Incrementally Computing Minimal Unsatisfiable Cores of QBFs via a Clause Group Solver API

Florian Lonsing and Uwe Egly

Knowledge-Based Systems Group
Institute of Information Systems
Vienna University of Technology, Austria
http://www.kr.tuwien.ac.at/

18th International Conference on Theory and Applications of Satisfiability Testing,
September 24 - 27, Austin, Texas, USA

Overview

Quantified Boolean Formulas (QBF):

- Propositional logic with explicitly existentially/universally quantified variables.
- PSPACE-completeness: applications in AI, verification, synthesis,...

Incremental QBF Solving:

- Solving sequences of related QBFs while keeping learned information.
- Solver API called incrementally from application programs.

DepQBF:

- Incremental search-based QBF solver with clause and cube learning.
- Free software (GPLv3): `http://lonsing.github.io/depqbf/`

Contributions (1/2)

Clause Groups:

- Clause group: set of clauses incrementally added to/removed from formula.
- First implemented in SAT solver zChaff (2001) using bit masking to track learned clauses, no support of assumptions.

Novel Clause Group API in DepQBF:

- Clause groups implemented based on selector variables and incremental solving under assumptions.
- *Internally*, solver augments added clauses by selector variables.
- Unique feature: handling of selector variables and assumptions entirely carried out by the solver.
- User's perspective: encodings are not cluttered with selector variables.

Contributions (1/2)

Clause Groups:

- Clause group: set of clauses incrementally added to/removed from formula.
- First implemented in SAT solver zChaff (2001) using bit masking to track learned clauses, no support of assumptions.

Novel Clause Group API in DepQBF:

- Clause groups implemented based on selector variables and incremental solving under assumptions.
- *Internally*, solver augments added clauses by selector variables.
- Unique feature: handling of selector variables and assumptions entirely carried out by the solver.
- User's perspective: encodings are not cluttered with selector variables.

Contributions (2/2)

Minimal Unsatisfiable Cores (MUCs) of QBFs:

- Alternative terminology: minimal unsatisfiable subsets (MUS).
- Consider QBF $\hat{Q}.\phi$ in prenex CNF with prefix $\hat{Q}$ and CNF ϕ.
- Let $\phi' \subseteq \phi$ be a minimal subset such that $\hat{Q}.\phi'$ is unsatisfiable, then $\hat{Q}.\phi'$ is a MUC of QBF $\hat{Q}.\phi$.

Computation of MUCs of QBFs:

- Well-studied problem for SAT but not for QBF.
- First experimental results for computation of MUCs of QBFs based on DepQBF's novel clause group API.
- Iterative refinement of nonminimal unsatisfiable cores.

Clause Group API Example (1/7)

```
Solver *s = create();
new_scope_at_nesting
  (s,QTYPE_FORALL,1);
add(s,1);add(s,2);add(s,0);
new_scope_at_nesting
  (s,QTYPE_EXISTS,2);
add(s,3);add(s,4);add(s,0);
```

$\forall x_1, x_2 \exists x_3, x_4.$

- `create()`: create solver instance.
- `new_scope_at_nesting(...)`: add new quantifier block to prefix.
- `add(...)`: add variables to quantifier blocks, terminated by zero.

Clause Group API Example (2/7)

```
ClauseGroupID id1 =
  new_cls_grp(s);
open_cls_grp(s,id1);
add(s,-1);add(s,-3);
  add(s,0);
close_cls_grp(s,id1);
```

$\forall x_1, x_2 \exists x_3, x_4.$
$(\mathbf{s_1} \vee \neg \mathbf{x_1} \vee \neg \mathbf{x_3})$

- `new_cls_grp(...)`: create new clause group and return its ID.
- `open_cls_grp(id)`: open clause group `id`; clauses added in the following are put into group `id`.
- `add(...)`: add literals to clauses, terminated by zero.
- *Internally*, solver augments clauses in a group by a selector variable (s_1).
- `close_cls_grp(id)`: closes group `id`.

Clause Group API Example (3/7)

```
ClauseGroupID id2 =
  new_cls_grp(s);
open_cls_grp(s,id2);
add(s,1);add(s,2);
  add(s,4);add(s,0);
add(s,1);add(s,-4);
  add(s,0);
close_cls_grp(s,id2);
```

$\forall x_1, x_2 \exists x_3, x_4.$
$(s_1 \vee \neg x_1 \vee \neg x_3) \wedge$
$(\mathbf{s_2} \vee \mathbf{x_1} \vee \mathbf{x_2} \vee \mathbf{x_4}) \wedge$
$(\mathbf{s_2} \vee \mathbf{x_1} \vee \neg \mathbf{x_4})$

- Arbitrary number of clause groups can be created, identified by their IDs.
- Selector variables are invisible to the user.
- Name clashes between user-given variables and selector variables are avoided by *internal dynamic* renaming of selector variables.

Clause Group API Example (4/7)

```
Result res = sat(s);
assert(res == RESULT_UNSAT);
ClauseGroupID *rgrps =
  get_relevant_cls_grps(s);
assert(rgrps[0] == id2);
reset(s);
```

$\forall x_1, x_2 \exists x_3, x_4.$
$(\bot \lor \neg x_1 \lor \neg x_3) \land$
$(\bot \lor x_1 \lor x_2 \lor x_4) \land$
$(\bot \lor x_1 \lor \neg x_4)$

- `sat(...)`: solve formula, *internally* selector variables are assigned to activate clause groups and their clauses (s_i replaced by $\bot$).
- `get_relevant_cls_grps(...)`: if formula ψ is unsatisfiable, returns a list of group IDs which contain clauses participating in the resolution refutation.
- Unsatisfiable core (UC) of ψ, not necessarily minimal.
- *Internally*, solver maps selector variables to IDs of clause groups.

Clause Group API Example (5/7)

```
deactivate_cls_grp(s,rgrps[0]);
res = sat(s);
assert(res == RESULT_SAT);
reset(s);
```

$\forall x_1, x_2 \exists x_3, x_4.$
$(\bot \vee \neg x_1 \vee \neg x_3) \wedge$
$(\top \vee x_1 \vee x_2 \vee x_4) \wedge$
$(\top \vee x_1 \vee \neg x_4)$

- `deactivate_cls_grp`: *internally* selector variable of group `id` is *temporarily* assigned to satisfy clauses (s_i replaced by $\top$).
- Deactivated groups stay deactivated in all forthcoming calls of `sat(...)`.

Clause Group API Example (6/7)

```
activate_cls_grp(s,rgrps[0]);
free(rgrps);
```

$\forall x_1, x_2 \exists x_3, x_4.$
$(s_1 \vee \neg x_1 \vee \neg x_3) \wedge$
$(\bot \vee x_1 \vee x_2 \vee x_4) \wedge$
$(\bot \vee x_1 \vee \neg x_4)$

- `activate_cls_grp`: *internally* selector variable of group `id` is assigned to *not* satisfy clauses.
- Activated groups stay activated in all forthcoming calls of `sat(...)`.
- Newly created groups are always activated.

Clause Group API Example (7/7)

```
delete_cls_grp(s,id1);
res = sat(s);
assert(res == RESULT_UNSAT);
delete(s);
```

$\forall x_1, x_2 \exists x_3, x_4.$
$\cancel{(\top \vee \neg x_1 \vee \neg x_3)} \wedge$
$(\bot \vee x_1 \vee x_2 \vee x_4) \wedge$
$(\bot \vee x_1 \vee \neg x_4)$

- delete_cls_grp: *internally* selector variable of group id is *permanently* assigned to satisfy clauses.
- IDs of deleted groups are invalid, group can no longer be accessed via API.
- Clauses in deleted groups are physically removed from data structures in a garbage collection phase.

Computing MUCs of QBFs by Clause Group API

1. Let $\hat{Q}.\phi$ be an unsatisfiable QBF. Every clause $C \in \phi$ is put in an individual clause group.
2. Let $\hat{Q}.\phi'$ denote a (nonminimal) unsatisfiable core (UC) of $\hat{Q}.\phi$.
3. Initially, set $\hat{Q}.\phi' := \hat{Q}.\phi$ (overapproximation of final MUC).
4. Test removal of every clause C in UC $\hat{Q}.\phi'$ by `deactivate_cls_grp`. If $\hat{Q}.(\phi' \setminus \{C\})$ satisfiable then C is part of an MUC, call `activate_cls_grp`.
5. Otherwise, $\hat{Q}.(\phi' \setminus \{C\})$ is unsatisfiable. Replace $\hat{Q}.\phi'$ by a UC of $\hat{Q}.(\phi' \setminus \{C\})$ obtained by `get_relevant_cls_grps`. Clauses not in the UC are irrelevant and are deleted by `delete_cls_grp`.
6. Repeat steps 4 and 5 until every clause in current UC has been tested.
7. Finally, $\hat{Q}.(\phi' \setminus \{C\})$ is satisfiable for every $C \in \phi'$ and $\hat{Q}.\phi'$ is an MUC.

Computing MUCs of QBFs by Clause Group API

1. Let $\hat{Q}.\phi$ be an unsatisfiable QBF. Every clause $C \in \phi$ is put in an individual clause group.
2. Let $\hat{Q}.\phi'$ denote a (nonminimal) unsatisfiable core (UC) of $\hat{Q}.\phi$.
3. Initially, set $\hat{Q}.\phi' := \hat{Q}.\phi$ (overapproximation of final MUC).
4. Test removal of every clause C in UC $\hat{Q}.\phi'$ by `deactivate_cls_grp`. If $\hat{Q}.(\phi' \setminus \{C\})$ satisfiable then C is part of an MUC, call `activate_cls_grp`.
5. Otherwise, $\hat{Q}.(\phi' \setminus \{C\})$ is unsatisfiable. Replace $\hat{Q}.\phi'$ by a UC of $\hat{Q}.(\phi' \setminus \{C\})$ obtained by `get_relevant_cls_grps`. Clauses not in the UC are irrelevant and are deleted by `delete_cls_grp`.
6. Repeat steps 4 and 5 until every clause in current UC has been tested.
7. Finally, $\hat{Q}.(\phi' \setminus \{C\})$ is satisfiable for every $C \in \phi'$ and $\hat{Q}.\phi'$ is an MUC.

Computing MUCs of QBFs by Clause Group API

1. Let $\hat{Q}.\phi$ be an unsatisfiable QBF. Every clause $C \in \phi$ is put in an individual clause group.
2. Let $\hat{Q}.\phi'$ denote a (nonminimal) unsatisfiable core (UC) of $\hat{Q}.\phi$.
3. Initially, set $\hat{Q}.\phi' := \hat{Q}.\phi$ (overapproximation of final MUC).
4. Test removal of every clause C in UC $\hat{Q}.\phi'$ by `deactivate_cls_grp`. If $\hat{Q}.(\phi' \setminus \{C\})$ satisfiable then C is part of an MUC, call `activate_cls_grp`.
5. Otherwise, $\hat{Q}.(\phi' \setminus \{C\})$ is unsatisfiable. Replace $\hat{Q}.\phi'$ by a UC of $\hat{Q}.(\phi' \setminus \{C\})$ obtained by `get_relevant_cls_grps`. Clauses not in the UC are irrelevant and are deleted by `delete_cls_grp`.
6. Repeat steps 4 and 5 until every clause in current UC has been tested.
7. Finally, $\hat{Q}.(\phi' \setminus \{C\})$ is satisfiable for every $C \in \phi'$ and $\hat{Q}.\phi'$ is an MUC.

Computing MUCs of QBFs by Clause Group API

1. Let $\hat{Q}.\phi$ be an unsatisfiable QBF. Every clause $C \in \phi$ is put in an individual clause group.
2. Let $\hat{Q}.\phi'$ denote a (nonminimal) unsatisfiable core (UC) of $\hat{Q}.\phi$.
3. Initially, set $\hat{Q}.\phi' := \hat{Q}.\phi$ (overapproximation of final MUC).
4. Test removal of every clause C in UC $\hat{Q}.\phi'$ by `deactivate_cls_grp`. If $\hat{Q}.(\phi' \setminus \{C\})$ satisfiable then C is part of an MUC, call `activate_cls_grp`.
5. Otherwise, $\hat{Q}.(\phi' \setminus \{C\})$ is unsatisfiable. Replace $\hat{Q}.\phi'$ by a UC of $\hat{Q}.(\phi' \setminus \{C\})$ obtained by `get_relevant_cls_grps`. Clauses not in the UC are irrelevant and are deleted by `delete_cls_grp`.
6. Repeat steps 4 and 5 until every clause in current UC has been tested.
7. Finally, $\hat{Q}.(\phi' \setminus \{C\})$ is satisfiable for every $C \in \phi'$ and $\hat{Q}.\phi'$ is an MUC.

Computing MUCs of QBFs by Clause Group API

1. Let $\hat{Q}.\phi$ be an unsatisfiable QBF. Every clause $C \in \phi$ is put in an individual clause group.
2. Let $\hat{Q}.\phi'$ denote a (nonminimal) unsatisfiable core (UC) of $\hat{Q}.\phi$.
3. Initially, set $\hat{Q}.\phi' := \hat{Q}.\phi$ (overapproximation of final MUC).
4. Test removal of every clause C in UC $\hat{Q}.\phi'$ by `deactivate_cls_grp`. If $\hat{Q}.(\phi' \setminus \{C\})$ satisfiable then C is part of an MUC, call `activate_cls_grp`.
5. Otherwise, $\hat{Q}.(\phi' \setminus \{C\})$ is unsatisfiable. Replace $\hat{Q}.\phi'$ by a UC of $\hat{Q}.(\phi' \setminus \{C\})$ obtained by `get_relevant_cls_grps`. Clauses not in the UC are irrelevant and are deleted by `delete_cls_grp`.
6. Repeat steps 4 and 5 until every clause in current UC has been tested.
7. Finally, $\hat{Q}.(\phi' \setminus \{C\})$ is satisfiable for every $C \in \phi'$ and $\hat{Q}.\phi'$ is an MUC.

Experiments

MUCs	$\Sigma\|CNF\|$	$\Sigma\|MUC\|$	*Solver Calls*	Avg. $\|MUC\|$	Med. $\|MUC\|$
182	4,744,494	73,206	81,631	6.1%	2.9%

- 190 unsatisfiable instances from applications track of the QBF Gallery 2014.
- All instances preprocessed by Bloqqer.
- 900s timeout for whole workflow (solving initial formula, computing MUC).
- MUCs computed for 95% of solved unsatisfiable instances.
- MUCs are small: 1.55% of total CNF sizes, small average and median sizes.
- Worst case: one solver call for each clause in initial CNF.
- UC extraction pays off: number of solver calls reduced by factor of 58.

Conclusion

Incremental QBF Solving based on Clause Groups:

- Incrementally add/remove sets of clauses via solver API.
- API on top of state of art technology: selector variables and assumptions.
- Unique feature: *internal* management of selector variables and assumptions.
- Easier and less error-prone integration of solver in tool chains.
- Implementation applicable to any SAT/QBF solver supporting assumptions.

Computation of Minimal Unsatisfiable Cores (MUCs):

- First experimental results based on clause group API.
- Further approaches from computation of SAT MUCs may be applied to QBF.

Extended version of paper with appendix: `http://arxiv.org/abs/1502.02484`
DepQBF source code: `http://lonsing.github.io/depqbf/`