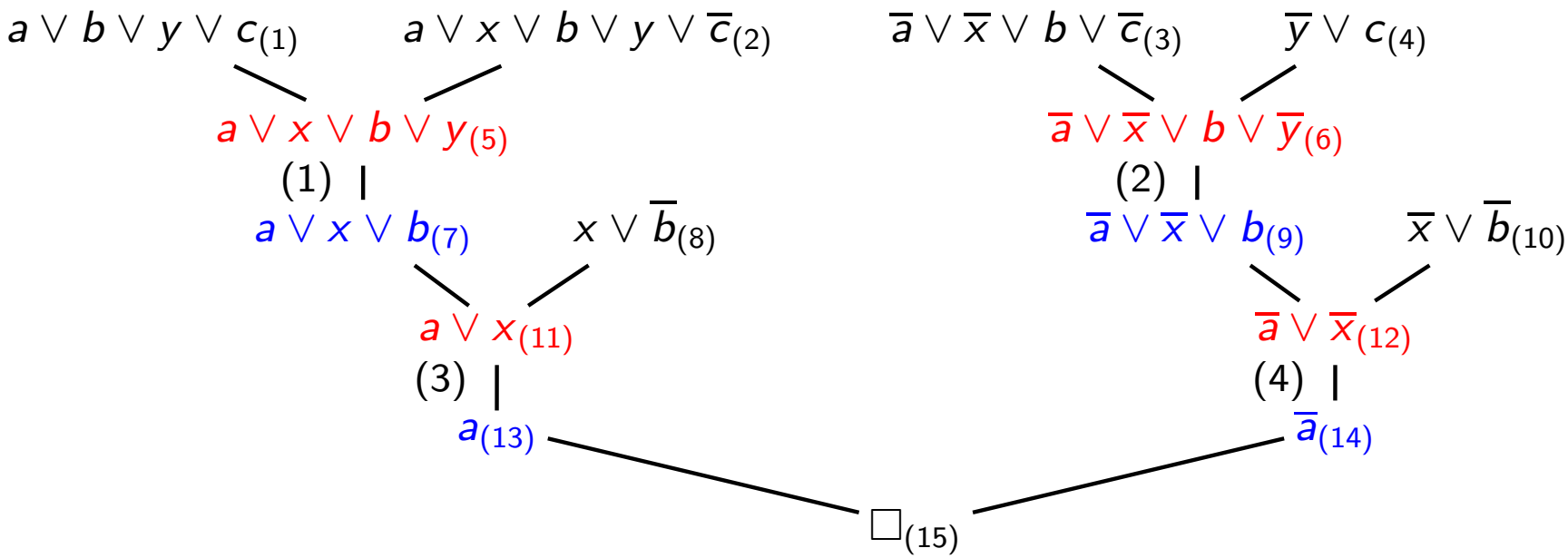
$\exists a \forall x \exists b \forall y \exists c$



Fix ordering on UR proof steps compatible with the chosen topological ordering

Certificate extraction: The details

$\exists a \forall x \exists b \forall y \exists c$

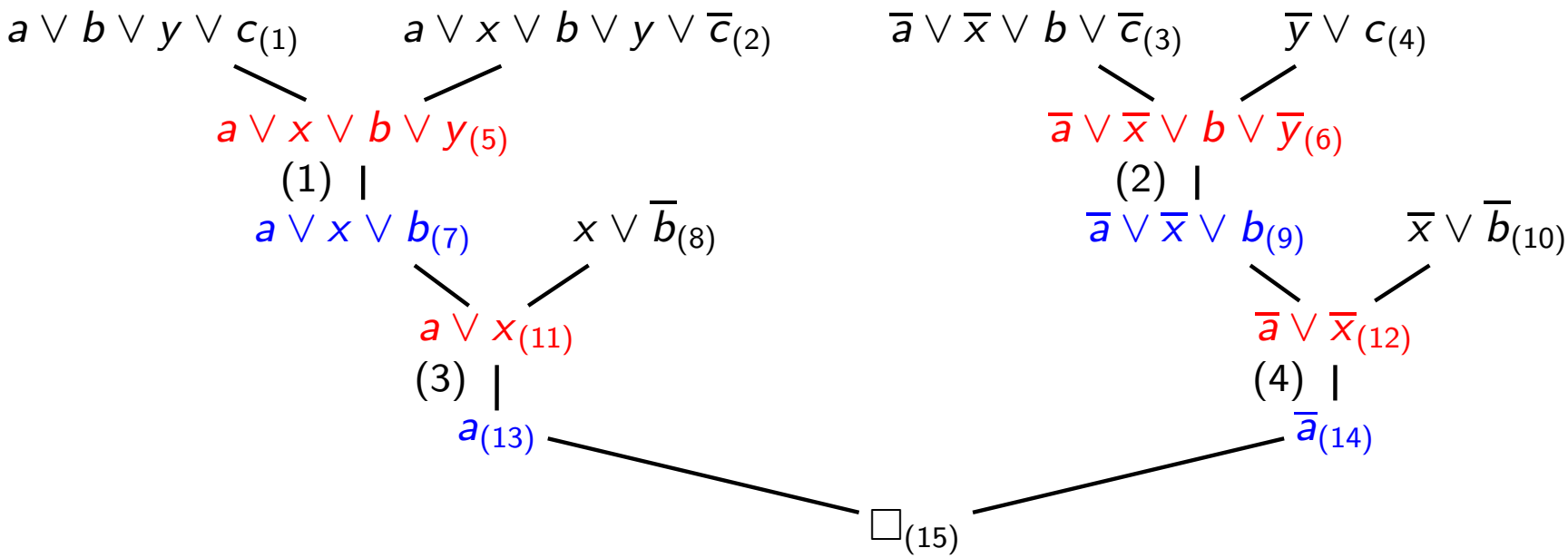


If $k_1 \vee \cdots \vee k_m \vee \ell_1 \vee \cdots \vee \ell_n$ is the premise of UR and $k_1 \vee \cdots \vee k_m$ the conclusion, define

- $\mathit{Clause} = k_1 \vee \cdots \vee k_m$, if ℓ is atomic.
- $\mathit{Cube} = k_1^c \wedge \cdots \wedge k_m^c$ if ℓ is $\neg x$. k^c is a if $k = \neg a$ and $\neg k$ otherwise.

Certificate extraction: The details

$\exists a \forall x \exists b \forall y \exists c$



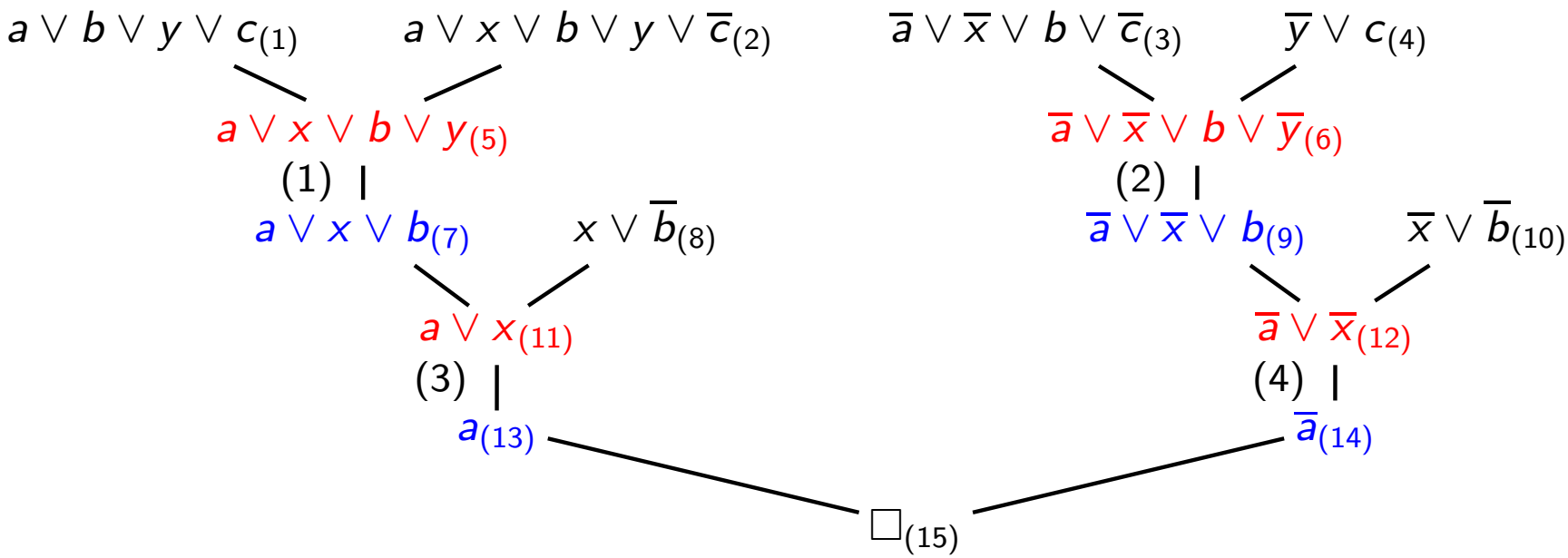
Take $f'_y = \bullet$ and compute

(1) $f'_y = (a \vee x \vee b) \wedge (\bullet)$

(2) $f'_y = (a \vee x \vee b) \wedge ((a \wedge x \wedge \neg b) \vee (\bullet))$

Certificate extraction: The details

$\exists a \forall x \exists b \forall y \exists c$



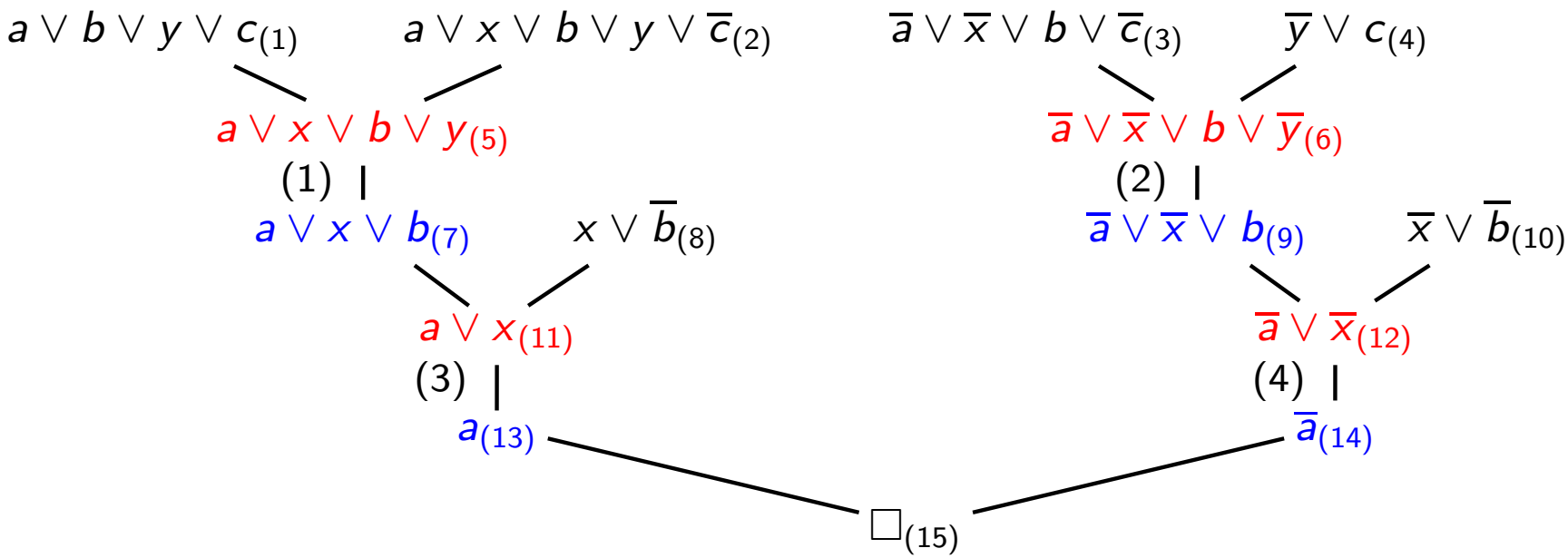
Take $f'_x = \bullet$ and compute

(3) $f'_x = a \wedge (\bullet)$

(4) $f'_x = a \wedge (a \vee (\bullet))$

Certificate extraction: The details

$\exists a \forall x \exists b \forall y \exists c$



Finally, we get rid of $\bullet$ and simplify

- $f_x = a \wedge a$ simplifies to a
- $f_y = (a \vee x \vee b) \wedge (a \wedge x \wedge \neg b)$ simplifies to $(a \vee b) \wedge (a \wedge \neg b)$

Conclusion

Some open problems

- Certificates are often too large
  - Q-resolution calculus is "weak" (wrt the generation of succinct proofs)
  - The clause learning component in QBF solvers produce "verbose" proofs
    ☞ Sometimes exponential size proofs although poly-size proofs exist

➥ Strengthen the Q-resolution calculus (e.g., by long distance res)

➥ Improve the clause learning component

- Integration of preprocessing activities into Q-resolution proofs
- Improving the handling of true QBFs

Generating PCNFs

Why are formulas in PCNF necessary?

- Most QBF solvers require the input being in PCNF

☞ Translation procedure required

- This procedure can be based on distributivity or Tseitin

The translation procedure:

- Start by eliminating equivalences:
  replace $\varphi \leftrightarrow \psi$ by $\varphi \rightarrow \psi \wedge \psi \rightarrow \varphi$
- Remove $\rightarrow$, i.e., replace $\varphi \rightarrow \psi$ by $\neg\varphi \vee \psi$
- Push $\neg$ inward, e.g., replace $\neg(\varphi \wedge \psi)$ by $\neg\varphi \vee \neg\psi$

☞ Results in a formula in negation normal form (NNF)

Generating PCNFs (cont'd)

The Tseitin-based algorithm works on a NNF in three steps

1. Generate a prenex form $\Psi_p \colon Q_i X_i \cdots Q_k X_k \, \psi$ of the input QBF Ψ with a minimal number of quantifier alternations. Then the matrix ψ is purely propositional.
2. Use Tseitin's translation to transform ψ into 3-CNF.
3. Place the $\exists$ quantifiers for the newly introduced variables $\ell_1, \cdots \ell_m$ abbreviating $\varphi_1, \ldots, \varphi_m$ "correctly", e.g.,
   - place all the new $\exists$ at the end of the quantifier prefix, or
   - place $\exists \ell_i$ $(1 \leq i \leq m)$ after all quantifiers of those variables which occur in φ_i.

☞ The use of quantifiers results in an equivalent CNF

Pros and cons of PCNFs

Advantages

- Simple input format (QDIMACS)
- Simpler data structures and algorithms

Possible disadvantages

- The formula structure may be disrupted
- Additional variables are introduced
- The size of the PCNF is potentially increased
- The transformation (esp. prenexing) is non-deterministic

Conflict-driven clause learning (CDCL) (for PCNF))

- Detect, record and exploit information unveiled during search
- Goal: reduction of useless explorations of the search space
- CDCL results in a drastically performance improvement
- Side effect: Info gained during clause learning can be used to construct a resolution proof (as a witness for unsatisfiability)
- Extensions allow cube learning and cube resolution proofs

Example (Propaganda)

If you are looking for a QBF solver, try depQBF of F. Lonsing. It is available at

`http://lonsing.github.com/depqbf/`

The qdpll procedure

qdpll ()

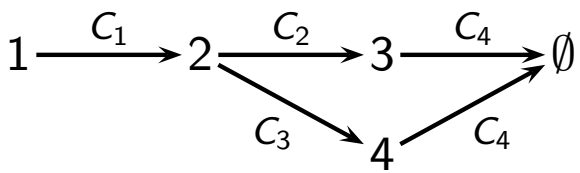
```
while (true) do
    val = simplify ();
    ;                                          /* select next variable */
    if val == UNDEF then
        v = select_next_branch_var ();
        assign_branch_var (v);
    else
        ;                                      /* solution / conflict found */
        btlevel = analyze_leaf ();
        if btlevel == INVALID then
            return val;
        else
            backtrack (btlevel);
```

Conflict-Driven Clause Learning (CDCL)

Example (Learn clause starting from current falsified clause)

```
p cnf 5 4
e 1 3 4 0
a 5 0
e 2 0
-1 2 0
3 5 -2 0
4 -5 -2 0
-3 -4 0
```

Implication graph
(clause (-3 -4) is falsified)



Derive the learned clause (-1)