A strategy game based approach on solving logic formulas

Yifan He

Supervised By Dr. Abdallah Saffidine

Thesis A seminar

2021 T2

Motivation

- QBF is similar to 2-player strategy game in a theoretical level
- All search based QBF solving algorithms used depth first search
- Depth first search or iterative deepening search is rarely used in **solving** strategy games
  - Shogi dfs cannot solve a game with more than 17 steps
  - best first search can solve games with 100+ steps.
- Depth first search based algorithms might be trapped on the wrong side of the searching space
- QBF has a very deep searching space, dfs is not ideal

Outline

- Technical Background
- Aim of the project and Preliminary Results
- Future Plan

Outline

- Technical Background
  - QBF preliminaries
  - QDLL
  - Backjumping
  - QCDCL
  - QBF as strategy game
  - PNS
- Aim of the project and Preliminary Results
- Future Plan

QBF

- QBF expression: quantifier prefix + propositional formula (CNF)

  f: $Q_1 X_1 Q_2 X_2 \dots Q_n X_n \Phi$ $\quad (Q_i \in \{\exists, \forall\})$

- Semantic of QBF

  - If Φ contains a contradictory clause (i.e. a clause without existential literal), false
  - If Φ has all clauses satisfied, true
  - If Q_1 is existential, f is true iff either $Q_2 X_2 \dots Q_n X_n \Phi(X_1)$ **or** $Q_2 X_2 \dots Q_n X_n \Phi(\neg X_1)$ is true.
  - If Q_1 is universal, f is true iff both $Q_2 X_2 \dots Q_n X_n \ \Phi(X_1)$ **and** $Q_2 X_2 \dots Q_n X_n \ \Phi(\neg X_1)$ is true.

- Significance of QBF: model checking, planning, games…
- QBF solvers: expansion-based (e.g. caqe), search-based (e.g. DepQBF)

Unit and pure literals

- Unit literal

  - A literal l is called unit if it is the only existential literal in a clause C in Φ, and all universal variables in C occurs to the right of l in the prefix.

  - unit propagation: $\Phi = \Phi_l$

- Pure literal:

  - A literal l is called pure if l is existential and $\neg l$ does not appear in Φ or it is universal and l does not appear in Φ.

  - pure literal elimination: $\Phi = \Phi_l$

- $\exists x, y \forall z \exists w \ \ (x \vee \neg z) \wedge (y \vee w \vee \neg z) \wedge (y \vee \neg w \vee \neg z)$
- In the example, x is unit, z and y are pure

QDLL

- Modification of the DPLL (Davis–Putnam-Logemann–Loveland) algorithm in SAT
- Proposed by Cadoli, Giovanardi, and Schaerf in around 1998
- Baseline procedure for search based solver
- QBF semantic + unit propagation + pure literal elimination

QDLL (conventional)

Algorithm 1: QDLL algorithm

```
1  Boolean QDLL(f, μ)
2      if f_μ has a contradictory clause then
3          return False
4      if all clauses in f_μ are satisfied then
5          return True
6      if l is unit in f_μ then
7          return QDLL(f, μ; l)
8      if l is pure in f_μ then
9          return QDLL(f, μ; l)
10     l = get_literal(f, μ)
11     if l is existential in f then
12         return QDLL(f, μ; l) or QDLL(f, μ; ¬l)
13     return QDLL(f, μ; l) and QDLL(f, μ; ¬l)
```

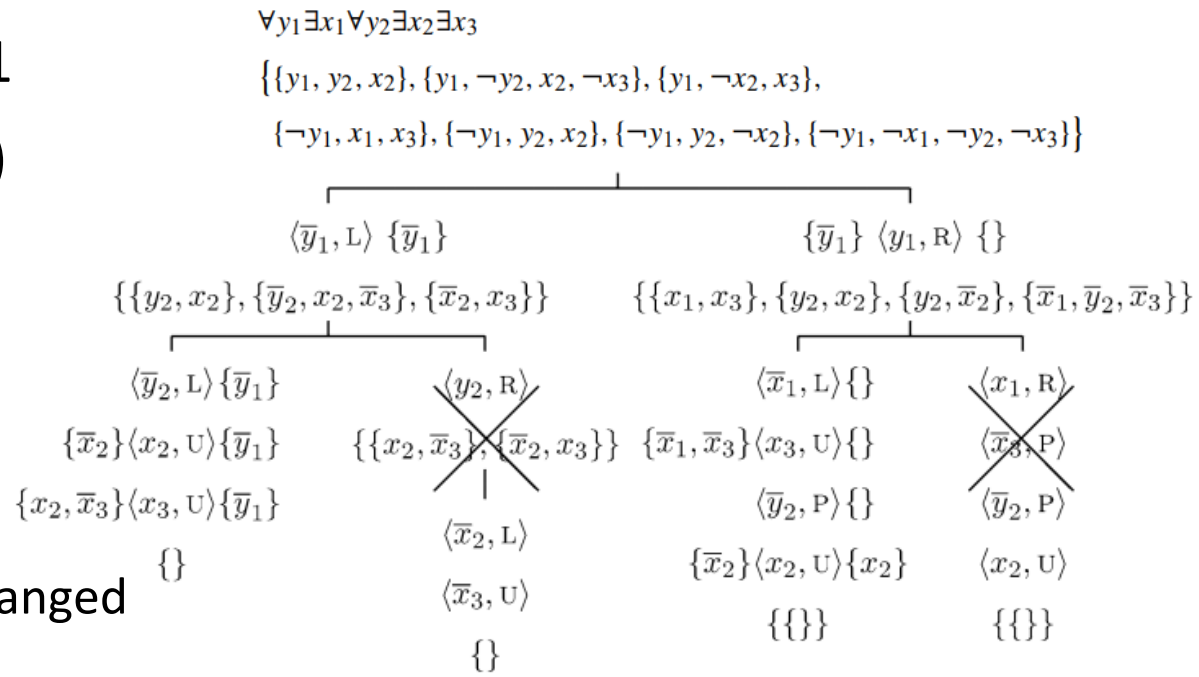
QDLL (original paper)

Algorithm 2 : QDLL algorithm

```
1  Boolean QDLL(f, μ)
2      if f_μ has a contradictory clause then
3          return False
4      if all clauses in f_μ are satisfied then
5          return True
6      if f_μ contains only existential literals then
7          return SAT(f_μ)
8      G := propositional formula obtained by removing all universal
         variables from f_μ
9      if SAT(G) then
10         return True
11     if l is unit in f_μ then
12         return QDLL(f, μ; l)
13     if l is pure in f_μ then
14         return QDLL(f, μ; l)
15     l = get_literal(f, μ)
16     if l is existential in f then
17         return QDLL(f, μ; l) or QDLL(f, μ; ¬l)
18     return QDLL(f, μ; l) and QDLL(f, μ; ¬l)
```

Backjumping (Dependency backtracking)

- Proposed by E. Giunchiglia in 2001
- Quaffle-BJ (2006), QUBE-BJ (2004)
- Main idea:
  - reason for conflict/solution
  - maintain reason during backtrack
  - pruning and level-cut
  - the body of the QDLL algorithm is unchanged



Reference:
https://www.sciencedirect.com/science/article/pii/S0004370202003739

Backjumping (Dependency backtracking)

- Reason for conflict : Suppose that ϕ_μ is unsatisfiable. Let U be the set of universal literals in μ. Then, the reason for conflict ν is a subset of existential literals in μ such that $\phi_{U;\nu}$ is unsatisfiable.

- Reason for solution: Suppose that ϕ_μ is satisfiable. Let E be the set of existential literals in μ. Then, the reason for solution ν is a subset of universal literals in μ such that $\phi_{E;\nu}$ is satisfiable.

- Pruning condition: if l is the branching literal in ϕ_μ, and it is not in the reason for conflict/solution of ϕ_μ, we can skip the exploration of $\neg l$.

Backjumping (Reason computation rules)

- Terminal node:
  - 1. If ϕ_μ contains an empty clause C. The reason for conflict is the set of existential literals in C.
  - 2. If ϕ_μ has all clauses satisfied. The reason for solution is a set of universal literals in μ that can be obtained by repeatedly removing universal literals from μ such that all clauses are still satisfied.
  - **Remark: Not explicitly stated in the original paper, the removing order is not arbitrary, last assigned to first assigned is one possible valid order but first assigned to last assigned is not.**

Backjumping (Reason computation rules)

- Maintain reason during backtrack, calculate reason r for ϕ_μ
- ν is the reason for $\phi_{\mu;l}$ and ν' is the reason for $\phi_{\mu;\neg l}$

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*
   - *$l \in C$,*
   - *$C \cap \{l : l \text{ is in } \mu\} = \{\}$,*
   - *for each existential literal $l' \neq l$ in C, $\bar{l}'$ is in μ, and*
   - *for each universal literal $l' \in C$ with a greater level than l, $\bar{l}'$ is in μ.*

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

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.*

Reference: https://www.sciencedirect.com/science/article/pii/S0004370202003739

Backjumping (Reason computation rules)

- Maintain reason during backtrack, calculate reason r for ϕ_μ
- ν is the reason for $\phi_{\mu;l}$ and ν' is the reason for $\phi_{\mu;\neg l}$

Property of l	**Result of $\phi_{\mu;l}$**	**Result of $\phi_{\mu;\neg l}$**	**Reason for ϕ_μ**
Pure	SAT/UNSAT	-	$\nu \backslash \{l\}$
Unit	SAT	-	$\nu \backslash \{l\}$
	UNSAT $l \notin \nu$	UNSAT	ν
	UNSAT	UNSAT	$\nu \cup \nu' \backslash \{l, \neg l\}$
Universal branching	UNSAT	-	$\nu \backslash \{l\}$
	SAT and $l \notin \nu$	Pruned	ν
	SAT	UNSAT	$\nu' \backslash \{\neg l\}$
	SAT	SAT	$\nu \cup \nu' \backslash \{l, \neg l\}$
Existential branching	Dual to universal branching case		

Backjumping (Dependency backtracking)

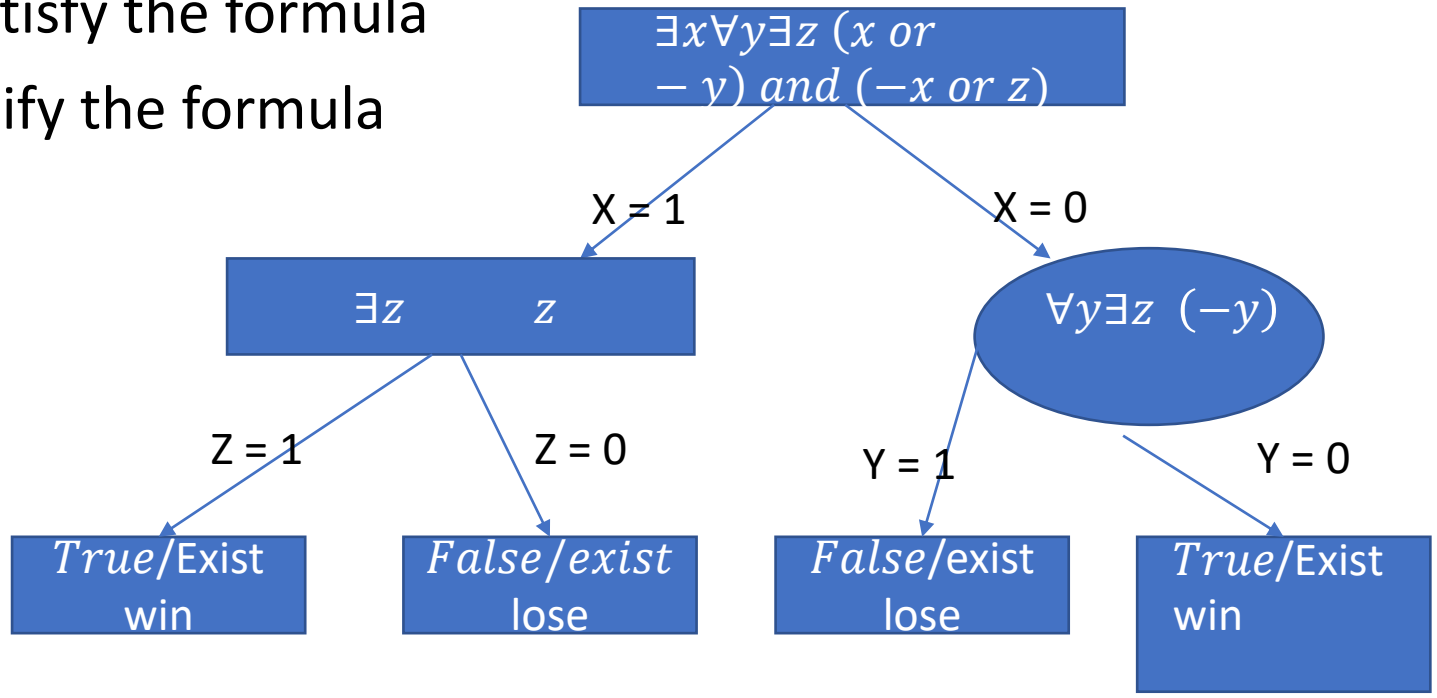
- Pros: relatively easy to implement without too much computation overhead, guaranteed to be not worse than baseline QDLL
- Cons: Conflicts/solutions can only affect the search on the current searching path

QCDCL

- Modify the CDCL algorithm in SAT, proposed by l. Zhang/E. Giunchiglia
- QUBE (2004-2010), DepQBF (2010-)
- Learned new clauses/terms based on conflicts/solutions using Q-resolution
- Pros: conflicts/solutions can affect the search not restricted to the current path
- Cons: implementation is hard, does not guarantee to perform better than baseline QDLL if the clauses/terms are not learned properly

Game Based Approach

- QBF can be viewed as an and-or two player strategy game (GhostQ 2010-)
  - existential quantifier is the or player (maximizer)
  - universal quantifier is the and player (minimizer)
  - existential quantifier tries to satisfy the formula
  - universal quantifier tries to falsify the formula



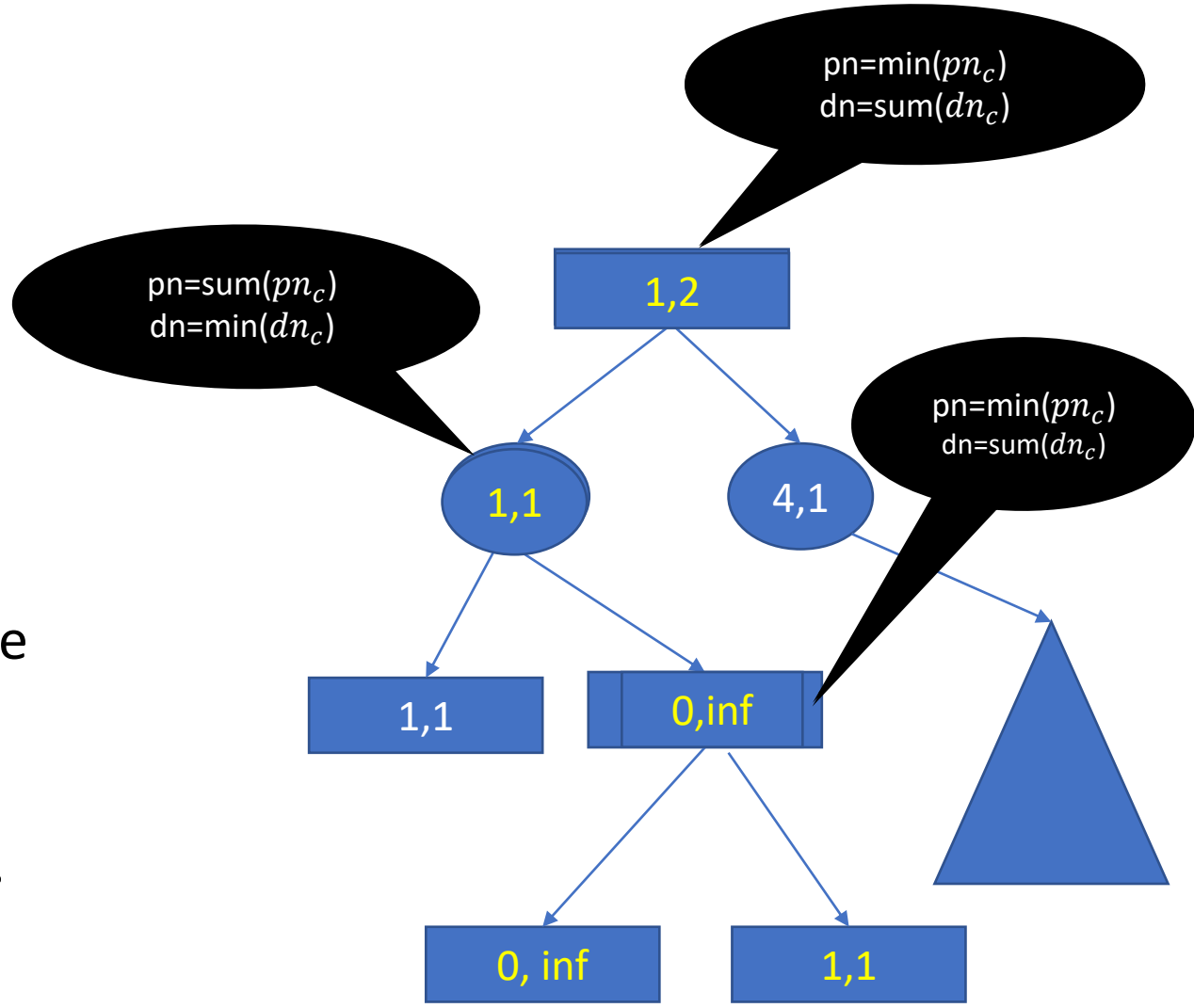
PNS Overview

- Proposed by V. Allis in 1994
- A best first search algorithm that prioritized the search on the direction of the most proving node (MPN)
- Achieved massive success in strategy games

  - Shogi, hex, checkers, Othello

PNS Overview

- Each node stores 2 information:
  - proof number (pn)
  - disproof number (dn)
- Selection (find MPN) -> Expansion -> Initialization -> Backpropagation
- Advantage: not get stuck at one side of the searching space
- Drawback: memory issue, seesaw effect
- Variations: DeepPNS, df-pn, PN2, PN* etc.



Outline

- Technical Background
- Project Aim and Preliminary Results
- Future Plan

Problem Formulation

- **Investigate if proof number search can improve the performance of the depth first search based QBF solvers**
- Combine PNS algorithm and search based QBF solving methods (Backjumping or QCDCL)
- Modify PNS algorithms and let it works well in the QBF world
- Expected result

  - beat state-of-art search based QBF solver DepQBF? Unfeasible

  - PNS is helpful on all benchmarks, unlikely

  - PNS is helpful on some family of benchmarks and analyse the reason

Backjumping modification (Reason computation rules)

- Terminal node:
  - 1. If ϕ_μ contains an empty clause C. The reason for conflict is the set of existential literals in C.
  - 2. If ϕ_μ has all clauses satisfied. The reason for solution is a set of universal literals in μ that can be obtained by repeatedly removing universal literals from μ such that all clauses are still satisfied.
  - 3. If the ϕ_μ contains no universal variables, the satisfiability of ϕ_μ can be determined by calling a SAT solver. The reason is μ. (weaken pruning, e.g. BLOCK family)

Preliminary Results

- DeepPNS + Backjumping
- Experiment: QDLLBJ vs DeepPNSBJ
  - 4 million node expansions
  - SAT solver enabled
- Proof number search can potentially improve the performance of search based QBF solver

Total	DeepPNSBJ	QDLLBJ
95	**47**	32

Table 2: number of solved encoded gttt4x4 (game) instances for DeepPNSBJ and QDLLBJ

Instance	DeepPNSBJ	QDLLBJ
CHAIN12v13	8,239	8,215
CHAIN13v14	16,435	16,409
CHAIN14v15	32,823	32,795
CHAIN16v17	131,135	131,103
CHAIN17v18	262,211	262,177
CHAIN18v19	524,359	524,323
TOILET6.1.iv.11	1,896,123	1,441,537
TOILET6.1.iv.12	**10,073**	218,581
TOILET10.1.iv.20	**10,373**	> 4,000,000
TOILET16.1.iv.32	**11271**	> 4,000,000
Adder-2-S	4,519	3,543
Adder-2-U	45,923	46,335
K_d4_n-1	809	582
K_d4_p-3	**1,482,245**	> 4,000,000
K_lin_p-2	3,522,933	**1,188,617**
K_path_p-4	28,313	**10,927**
K_path_n-3	> 4,000,000	**505,581**
K_t4p_n-1	2,446,447	1,816,773
3qbf-5cnf-50var-500cl.2	**29,053**	> 4000,000

Table 1: number of node expansions for DeepPNSBJ and QDLLBJ on 10 families of benchmarks

Outline

- Technical Background
- Aim of the project and Preliminary Results
- Future Plan

TODO and difficulty

- Improve PNS + Backjumping
  - Reason computation when SAT solver is enabled
  - Seesaw effect of PNS in QBF
- Implement PNS + QCDCL
  - Possible case: PNS + Backjumping + learned clauses
- Two main tasks: PNS + QCDCL and improve PNS + Backjumping
- Difficulty: PNS + QCDCL >> PNS + Backjumping

Plan

Thesis period	Week	Task	
A	9-11	Thesis A report	
B	1-2	QCDCL reading	
	3-6	Investigate computation of reason when SAT solver is activated + seesaw effect of PNS on different families of instances	Implement the data structure for QCDCL
	7-9	Finish PNS + Backjumping	start dfs + QCDCL implementation
	10-11	Thesis B presentation	
C	1	Finish dfs + QCDCL implementation	
	2-6	PNS + QCDCL + experiment	
	7-8	Thesis C presentation	
	9-11	Thesis C report	

Reference

- Cadoli. (1998). An Algorithm to Evaluate Quantified Boolean Formulae *. *AAAI*, 262-267.
- Giunchiglia, E. (2001). Backjumping for quantified boolean logic satisfiability. *IJCAI International Joint Conference on Artificial Intelligence*, 275-281. Retrieved from https://www.scopus.com/record/display.uri?eid=2-s2.0-84880887706&origin=inward&txGid=b61935ab48a6902fd012c2d7b673d36b
- Kishimoto. (2012). Game-Tree Search Using Proof Numbers: The First Twenty Years. *ICGA Journal*, 131-156.