Backjumping for Quantified Boolean Logic Satisfiability ⋆

Enrico Giunchiglia, Massimo Narizzano, Armando Tacchella

DIST - Università di Genova
Viale Causa 13, 16145 Genova, Italy

Abstract

The implementation of effective reasoning tools for deciding the satisfiability of Quantified Boolean Formulas (QBFs) is an important research issue in Artificial Intelligence. Many decision procedures have been proposed in the last few years, most of them based on the Davis, Logemann, Loveland procedure (DLL) for propositional satisfiability (SAT). In this paper we show how it is possible to extend the conflict-directed backjumping schema for SAT to the satisfiability of QBFs: When applicable, conflict-directed backjumping allows search to skip over existentially quantified literals while backtracking. We introduce solution-directed backjumping, which allows the same behavior for universally quantified literals. We show how it is possible to incorporate both conflict-directed and solution-directed backjumping in a DLL-based decision procedure for satisfiability of QBFs. We also implement and test the procedure: The experimental analysis shows that, because of backjumping, significant speed-ups can be obtained.

Summing up: We present the first algorithm that applies conflict and solution directed backjumping to QBF, and demonstrate the performance of this algorithm via an empirical study.

Key words: Quantified Boolean Logic, satisfiability testing, automated reasoning

1 Introduction

The implementation of effective reasoning tools for deciding the satisfiability of Quantified Boolean Formulas (QBFs) is an important research issue in

⋆ This paper builds on and extends the work presented in [1]. Work partially supported by ASI (Agenzia Spaziale Italiana).

Email address: {`enrico,mox,tac`}`@mrg.dist.unige.it` (Enrico Giunchiglia, Massimo Narizzano, Armando Tacchella).

Artificial Intelligence. Many reasoning tasks involving abduction, reasoning about knowledge, non monotonic reasoning, are PSPACE-complete reasoning problems and are reducible in polynomial time to the problem of determining the satisfaction of a QBF. More important, since QBF reasoning is the prototypical PSPACE problem, many of these reductions are readily available. For these reasons, we have seen in the last few years the presentation of several implemented decision procedures for satisfiability of QBFs, like QKN [2], Evaluate [3,4], decide [5], quip [6], Qsolve [7]. Most of the above decision procedures are based on the Davis, Logemann, Loveland procedure (DLL) for propositional satisfiability [8] (SAT). This is because it is rather easy to extend DLL to deal with satisfiability of QBFs, and this opens up the possibility to extend the reasoning techniques developed for DLL-based SAT solvers to DLL-based QBF solvers. Among these techniques, conflict-directed backjumping is one of the most important (see, e.g., [9–11]).

In this paper we show how it is possible to extend the conflict-directed backjumping schema for SAT to the satisfiability of QBFs: When applicable, it allows to jump over existentially quantified literals while backtracking. We introduce solution-directed backjumping, which allows the same behavior for universally quantified literals. We show how it is possible to incorporate both conflict-directed and solution-directed backjumping in a DLL-based decision procedure for the satisfiability of QBFs. We also implement and test the procedure: The experimental analysis shows that, because of backjumping, significant speed-ups can be obtained. While there have been several proposals for backjumping in SAT, this is the first time –as far as we know– this idea has been proposed, implemented and experimented for the satisfiability of QBFs.

The paper is structured as follows. In Section 2 we introduce some formal preliminaries necessary for the rest of the paper. In Section 3 we introduce QuBE, a DLL-based decision procedure for satisfiability of QBFs. QuBE is developed as a testbed for the algorithms herewith presented and, as such, QuBE features both a standard backtracking procedure (presented in Section 3) and the backjumping procedure subject of this paper (presented in Section 5). The description and the theoretical results at the basis of the backjumping procedure are to be found in Section 4. The experimental analysis is reported in Section 6. We end the paper with the conclusions and future work in Section 7.

2 Formal preliminaries

Consider a set P of propositional letters. An *atom* is an element of P. A *literal* is an atom or the negation of an atom. In the following, for any literal l,

- $|l|$ is the atom occurring in l; and
- $\bar{l}$ is $\neg l$ if l is an atom, and is $|l|$ otherwise.

A *clause* C is an n-ary ($n \geq 0$) disjunction of literals such that, for any two distinct disjuncts l, l' in C, it is not the case that $|l| = |l'|$. A *propositional formula* is a k-ary ($k \geq 0$) conjunction of clauses. As customary in SAT, we represent a clause as a set of literals, and a propositional formula as a set of clauses. With this notation, e.g.,

- The clause $\{\}$ is the *empty clause* and stands for the empty disjunction.
- The propositional formula $\{\}$ is the *empty set of clauses* and stands for the empty conjunction.
- The propositional formula $\{\{\}\}$ stands for the set of clauses whose only element is the empty clause.

A *QBF* is an expression of the form

$$Q_1 z_1 \ldots Q_n z_n \Phi \qquad (n \geq 0) \tag{1}$$

where

- every Q_i ($1 \leq i \leq n$) is a quantifier, either existential $\exists$ or universal $\forall$,
- $z_1, \ldots, z_n$ are distinct atoms in P, and
- Φ is a propositional formula in the atoms $z_1, \ldots, z_n$.

In (1), $Q_1 z_1 \ldots Q_n z_n$ is the *prefix*, Φ is the *matrix*, and Q_i is the *bounding quantifier* of z_i. Further, we say that a literal l is *existential* if $\exists |l|$ belongs to the prefix, and is *universal* otherwise. In the following, we use $x_1, x_2, \ldots$ for existential atoms; $y_1, y_2, \ldots$ for universal atoms; and $z_1, z_2, \ldots$ for arbitrarily bounded atoms. Thus, for example, the expression

$$\begin{array}{c} \forall y_1 \exists x_1 \forall y_2 \exists x_2 \exists x_3 \\ \{\{y_1, y_2, x_2\}, \{y_1, \neg y_2, x_2, \neg x_3\}, \{y_1, \neg x_2, x_3\}, \\ \{\neg y_1, x_1, x_3\}, \{\neg y_1, y_2, x_2\}, \{\neg y_1, y_2, \neg x_2\}, \{\neg y_1, \neg x_1, \neg y_2, \neg x_3\}\} \end{array} \tag{2}$$

is a QBF whose prefix is $\forall y_1 \exists x_1 \forall y_2 \exists x_2 \exists x_3$ and whose matrix has 7 clauses.

The semantics of a QBF φ can be defined recursively as follows:

(1) If φ contains an empty clause then φ is FALSE.
(2) If the matrix of φ is the empty set of clauses then φ is TRUE.
(3) If φ is $\exists x \psi$, φ is TRUE if and only if φ_x or $\varphi_{\neg x}$ are TRUE.
(4) If φ is $\forall y \psi$, φ is TRUE if and only if φ_y and $\varphi_{\neg y}$ are TRUE.

If φ is a QBF and l is a literal, φ_l is the QBF

(1) whose matrix Φ is obtained from the matrix of φ by deleting the clauses C such that $l \in C$, and removing $\overline{l}$ from the others, and
(2) whose prefix is obtained from the prefix of φ by deleting each atom and corresponding bounding quantifier not occurring in Φ.

For example, if φ is (2) then $\varphi_{\neg y_1}$ is

$$\forall y_2 \exists x_2 \exists x_3 \{\{y_2, x_2\}, \{\neg y_2, x_2, \neg x_3\}, \{\neg x_2, x_3\}\}.$$

We say that a QBF φ is *satisfiable* iff φ is TRUE. For example, the QBF

$$\forall y \exists x \{\{\neg y, x\}, \{y, \neg x\}\}$$

is satisfiable, while the QBF

$$\exists x \forall y \{\{\neg y, x\}, \{y, \neg x\}\}$$

is not satisfiable.

As the above two examples show, the satisfiability of a QBF (1) may depend on the order in which the expressions $Q_i z_i$ are listed in the prefix. A simple recursive procedure for determining the satisfiability of a QBF φ, simplifies φ to φ_z and/or $\varphi_{\neg z}$ if z is the leftmost atom in the prefix, till either an empty clause or the empty set of clauses are produced: On the basis of the satisfiability of φ_z and $\varphi_{\neg z}$, the satisfiability of φ can be determined according to the semantics of QBFs.

There are some simple improvements to this basic procedure.

Let φ be a QBF (1). Consider φ.

The first improvement is that we can directly conclude that φ is unsatisfiable if the matrix of φ contains a contradictory clause. A clause C is *contradictory* if it contains no existential literal. An example of a contradictory clause is the empty clause.

The second improvement is based on the fact that in a QBF we can swap two atoms in the prefix if they have the same level. In (1), the *level of an atom* z_i is 1 + the number of expressions $Q_j z_j Q_{j+1} z_{j+1}$ in the prefix with $j \geq i$ and $Q_j \neq Q_{j+1}$. For example, in (2) the level of y_1, x_2, x_3 is 4, 1, 1 respectively. Thus, assuming that z_i and z_1 have the same level in (1), (1) is logically equivalent to

$$Q_i z_i Q_2 z_2 \ldots Q_{i-1} z_{i-1} Q_1 z_1 Q_{i+1} z_{i+1} \ldots Q_n z_n \Phi$$

and we can determine φ satisfiability on the basis of φ_{z_i} and/or $\varphi_{\neg z_i}$. This allows to introduce some heuristics in the choice of the literal for branching.

Finally, if a literal l is unit or monotone in φ, then φ is logically equivalent to φ_l. In (1), a literal l is

- *Unit* if l is existential, and, for some $m \geq 0$,
  · a clause $\{l, l_1, \ldots, l_m\}$ belongs to Φ, and
  · each literal l_i $(1 \leq i \leq m)$ is universal and has a lower level than l. The *level of a literal* l is the level of $|l|$.
  For example, in a QBF of the form
  $$\ldots \exists x_1 \forall y_1 \exists x_2 \ldots \{\{x_1, y_1\}, \{x_2\}, \ldots\},$$
  both x_1 and x_2 are unit.
- *Monotone* or *pure* if
  · either l is existential, $\bar{l}$ does not belong to any clause in Φ, and l occurs in Φ;
  · or l is universal, l does not belong to any clause in Φ, and $\bar{l}$ occurs in Φ.
  For example, in the QBF
  $$\forall y_1 \exists x_1 \forall y_2 \exists x_2 \{\{\neg y_1, y_2, x_2\}, \{x_1, \neg y_2 \neg x_2\}\},$$
  the only monotone literals are y_1 and x_1.

See [3,4] for more details.

3 QuBE and QuBE-BT

QuBE [1] is a system for deciding the satisfiability of QBFs which incorporates the ideas outlined in the previous section. QuBE is implemented in C on top of SIM, an efficient decider for SAT developed by our group [11]. A system description of QuBE is presented in [12]. In this section, we briefly describe QuBE-BT, i.e., QuBE with a standard backtracking procedure, see Figure 1. [2] In Figure 1,

- φ is a global variable initially set to the input QBF.
- *Stack* is a global variable storing the search stack, and is initially empty.
- FALSE, TRUE, UNDEF, NULL, UNIT, PURE, L-SPLIT, R-SPLIT are constants.

[1] QuBE is available at `www.mrg.dist.unige.it/star/qube`.

[2] We use the following pseudocode conventions. Indentation indicates block structure. Two instructions on the same line belong to the same block.":=" is the assignment operator. The constructs **while** ⟨*cond*⟩ ⟨*block*⟩, **do** ⟨*block*⟩ **while** ⟨*cond*⟩, **if**⟨*cond*⟩ ⟨*block*$_1$⟩ **else** ⟨*block*$_2$⟩ have the same interpretation as in the C language.

```
1  φ := ⟨the input QBF⟩;     Stack := ⟨the empty stack⟩;
2  function Simplify()
3   do
4     φ′ := φ;
5     if (⟨a contradictory clause is in φ⟩) return FALSE;
6     if (⟨the matrix of φ is empty⟩) return TRUE;
7     if (⟨l is unit in φ⟩) |l|.mode := UNIT; Extend(l);
8     if (⟨l is monotone in φ⟩) |l|.mode := PURE; Extend(l);
9   while (φ′ ≠ φ);
10  return UNDEF;
11 function Backtrack(res)
12  while (⟨Stack is not empty⟩)
13    l := Retract();
14    if ((res = FALSE and |l|.type = ∃) or (res = TRUE and |l|.type = ∀))
15      if (|l|.mode = L-SPLIT) |l|.mode := R-SPLIT; return l̄;
16  return NULL;
17 function QuBE-BT()
18  do
19    res := Simplify();
20    if (res = UNDEF)   l := ChooseLiteral();
21    else   l := Backtrack(res);
22    if (l ≠ NULL)   Extend(l);
23  while (l ≠ NULL);
24  return res;
```

Fig. 1. The algorithm of QuBE-BT.

- For each atom z in the input QBF,
  - $z.mode$ is a property of z whose possible values are UNIT, PURE, L-SPLIT, R-SPLIT, which have the obvious meaning, and
  - $z.type$ is $\exists$ if z is existential, and $\forall$ otherwise.
- *Extend*(l) pushes l and φ in the stack, and deletes the clauses C of φ such that $l \in C$, and removes $\bar{l}$ from the others.
- *Retract*() pops the literal and corresponding QBF that are on top of the stack: the literal is returned, while the QBF is assigned to φ. (Intuitively, *Retract* is the "inverse" operation of *Extend*).
- *Simplify*() simplifies φ till a contradictory clause is generated (line 5), or the matrix of φ is empty (line 6), or no simplification is possible (lines 4, 9).
- *ChooseLiteral*() returns a literal l in the matrix of φ such that, for each atom z having a greater level than l in the input QBF, z does not occur in the matrix of φ. *ChooseLiteral*() also sets $|l|.mode$ to L-SPLIT.
- *Backtrack*(res): pops all the literals and corresponding QBFs (line 13) from the stack, till a literal l is reached such that (line 14)
  - l is existential and res = FALSE; or

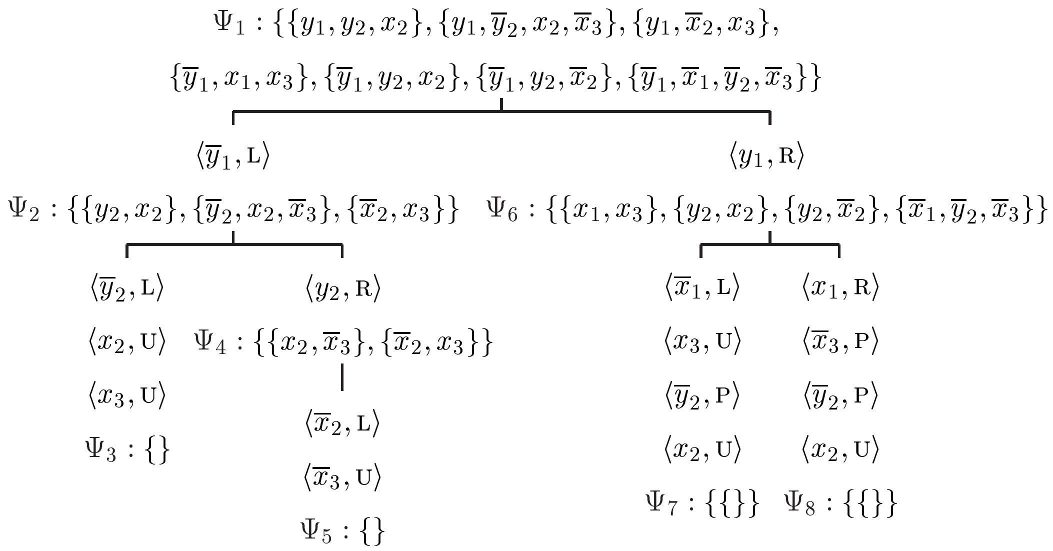

Fig. 2. QuBE-BT computation tree for (2). U, P, L, R stand for UNIT, PURE, L-SPLIT, R-SPLIT respectively. The prefix is $\forall y_1 \exists x_1 \forall y_2 \exists x_2 \exists x_3$.

- · l is universal and res = True.

  If such a literal l exists and $|l|.mode$ is L-SPLIT, then $|l|.mode$ is set to R-SPLIT, and $\overline{l}$ is returned (line 15). Otherwise, Null is returned (line 16).

It is easy to see that QuBE-BT is a generalization of DLL: QuBE-BT and DLL have the same behavior on QBFs without universal quantifiers.

To understand QuBE-BT behavior, consider the QBF (2). For simplicity, assume that *ChooseLiteral* returns the negation of the first atom in the prefix which occurs in the matrix of the QBF under consideration. Then, the tree searched by QuBE-BT when φ is (2) is represented in Figure 2. In Figure 2, each node shows

- the sequence of literals assigned by QuBE-BT before a branch takes place: for each literal l in the sequence, we also show the value of $|l|.mode$; and
- the matrix of the resulting QBF, prefixed by a label.

As the result of the computation, QuBE-BT would correctly return False, i.e., (2) is unsatisfiable.

4 Backjumping

Let φ be a QBF (1). Consider φ.

In the following, if ν is a consistent and finite set $\{l_1, \ldots, l_m\}$ $(m \geq 0)$ of literals, we write φ_ν as an abbreviation for the QBF

- whose matrix is obtained from the matrix of φ by
  - · deleting the clauses C such that $C \cap \nu \neq \{\}$, and
  - · substituting each clause C with $C \setminus \{\bar{l} : l \in \nu\}$,
- and whose prefix is obtained from the prefix of φ by deleting each atom and corresponding bounding quantifier not occurring in the matrix.

Intuitively, φ_ν is the QBF resulting from φ after the literals in ν are assigned. For example, considering Figure 2, if φ is (2), and ν is $\{\neg y_1, y_2\}$, then φ_ν is $\exists x_2 \exists x_3 \Psi_4$.

We extend the above notation to allow for finite sequences $\mu = l_1; \ldots; l_m$ $(m \geq 0)$ of literals: Assuming that $\{l_1, \ldots, l_m\}$ is consistent, φ_μ is defined as $\varphi_{\{l_1,\ldots,l_m\}}$. We also say that a literal l *is in* a sequence $l_1; \ldots; l_m$ $(m \geq 0)$ of literals if $l \in \{l_1, \ldots, l_m\}$. In the following, μ is used to represent an assignment and φ_μ represents the corresponding QBF.

As in constraint satisfaction (see, e.g., [9]) it may be the case that only a subset of the literals in the current assignment is responsible for the result (either TRUE or FALSE) of φ_μ satisfiability. Then, assuming that it is possible to effectively determine such a subset ν, we could avoid doing a right branch on a literal l, if l is not in ν. This process is best known as *backjumping*. To make the notion of backjumping precise we need the following definitions.

A finite sequence $\mu = l_1; \ldots; l_m$ $(m \geq 0)$ of literals such that $|l_i| \neq |l_j|$ $(1 \leq i < j \leq m)$ is an *assignment for* φ if for each literal l_i in μ

- l_i or $\bar{l}_i$ is unit in $\varphi_{l_1;\ldots;l_{i-1}}$; or
- l_i is monotone in $\varphi_{l_1;\ldots;l_{i-1}}$; or
- l_i has the greatest level in $\varphi_{l_1;\ldots;l_{i-1}}$.

We say that an assignment μ *satisfies* φ iff φ_μ is satisfiable. For example, any initial, nonempty subsequence of $\neg y_1; \neg y_2; x_2; x_3$, corresponding to the leftmost branch in Figure 2, is an assignment satisfying (2).

Consider an assignment μ $(m \geq 0)$ for φ. Let ν be a subset of the literals in μ, i.e., $\nu \subseteq \{l : l \text{ is in } \mu\}$.

Intuitively, ν is a reason for φ_μ result if the result of $\varphi_{\nu'}$ satisfiability is the same for each set ν' which agrees with μ on ν.

A consistent set ν' of literals *agrees with μ on ν* if the following three conditions are satisfied:

(1) each atom occurring in ν' does not occur in φ_μ, i.e.,

$$\{|l| : l \in \nu'\} \subseteq \{|l| : |l| \text{ does not occur in } \varphi_\mu\}.$$

(2) If μ satisfies φ, then
- the universal literals in ν are also in ν', i.e.,

$$\{l : l \text{ is universal}, l \in \nu\} \subseteq \nu',$$

- and the negation of the existential literals in ν' are not in μ, i.e.,

$$\{\bar{l} : l \text{ is existential}, l \in \nu'\} \cap \{l : l \text{ is in } \mu\} = \{\}.$$

(3) If μ does not satisfy φ, then
- the existential literals in ν are also in ν', i.e.,

$$\{l : l \text{ is existential}, l \in \nu\} \subseteq \nu',$$

- and the negation of the universal literals in ν' are not in μ, i.e.,

$$\{\bar{l} : l \text{ is universal}, l \in \nu'\} \cap \{l : l \text{ is in } \mu\} = \{\}.$$

Under the simplifying assumption that for each atom z in φ and not in φ_μ, either z or $\neg z$ are in μ, the three conditions correspond to

(1) ν' assigns a subset of the atoms assigned by μ, i.e.,

$$\{|l| : l \in \nu'\} \subseteq \{|l| : |l| \text{ is in } \mu\}.$$

(2) If μ satisfies φ, then
- the universal literals in ν are also in ν', i.e.,

$$\{l : l \text{ is universal}, l \in \nu\} \subseteq \nu',$$

- and the existential literals in ν' are also in μ, i.e.,

$$\{l : l \text{ is existential}, l \in \nu'\} \subseteq \{l : l \text{ is in } \mu\}.$$

(3) If μ does not satisfy φ, then
- the existential literals in ν are also in ν', i.e.,

$$\{l : l \text{ is existential}, l \in \nu\} \subseteq \nu',$$

- and the universal literals in ν' are also in μ, i.e.,

$$\{l : l \text{ is universal}, l \in \nu'\} \subseteq \{l : l \text{ is in } \mu\}.$$

For example, if φ is (2),

- If μ is $\neg y_1; \neg y_2$, and $\nu = \{\neg y_1\}$ then μ satisfies φ and there are 9 sets which agree with μ on ν. They are $\{\neg y_1\}$, $\{\neg y_1, \neg y_2\}$, $\{\neg y_1, y_2\}$, $\{\neg y_1, \neg x_1\}$, $\{\neg y_1, \neg x_1, \neg y_2\}$, $\{\neg y_1, \neg x_1, y_2\}$, $\{\neg y_1, x_1\}$, $\{\neg y_1, x_1, \neg y_2\}$ and $\{\neg y_1, x_1, y_2\}$.
- If μ is $y_1; \neg x_1; x_3$, and $\nu = \{y_1\}$ then μ does not satisfy φ and there are 18 sets which agree with μ on ν. They are $\{\}$, $\{y_1\}$, $\{\neg x_1\}$, $\{x_1\}$, $\{\neg x_3\}$, $\{x_3\}$, $\{y_1, \neg x_1\}$, $\{y_1, x_1\}$, $\{y_1, \neg x_3\}$, $\{y_1, x_3\}$, $\{\neg x_1, \neg x_3\}$, $\{\neg x_1, x_3\}$, $\{x_1, \neg x_3\}$, $\{x_1, x_3\}$, $\{y_1, \neg x_1, \neg x_3\}$, $\{y_1, \neg x_1, x_3\}$, $\{y_1, x_1, \neg x_3\}$ and $\{y_1, x_1, x_3\}$.

The set ν is a *reason for φ_μ result* if for each set ν' which agrees with μ on ν,

$$\varphi_\mu \text{ is satisfiable iff } \varphi_{\nu'} \text{ is satisfiable.}$$

For example, if φ is (2), with reference to Figure 2,

- If μ is $\neg y_1; \neg y_2$, then $\nu = \{\neg y_1\}$ is a reason for φ_μ result: for each of the 9 sets ν' which agrees with μ on ν, $\varphi_{\nu'}$ is satisfiable.
- If μ is $y_1; \neg x_1; x_3$, then $\nu = \{y_1\}$ is a reason for φ_μ result: for each of the 18 sets ν' which agrees with μ on ν, $\varphi_{\nu'}$ is unsatisfiable.

Intuitively, in the backjumping procedure,

- when given an assignment μ such that either the matrix φ_μ is empty or it contains a contradictory clause, we first compute a reason ν for φ_μ result, and
- while backtracking, we dynamically modify the reason ν for the current result. Furthermore, we use ν in order to avoid useless branches: In particular, we avoid doing a right branch on a literal l if l is not in ν.

In the next two subsections we show how to compute such reasons. In particular:

(1) In section 4.1 we show that it is possible to extend the idea of conflict directed backjumping to QBFs satisfiability. Conflict directed backjumping comes to play when an empty clause is generated, and can save a right branch on existential literals. As an example of its potential effectiveness, we show that conflict directed backjumping allows to avoid the exploration leading to Ψ_8 in Figure 2.

(2) In section 4.2 we introduce *solution directed backjumping*. Solution directed backjumping comes to play when the empty set of clauses is generated, and can save a right branch on universal literals. As an example

of its effectiveness, we show that conflict directed backjumping allows to avoid the exploration leading to Ψ_5 in Figure 2.

The logics of conflict and solution directed backjumping are symmetrical, and we could have provided a uniform treatment accounting for both forms of backjumping. However, we prefer to first concentrate on conflict-directed backjumping for QBFs in order to better display the tight relations with the conflict-directed backjumping schema for SAT.

Let ν be a reason for φ_μ result. We say that

- ν is a *reason for φ_μ satisfiability* if φ_μ is satisfiable, and
- ν is a *reason for φ_μ unsatisfiability*, otherwise.

In our example, i.e., if φ is (2), then

- $\{\neg y_1\}$ is a reason for $\varphi_{\neg y_1;\neg y_2}$ satisfiability, and
- $\{y_1\}$ is a reason for $\varphi_{y_1;\neg x_1;x_3}$ unsatisfiability.

As an easy consequence of the definitions, we have the following proposition, which establishes that some intuitive properties of reasons are valid.

Proposition 1 *Let φ be a QBF. Let μ be an assignment for φ. Let ν be a subset of the literals in μ. The following three facts hold:*

(1) If ν is a reason for φ_μ result, then any set $\nu' : \nu \subseteq \nu' \subseteq \{l : l \text{ is in } \mu\}$ is also a reason.
(2) If ν is a reason for φ_μ satisfiability, then the subset of ν consisting of the universal literals in ν is also a reason.
(3) If ν is a reason for φ_μ unsatisfiability, then the subset of ν consisting of the existential literals in ν is also a reason.

Proof. All the three items easily follow from the definitions. $\square$
If φ is (2), since $\{y_1\}$ is a reason for $\varphi_{y_1;\neg x_1;x_3}$ unsatisfiability, then the last item in Proposition 1 allows us to conclude that also the empty set is a reason for $\varphi_{y_1;\neg x_1;x_3}$ unsatisfiability.

4.1 Conflict-directed Backjumping

The following Theorem allows us to compute the reason for φ_μ result when the matrix of φ_μ contains a contradictory clause.

Theorem 1 *Let φ be a QBF. Let μ be an assignment for φ such that φ_μ contains a contradictory clause. Then there exists a clause C in the matrix of φ such that*

- *$C \cap \{l : l$ is in $\mu\} = \{\}$, and*
- *for each existential literal l in C, $\bar{l}$ is in μ.*

The set $\{l : \bar{l} \in C\} \cap \{l : l$ is in $\mu\}$ is a reason for φ_μ unsatisfiability.

Proof. In the following, if φ is a QBF, μ is an assignment for φ, and ν, ν' are two sets of literals, we write $\nu' \sim_\nu \mu$ to abbreviate that ν' agrees with μ on ν.

Let $\nu = \{l : \bar{l} \in C\} \cap \{l : l \text{ is in } \mu\}$. Consider a set $\nu' \sim_\nu \mu$. Then,

- for each existential literal l such that $\bar{l} \in C$, by construction $l \in \nu$ and thus $l \in \nu'$, and
- for each universal literal $l \in C$, l is not in μ and thus $l \notin \nu'$.

Thus, $\varphi_{\nu'}$ contains a contradictory clause and therefore is unsatisfiable. □

With reference to Figure 2, if φ is (2), this Theorem allows us to conclude, e.g., that

- if μ is $y_1; \neg x_1; x_3; \neg y_2; x_2$, then $\{y_1, \neg y_2, x_2\}$ is a reason for φ_μ unsatisfiability, and
- if μ is $y_1; \neg x_1; x_3; \neg y_2; \neg x_2$, then $\{y_1, \neg y_2, \neg x_2\}$ is a reason for φ_μ unsatisfiability.

From the above, as an obvious consequence of Proposition 1, we also have that

- if μ is $y_1; \neg x_1; x_3; \neg y_2; x_2$, then any set ν such that $\{x_2\} \subseteq \nu \subseteq \{y_1, \neg y_2, x_2\}$ is a reason for φ_μ unsatisfiability, and
- if μ is $y_1; \neg x_1; x_3; \neg y_2; \neg x_2$, then any set ν such that $\{\neg x_2\} \subseteq \nu \subseteq \{y_1, \neg y_2, \neg x_2\}$ is a reason for φ_μ unsatisfiability.

Our next step is to show how it is possible to compute reasons for φ_μ unsatisfiability while backtracking.

Theorem 2 *Let φ be a QBF. Let l be a literal. Let $\mu; l$ be an assignment for φ. Let ν be a reason for $\varphi_{\mu;l}$ unsatisfiability.*

(1) If $l \notin \nu$, then ν is a reason for φ_μ unsatisfiability.
(2) If $l \in \nu$ and l is universal, then $\nu \setminus \{l\}$ is a reason for φ_μ unsatisfiability.
(3) If $l \in \nu$, l is existential and is neither unit nor monotone in φ_μ, $\bar{l} \in \nu'$, ν' is a reason for $\varphi_{\mu;\bar{l}}$ unsatisfiability, then $(\nu \cup \nu') \setminus \{l, \bar{l}\}$ is a reason for φ_μ unsatisfiability.
(4) If $l \in \nu$, l is existential and unit in φ_μ, then there exists a clause C in the matrix of φ such that
  - *$l \in C$,*
  - *$C \cap \{l : l$ is in $\mu\} = \{\}$,*

- *for each existential literal $l' \neq l$ in C, $\overline{l'}$ is in μ, and*
- *for each universal literal $l' \in C$ with a greater level than l, $\overline{l'}$ is in μ.*

The set $(\{l' : \overline{l'} \in C\} \cap \{l : l \text{ is in } \mu\}) \cup (\nu \setminus \{l\})$ is a reason for φ_μ unsatisfiability.

Proof. We prove each statement separately. We recall that $\nu' \sim_\nu \mu$ means that ν' agrees with μ on ν.

(1) Assume that ν is not a reason for φ_μ unsatisfiability. Then there exists a set $\nu' \sim_\nu \mu$ such that $\varphi_{\nu'}$ is satisfiable. But this is not possible since $\nu' \sim_\nu \mu; l$, and ν is a reason for $\varphi_{\mu;l}$ unsatisfiability.

(2) Since l is universal, also $\nu \setminus \{l\}$ is a reason for $\varphi_{\mu;l}$ (see Proposition 1) and, given the first statement, $\nu \setminus \{l\}$ is a reason for φ_μ.

(3) Let $\nu'' = (\nu \cup \nu') \setminus \{l, \bar{l}\}$. Assume ν'' is not a reason for φ_μ. Then, there exists a set $\nu''' \sim_{\nu''} \mu$ such that $\varphi_{\nu'''}$ is satisfiable. Consider the set S of atoms having greater level than l in $\varphi_{\nu'''}$. Since $\varphi_{\nu'''}$ is satisfiable, there exists an eventually empty set of literals S' such that
- $\{|l| : l \in S'\} = S$,
- $\nu''' \cup S' \cup \{l\} \sim_{\nu'' \cup \{l\}} \mu; l$,
- $\nu''' \cup S' \cup \{\bar{l}\} \sim_{\nu'' \cup \{\bar{l}\}} \mu; \bar{l}$, and
- at least one between $\varphi_{\nu''' \cup S' \cup \{l\}}$ and $\varphi_{\nu''' \cup S' \cup \{\bar{l}\}}$ is satisfiable.

However, this is not possible because $\nu''' \cup S' \cup \{l\} \sim_\nu \mu; l$ and $\nu''' \cup S' \cup \{\bar{l}\} \sim_{\nu'} \mu; \bar{l}$.

(4) Let $\nu' = \{l' : \overline{l'} \in C\} \cap \{l : l \text{ is in } \mu\}$. Let $\nu'' = \nu \cup \nu' \setminus \{l\}$. From the hypotheses of the statement, it follows that
- $\nu'' \cup \{\bar{l}\}$ is a reason for $\varphi_{\mu;\bar{l}}$ unsatisfiability (see Proposition 1 and Theorem 1), and
- $\nu'' \cup \{l\} = \nu \cup \nu'$ is a reason for $\varphi_{\mu;l}$ unsatisfiability (see Proposition 1).

Assume ν'' is not a reason for φ_μ. Then, there exists a set $\nu''' \sim_{\nu''} \mu$ such that $\varphi_{\nu'''}$ is satisfiable. Let

$$S = \{l' : l' \in \nu', l' \notin \nu'''\}.$$

Clearly, for each $l' \in S$, l' is universal, l' is in μ, and $\overline{l'} \notin \nu'''$.

Consider the QBF $\varphi'_{\nu'''}$
- whose matrix is the same of $\varphi_{\nu'''}$, and
- whose prefix is obtained from the prefix of $\varphi_{\nu'''}$ by moving $\forall|l'|$ to the left of all the existential quantifiers whenever $l' \in S$.

Since $\varphi_{\nu'''} \supset \varphi'_{\nu'''}$ holds, $\varphi'_{\nu'''}$ is satisfiable. Then, also $\varphi'_{\nu''' \cup S}$ is satisfiable, and, since l is unit in $\varphi'_{\nu''' \cup S}$, also $\varphi'_{\nu''' \cup S \cup \{l\}}$ is satisfiable. However, this is not possible because $\varphi'_{\nu''' \cup S \cup \{l\}} = \varphi_{\nu''' \cup S \cup \{l\}}$, and $\nu''' \cup S \cup \{l\} \sim_\nu \mu; l$. □

If φ is (2), considering Theorem 2 and Figure 2:

(1) given that $\{y_1, \neg y_2, x_2\}$ and $\{y_1, \neg y_2, \neg x_2\}$ are reasons for $\varphi_{y_1;\neg x_1;x_3;\neg y_2;x_2}$

and $\varphi_{y_1;\neg x_1;x_3;\neg y_2;\neg x_2}$ unsatisfiability respectively (see the first paragraph below Theorem 1), the fourth statement allows us to conclude that $\{y_1, \neg y_2\}$ is a reason for φ_μ unsatisfiability when μ is $y_1; \neg x_1; x_3; \neg y_2$,

(2) then, the second statement allows us to conclude that $\{y_1\}$ is a reason for φ_μ unsatisfiability when μ is $y_1; \neg x_1; x_3$,

(3) then, the first statement allows us to conclude that $\{y_1\}$ is a reason for φ_μ unsatisfiability when μ is $\{y_1; \neg x_1\}$ and by the same process when μ is $\{y_1\}$,

(4) and finally, the second statement allows us to conclude that the empty set $\{\}$ is a reason for φ_μ unsatisfiability when μ is empty.

From the third item, it follows that checking whether $y_1; x_1$ satisfies φ is useless. Indeed, $\{y_1, x_1\}$ agrees with $y_1; \neg x_1$ on $\{y_1\}$, and thus we can immediately conclude that $\varphi_{y_1;x_1} = \varphi_{\{y_1,x_1\}}$ is unsatisfiable. Given this, a "backjumping" procedure would have avoided the generation of the branch leading to Ψ_8 in Figure 2.

Given Proposition 1, also $\{x_2\}$ and $\{\neg x_2\}$ are reasons for $\varphi_{y_1;\neg x_1;x_3;\neg y_2;x_2}$ and $\varphi_{y_1;\neg x_1;x_3;\neg y_2;\neg x_2}$ unsatisfiability respectively. The fourth statement of Theorem 2 allows us to conclude that the empty set is a reason for $\varphi_{y_1;\neg x_1;x_3;\neg y_2}$ unsatisfiability, and thus (first statement) also for $\varphi_{y_1;\neg x_1;x_3}$, $\varphi_{y_1;\neg x_1}$, φ_{y_1}, φ.

4.2 Solution-directed Backjumping

The following Theorem allows us to compute the reason for φ_μ result when the matrix of φ_μ is empty.

Theorem 3 *Let φ be a QBF. Let μ be an assignment for φ such that the matrix of φ_μ is empty. Let ν be a subset of the literals in μ such that the matrix of φ_ν is empty. Then ν is a reason for φ_μ satisfiability.*

Proof. As in previous proofs, we write $\nu' \sim_\nu \mu$ meaning that ν' agrees with μ on ν.

Assume that there exists a set $\nu' \sim_\nu \mu$ such that $\varphi_{\nu'}$ is unsatisfiable. Let

$$S = \{l : l \in \nu, l \notin \nu'\}.$$

Clearly, for each $l \in S$, l is existential, l is in μ, and $\bar{l} \notin \nu'$.

Consider the QBF $\varphi'_{\nu'}$

- whose matrix is the same of $\varphi_{\nu'}$, and
- whose prefix is obtained from the prefix of $\varphi_{\nu'}$ by moving $\exists|l|$ to the left of all the universal quantifiers whenever $l \in S$.

Since $\varphi'_{\nu'} \supset \varphi_{\nu'}$ holds, $\varphi'_{\nu'}$ is unsatisfiable. Then, also $\varphi'_{\nu' \cup S}$ is unsatisfiable. However, this is not possible because $\nu \subseteq \nu' \cup S$ and by hypothesis, the matrix of φ_ν is empty. $\square$

With reference to Figure 2, the above Theorem allows us to conclude that, e.g., $\{\neg y_1, x_2, x_3\}$ is a reason for φ_μ satisfiability, if μ is $\neg y_1; \neg y_2; x_2; x_3$ and φ is (2). Proposition 1 allows us to conclude that any set ν with $\{\neg y_1\} \subseteq \nu \subseteq \{\neg y_1, x_2, x_3\}$ is a reason for $\varphi_{\neg y_1; \neg y_2; x_2; x_3}$ satisfiability.

Theorem 4 *Let φ be a QBF. Let l be a literal. Let $\mu; l$ be an assignment for φ. Let ν be a reason for $\varphi_{\mu;l}$ satisfiability.*

(1) If l is not in ν, then ν is a reason for φ_μ satisfiability.
(2) If $l \in \nu$, and l is existential, then $\nu \setminus \{l\}$ is a reason for φ_μ satisfiability.
(3) If $l \in \nu$, l is universal and not monotone in φ_μ, $\bar{l} \in \nu'$, ν' is a reason for $\varphi_{\mu;\bar{l}}$ satisfiability, then $(\nu \cup \nu') \setminus \{l, \bar{l}\}$ is a reason for φ_μ satisfiability.

Proof. The proof of each item is analogous to the proof of the corresponding item in Theorem 2. $\square$

If φ is (2), considering Theorem 4 and Figure 2:

(1) given what we said in the paragraph below Theorem 3, the second statement allows us to conclude that $\{\neg y_1, x_2\}$ is a reason for φ_μ satisfiability when μ is $\neg y_1; \neg y_2; x_2$,
(2) then, the second statement allows us to conclude that $\{\neg y_1\}$ is a reason for φ_μ satisfiability when μ is $\neg y_1; \neg y_2$,
(3) and finally, the first statement allows us to conclude that $\{\neg y_1\}$ is a reason for φ_μ satisfiability when μ is $\neg y_1$.

From the second item it follows that checking whether $\neg y_1; y_2$ does not satisfy φ is useless. Indeed, $\{\neg y_1, y_2\}$ agrees with $\neg y_1; y_2$ on $\{\neg y_1\}$, and thus we can immediately conclude that $\varphi_{\neg y_1; y_2} = \varphi_{\{\neg y_1, y_2\}}$ is satisfiable. Given this, a "backjumping" procedure would have avoided the generation of the branch leading to Ψ_5 in Figure 2.

Given Proposition 1, also $\{\neg y_1\}$ is a reason for $\varphi_{\neg y_1; \neg y_2; x_2; x_3}$ satisfiability, and thus (first statement) also for $\varphi_{\neg y_1; \neg y_2; x_2}$, $\varphi_{\neg y_1; \neg y_2}$, $\varphi_{\neg y_1}$.

```
φ := ⟨the input QBF⟩;      Stack := ⟨the empty stack⟩;
function Simplify()
  do
    φ' := φ;
    if (⟨a contradictory clause is in φ⟩) return FALSE;
    if (⟨the matrix of φ is empty⟩) return TRUE;
    if (⟨l is unit in φ⟩)
      |l|.mode := UNIT; Extend(l); |l|.reason := SetReason(l);
    if (⟨l is monotone in φ⟩) |l|.mode := PURE; Extend(l);
  while (φ' ≠ φ);
  return UNDEF;
function Backjump(res)
  wr := InitWr(res);
  while (⟨Stack is not empty⟩)
    l := Retract();
    if (l ∈ wr)
      if(|l|.mode = L-SPLIT)
        |l|.mode := R-SPLIT; |l|.reason := wr; return l̄;
      wr := (wr ∪ |l|.reason) \ {l, l̄};
  return NULL;
function QuBE-BJ()
  do
    res := Simplify();
    if (res = UNDEF)   l := ChooseLiteral();
    else   l := Backjump(res);
    if (l ≠ NULL)   Extend(l);
  while (l ≠ NULL);
  return res;
```

Fig. 3. The algorithm of QuBE-BJ.

5 Implementation in QuBE

A procedure incorporating both conflict-directed and solution-directed backjumping has been implemented in QuBE. The idea behind the implementation is simple. When the search stops —because either a contradictory clause or the empty set of clauses is generated— an initial "working reason" is computed according to the results in Theorems 1, 3. Then, while backtracking,

- we simply retract the literal if it does not belong to the working reason (statement 1 of Theorems 2, 4): Even assuming that the literal had been assigned as a left split, performing the right branch would not change the result.

- if the literal belongs to the working reason, we update the working reason in order to keep it a subset of the current assignment. In doing this, we use the results in statements 2,3,4 of Theorem 2, and 2,3 of Theorem 4.

Notice that if we are retracting a literal l in the working reason and assigned as left split, then we have to store the reason (in a variable $|l|.reason$) for a possible use when backtracking from the right branch (see statement 3 of Theorems 2, 4). Finally, if a literal l is assigned as a unit, we store in $|l|.reason$ the subset of the existential literals in the current assignment, whose negation belongs to a clause in which l is unit (see statement 4 of Theorem 2): Indeed, $|l|.reason$ is a reason why assigning $\overline{l}$ would have produced a contradictory clause.

A more detailed description of this procedure (that we call QuBE-BJ) is presented in Figure 3. Consider Figure 3. Assume that φ is the input QBF and that μ is the assignment for φ corresponding to the sequence of literals stored in the stack. Then, in the procedure *Backjump*(res),

- *InitWr*(res) initializes the variable wr, storing the reason for φ_μ result. In our implementation:
  - If res = FALSE, φ_μ contains a contradictory clause. Let C be a clause in φ such that, for each literal l in C, $\overline{l}$ is in μ or l is universal in φ. Then *InitWr*(res) returns the set of existential literals l in μ such that $\overline{l} \in C$ (see Theorem 1). For example, if φ is (2), and μ is $y_1; \neg x_1; x_3; \neg y_2; x_2$, then *InitWr*($res$) returns $\{x_2\}$.
  - If res = TRUE, the matrix of φ_μ is empty. Then *InitWr*(res) returns the set of universal literals in the sequence obtained from μ by recursively eliminating universal literals l such that, for each clause C in φ, if $l \in C$ then there is another literal l' in the sequence with $l' \in C$ (see Theorem 3). For example, if φ is (2), and μ is $\neg y_1; \neg y_2; x_2; x_3$, then *InitWr*($res$) returns $\{\neg y_1\}$.
- If l is a literal l_i in $\mu = l_1; \ldots; l_m$ $(m \geq 0)$, $|l|.reason$ is the property of $|l|$ that, if set, stores the reason for $\varphi_{l_1;\ldots;l_{i-1};\overline{l_i}}$ result.

The procedure *Backjump*(res) replaces *Backtrack*(res) in Figure 1. Accordingly, *Backjump*(res) has to be invoked in place of *Backtrack*(res) (line 25 of Figure 3). Also, if l is a unit in φ_μ, then $\varphi_{\mu;\overline{l}}$ contains a contradictory clause. Let C be a clause in φ such that, for each literal $l' \in C$, $\overline{l'}$ is in $\mu;\overline{l}$ or l' is universal in φ. Then, the set ν of existential literals l' in $\mu;\overline{l}$ such that $\overline{l'} \in C$ is a reason for $\varphi_{\mu;\overline{l}}$ unsatisfiability. If $SetReason(l)$ is invoked when l is a unit in φ_μ, and assuming this function returns a set ν defined as above, the instruction

$$|l|.reason := SetReason(l);$$

is added to line 8. For example, if φ is (2), and μ is $y_1; \neg x_1; x_3; \neg y_2$, then

(1) x_2 is a unit in φ_μ,
(2) $\varphi_{\mu;\neg x_2}$ contains a contradictory clause, and
(3) $\{\neg x_2\}$ is stored in $x_2.reason$.

Considering the procedure *Backjump*(res) in Figure 3 —once the working reason is initialized (line 13) as we described above— *Backjump*(res) pops all the literals and corresponding QBFs (line 15) from the stack, till a literal l is reached such that l belongs to the working reason wr (line 16) and $|l|.mode$ is L-SPLIT (line 17). Indeed, if $l \in wr$ then

(1) either l is existential and $res =$ FALSE,
(2) or l is universal and $res =$ TRUE.

Thus, if $l \in wr$ and $|l|.mode$ is L-SPLIT, branching occurs, corresponding to (i) setting $|l|.mode$ to R-SPLIT, (ii) storing the working reason in $|l|.reason$, and (iii) returning $\bar{l}$ (line 18). If no such literal exists, NULL is returned (line 20).

Notice that if l is not in wr, we can safely retract l despite the other conditions (see statement 1 in Theorems 2, 4): Assigning $\bar{l}$ would not change the result of the computation. [3]

If l is in wr, but $|l|.mode$ is not L-SPLIT (i.e., see the paragraph below) is UNIT or R-SPLIT), we can use the results in statements 3,4 of Theorems 2, 4 to compute the new working reason (line 19).

Finally, given the reasons returned by our implementation of *InitWr*(res) and *SetReason*(l), it is easy to see that

- neither a universal, monotone literal nor an existential literal can belong to a reason for satisfiability,
- neither an existential, monotone literal nor a universal literal can belong to a reason for unsatisfiability.

QuBE-BJ is a generalization of the conflict-directed backjumping procedure implemented, e.g., in RELSAT: assuming that our procedure and RELSAT – without learning– perform the same nondeterministic choices, the two systems have the same behavior on QBFs without universal quantifiers.

To understand QuBE-BJ behavior, assume that φ is the QBF (2). As in section 3, we assume that *ChooseLiteral* returns the negation of the first atom in the prefix which occurs in the matrix of the QBF under consideration. Then,

[3] As a minor remark, notice that there is no need to clear the reason stored in $|l|.reason$ even if $|l|.mode$ is R-SPLIT.

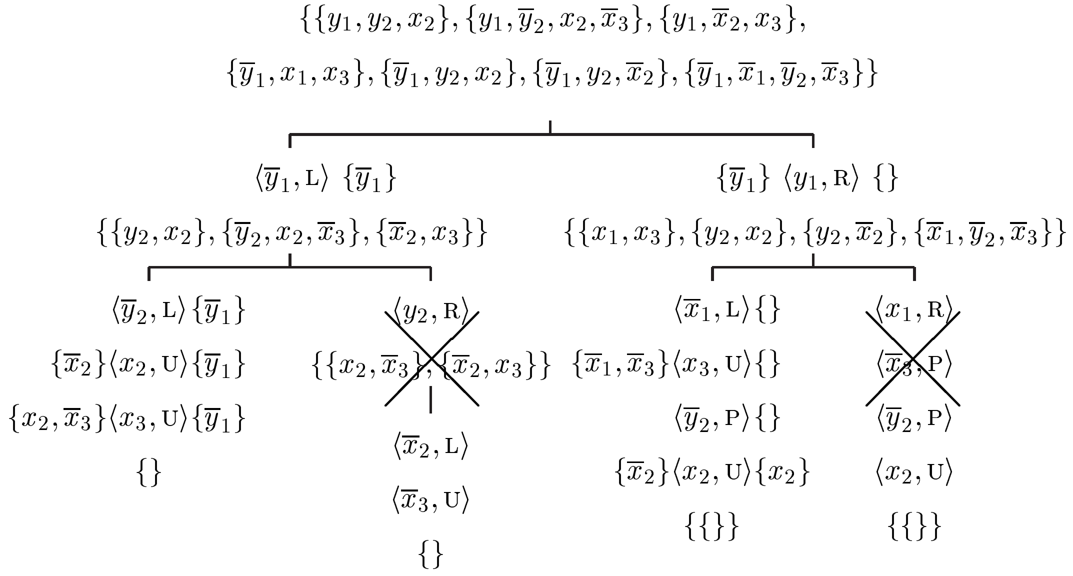

Fig. 4. QuBE-BJ computation tree for (2). U, P, L, R stand for UNIT, PURE, L-SPLIT, R-SPLIT respectively. The prefix is $\forall y_1 \exists x_1 \forall y_2 \exists x_2 \exists x_3$.

the tree searched by QuBE-BJ is represented in Figure 4. In the Figure, each node not belonging to a barred branch consists of

- a sequence of triples, each one having the form
  $$S_1\ \langle l, |l|.mode \rangle\ S_2$$
  where
  - $\langle l, |l|.mode \rangle$ means that l is the last literal assigned, with mode $|l|.mode$,
  - S_1 —when specified— is the set of literals stored in $|l|.reason$. S_1 thus is a reason for $\varphi_{\mu;\bar{l}}$ result (assuming μ is the sequence of literals assigned before l in the branch). In the Figure, all the variables $|l|.reason$ but $y_1.reason$, are set by *Simplify* while descending the search tree. $y_1.reason$ is set by *Backjump* while backtracking.
  - S_2 is the value assumed by the working reason wr, while backtracking from l.
- the matrix of the resulting QBF.

The barred branches are those explored by QuBE-BT and not by QuBE-BJ.

6 Experimental analysis

To evaluate the benefits deriving from backjumping, we compare QuBE-BT, and QuBE-BJ. For both systems, we use the generalization of Bömh's and Speckenmeyer's branching heuristics for SAT [13,14] which is used by

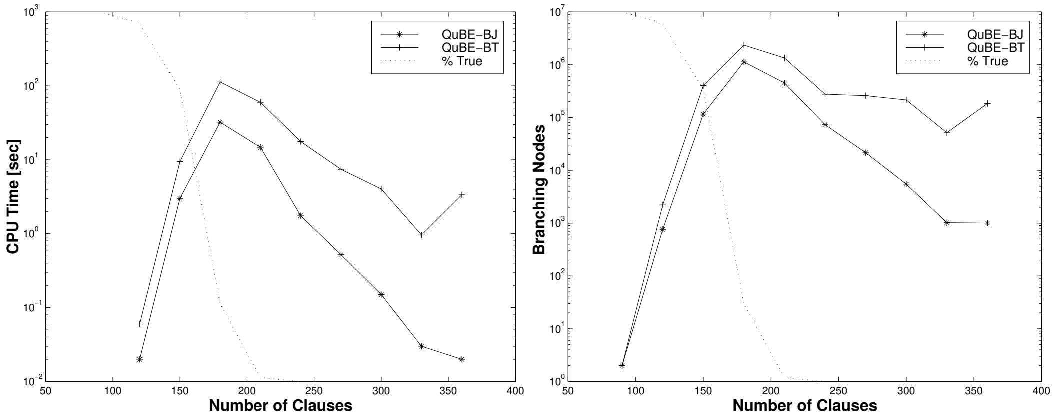

Fig. 5. QuBE-BT and QuBE-BJ median CPU time in seconds (left) and number of branching nodes (right). 100 samples/point. Background: satisfiability percentage.

Qsolve [7]: With reference to Figures 1 and 3, the function *ChooseLiteral*() returns a literal l satisfying the following three conditions:

(1) For each atom z having a greater level than l in the input QBF, z does not occur in φ.
(2) $|l|$ has the maximal vector

$$\langle H_1(|l|), H_2(|l|), \ldots, H_p(|l|) \rangle$$

under the lexicographic order. In the above vector,
- p is the maximum length of the clauses in the input QBF. By *length of a clause* we mean the number of existential literals in it: For example, the length of the first clause in (2) is 1.
- $H_i(|l|)$ $(1 \leq i \leq p)$ is defined as

$$H_i(|l|) = max(h_i(|l|), h_i(\neg|l|)) + 2 \times min(h_i(|l|), h_i(\neg|l|))$$

in which, for any literal l', $h_i(l')$ is the number of clauses C in the matrix of φ having length i and such that $l' \in C$.

(3) The "sign" of l has to maximize the number of subsumptions if l is existential, and the number of resolutions if l is universal. More precisely, given the condition

$$\sum_{i=1}^{p} h_i(|l|) \geq \sum_{i=1}^{p} h_i(\neg|l|), \tag{3}$$

- l has to be equal to $|l|$, if either l is existential and (3) is satisfied, or l is universal and (3) is not satisfied,
- l has to be equal to $\neg|l|$, otherwise.

The idea behind the selection of the atom is to choose the one that occurs as often as possible in shortest clauses: This idea comes from the SAT literature, see, e.g., [15]. The choice of the sign tries to minimize the chances to do both branches. All the tests have been run on a Pentium III, 600MHz, 128MB RAM, running SUSE Linux ver. 6.2.

We first consider sets of randomly generated QBFs. The generation model that we use is model A by Gent and Walsh [16]. In this model, each QBF has the following 4 properties:

(1) The prefix consists of k sequences, each sequence has n quantifiers, and each two quantifiers in a same sequence, are of the same type.
(2) The rightmost quantifier is $\exists$.
(3) The matrix consists of l clauses.
(4) Each clause consists of h literals of which at least 2 are existential.

Figure 5 shows the median of the CPU times (left) and number of branching nodes[4] (right) of QuBE-BT and QuBE-BJ when $k = 4$, $n = 30$, $h = 5$, and l (on the x-axis) is varied in such a way to empirically cover the "100% satisfiable – 100% unsatisfiable" transition (shown in the background). Notice the logarithmic scale on the y-axis. Consider Figure 5-left. As it can be observed, QuBE-BJ is faster (up-to two orders of magnitude) than QuBE-BT. QuBE-BJ better performances are due to its smaller number of branching nodes, as shown in Figure 5-right.

We also test QuBE-BT and QuBE-BJ on 91 structured QBFs corresponding to 75 planning problems [17] and 16 formal verification problems [18]. QuBE-BT and QuBE-BJ performances on 25 out of these 91 instances are summarized in Table 1. Of the other 66 problems,

- 36 (22 planning and 14 formal verification) cannot be solved by the two systems within the time limit,
- 20 (19 planning and 1 formal verification) are solved in less than 0.1s by both systems,
- 10 are simpler versions of the C*23V.24 and C*22V.23 instances: they are solved in less time, and the two systems have the same number of branching nodes.

Consider the data in Table 1. First, observe that QuBE-BJ never performs more branching nodes than QuBE-BT. This is to be expected since both systems use the same heuristic and the same pruning techniques. Indeed, as shown in [19], the introduction of new pruning techniques (such as learning [20]) can degrade the performance of a solver with backjumping.

[4] By number of branching nodes we mean the number of times an atom is assigned by left split.

		QUBE-BJ		QUBE-BT	
Test File	Value	Time	Br. Nodes	Time	Br. Nodes
B*3ii.4.3	FALSE	2.28	59390	> 1200	–
B*3ii.5.2	FALSE	52.47	525490	> 1200	–
B*3iii.4	FALSE	0.56	18952	> 1200	–
C*22v.23	TRUE	348.92	4194347	296.92	4194347
C*23v.24	TRUE	726.38	8388653	613.71	8388653
T*6.1.iv.11	FALSE	7.05	139505	31.47	567923
T*6.1.iv.12	TRUE	2.74	56387	13.99	318634
T*7.1.iv.13	FALSE	120.60	1948541	735.56	10106569
T*7.1.iv.14	TRUE	39.99	688621	309.38	5162011
I*10	TRUE	0.20	22626	0.60	93854
I*12	TRUE	1.15	127770	4.49	698411
I*14	TRUE	6.72	721338	33.22	5174804
I*16	TRUE	39.03	4072402	245.52	38253737
I*18	TRUE	227.45	22991578	> 1200	–
L*A1	FALSE	224.57	265610	> 1200	–
R*3*.50.3	TRUE	0.00	67	0.18	5345
R*3*.50.6	FALSE	0.05	1133	0.11	2802
R*3*.50.8	FALSE	0.10	2751	0.34	10353
R*7*.60.0	FALSE	0.05	2730	0.68	34770
R*7*.60.3	TRUE	0.12	5476	0.91	44665
R*7*.60.4	TRUE	0.05	2336	1.01	49421
R*7*.60.5	FALSE	0.11	5666	0.93	55765
R*7*.60.6	TRUE	0.09	3316	0.23	8440
R*7*.60.7	TRUE	0.11	4094	1.30	45313
Adder-2-unsat	FALSE	0.49	20256	2.28	150969

Table 1
Performances of QUBE-BT and QUBE-BJ on structured problems. Times are in CPU seconds.

Then, QUBE-BJ is able to solve 5 more instances within the time limit and, on the instances solved by both systems, sometimes it is much faster (compare, e.g., the performances on T*7.1.iv.14). Still, backjumping introduces some overhead: this is clear from the performances on the instances such as C*23V.24 and C*22V.23, where QuBE-BT and QuBE-BJ perform the same number of branching nodes (meaning that QuBE-BJ skips no nodes), and QuBE-BT is faster than QuBE-BJ. However, as it can be observed from the performances on C*23V.24, the computational overhead paid by QuBE-BJ is not dramatic (726.38s vs 613.71s).

7 Conclusions and Future work

In this paper we have shown that it is possible to generalize the conflict-directed backjumping schema for SAT to QBFs satisfiability. We have introduced solution-directed backjumping. As we have seen, the logic and implementation of conflict and solution directed backjumping are symmetric: they only differ for the computation of the initial working reason. Our implementation in QuBE shows that these forms of backjumping can produce significant speed ups. As far as we know, this is the first time a backjumping schema has been proposed, implemented and experimented for satisfiability of QBFs.

Beside the results presented in section 6, we have also comparatively tested QuBE and some of the other QBF solvers mentioned in the introduction. Our results show that QuBE compares well with respect to the other deciders, even without backjumping. For example, of the 91 structured problems, DECIDE, QuBE-BJ, QuBE-BT, QKN, Qsolve, Evaluate are able to solve 64, 55, 50, 46, 44, 2 samples respectively in less than 1200s. In this regard, we point out that DECIDE features "inversion of quantifiers" and "sampling" mechanisms which seem particularly effective on these benchmarks. For a comparative evaluation of these solvers, see [21].

QuBE, the tables with all the experimental results, and the test sets used are available at QuBE web page:

`www.mrg.dist.unige.it/star/qube`.

QuBE, besides the backjumping procedure above described, features six different branching heuristics, plus an adaptation of trivial truth (see [3,4]). (See [12] for a description of QuBE's available options). Concerning trivial truth, backjumping and their interactions, we have conducted an experimental evaluation on this. The result is that neither trivial truth is always better than backjumping, nor the other way around. On the other hand, the overhead of each of these techniques (on the test sets we have tried) is not dramatic, and thus it seems a good idea to use both of them. These and other results are reported in [21]. Recently, see [20], the authors extended QuBE to incorporate a learning schema which generalizes what has been done in SAT.

Finally, this work opens up the possibility to extend to QBFs satisfiability or, more in general, to Constraint Satisfaction extended with quantification over variables, all previous works studying the theoretical or experimental properties of conflict-directed backjumping, see, e.g. [9,22,23].

Acknowledgments

Thanks to Ian Gent and Patrick Prosser for the useful suggestions and comments. Ian also suggested the name "solution-directed backjumping", that we adopted. Thanks to Marco Cadoli, Rainer Feldmann, Theodor Lettman, Jussi Rintanen, Marco Schaerf and Stefan Schamberger for providing us with their systems and helping us to figure them out during our experimental analysis. Ayari Abdelwaheb and David Basin are thanked for providing us the formal verification benchmarks.

References

[1] E. Giunchiglia, M. Narizzano, A. Tacchella, Backjumping for quantified boolean logic satisfiability, in: B. Nebel (Ed.), Proceedings of the seventeenth International Conference on Artificial Intelligence (IJCAI-01), Morgan Kaufmann Publishers, Inc., San Francisco, CA, 2001, pp. 275–281.

[2] H. Kleine-Büning, M. Karpinski, A. Flögel, Resolution for quantified boolean formulas, Information and computation 117 (1) (1995) 12–18.

[3] M. Cadoli, A. Giovanardi, M. Schaerf, An algorithm to evaluate quantified Boolean formulae, in: Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98) and of the 10th Conference on Innovative Applications of Artificial Intelligence (IAAI-98), AAAI Press, Menlo Park, 1998, pp. 262–267.

[4] M. Cadoli, M. Schaerf, A. Giovanardi, M. Giovanardi, An algorithm to evaluate quantified boolean formulae and its experimental evaluation, Journal of Automated Reasoning 28 (2002) 101–142.

[5] J. Rintanen, Improvements to the evaluation of quantified boolean formulae, in: D. Thomas (Ed.), Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99-Vol2), Morgan Kaufmann Publishers, S.F., 1999, pp. 1192–1197.

[6] U. Egly, T. Eiter, H. Tompits, S. Woltran, Solving advanced reasoning tasks using quantified boolean formulas, in: Proceedings of the 7th Conference on Artificial Intelligence (AAAI-00) and of the 12th Conference on Innovative Applications of Artificial Intelligence (IAAI-00), AAAI Press, Menlo Park, CA, 2000, pp. 417–422.

[7] R. Feldmann, B. Monien, S. Schamberger, A distributed algorithm to evaluate quantified boolean formulae, in: Proceedings of the 7th Conference on Artificial Intelligence (AAAI-00) and of the 12th Conference on Innovative Applications of Artificial Intelligence (IAAI-00), AAAI Press, Menlo Park, CA, 2000, pp. 285–290.

[8] M. Davis, G. Logemann, D. Loveland, A machine program for theorem proving, Journal of the ACM 5 (7).

[9] P. Prosser, Hybrid algorithms for the constraint satisfaction problem, Computational Intelligence 9 (3) (1993) 268–299.

[10] R. J. Bayardo, Jr., R. C. Schrag, Using CSP look-back techniques to solve real-world SAT instances, in: Proceedings of the 14th National Conference on Artificial Intelligence and 9th Innovative Applications of Artificial Intelligence Conference (AAAI-97/IAAI-97), AAAI Press, Menlo Park, 1997, pp. 203–208.

[11] E. Giunchiglia, M. Maratea, A. Tacchella, D. Zambonin, Evaluating search heuristics and optimization techniques in propositional satisfiability, in: Proc. of the International Joint Conference on Automated Reasoning (IJCAR'2001), LNAI 2083, 2001, pp. 347–363.

[12] E. Giunchiglia, M. Narizzano, A. Tacchella, QuBE: A system for deciding quantified boolean formulas satisfiability, in: Proc. of the International Joint Conference on Automated Reasoning (IJCAR'2001), LNAI 2083, 2001, pp. 364–369, QuBE is available at `www.mrg.dist.unige.it/star/qube`.

[13] M. Buro, H. Buning, Report on a SAT competition, Tech. Rep. 110, University of Paderborn, Germany (November 1992).

[14] M. Böhm, E. Speckenmeyer, A fast parallel SAT-solver – efficient workload balancing, Annals of Mathematics and Artificial Intelligence 17 (1996) 381–400.

[15] J. W. Freeman, Improvements to propositional satisfiability search algorithms, Ph.D. thesis, University of Pennsylvania (1995).

[16] I. Gent, T. Walsh, Beyond NP: The QSAT phase transition, in: Proceedings of the 6th National Conference on Artificial Intelligence (AAAI-99); Proceedings of the 11th Conference on Innovative Applications of Artificial Intelligence, AAAI/MIT Press, Menlo Park, Cal., 1999, pp. 648–653.

[17] J. Rintanen, Constructing conditional plans by a theorem prover, Journal of Artificial Intelligence Research 10 (1999) 323–352.

[18] A. Abdelwaheb, D. Basin, Bounded model construction for monadic second-order logics, in: 12th International Conference on Computer-Aided Verification (CAV'00), no. 1855 in Lecture Notes in Computer Science, Springer-Verlag, Chicago, USA, 2000, pp. 99–113.

[19] P. Prosser, Domain filtering can degrade intelligent backjumping search, in: Proc. IJCAI, 1993, pp. 262–267.

[20] E. Giunchiglia, M. Narizzano, A. Tacchella, Learning for Quantified Boolean Logic Satisfiability, in: Proc. 18th National Conference on Artificial Intelligence (AAAI) (AAAI'2002), 2002, pp. 649–654.

[21] E. Giunchiglia, M. Narizzano, A. Tacchella, An analysis of backjumping and trivial truth in quantified boolean formulas satisfiability, in: Proc. AI*IA 2001: Advances in Artificial Intelligence (AI*IA'2001), LNAI 2175, 2001, pp. 111–122.

[22] G. Kondrak, P. van Beek, A theoretical evaluation of selected backtracking algorithms, Artificial Intelligence 89 (1–2) (1989) 365–387.

[23] X. Chen, P. van Beek, Conflict-directed backjumping revisited, Journal of Artificial Intelligence Research 14 (2001) 53–81.