SAT-based Reasoning Techniques for LTL over Finite and Infinite Traces

Jianwen Li

East China Normal University
Main work done at Rice University

July 31, 2022

Linear Temporal Logic

- First introduced to Computer Science by A. Pnueli in 1977
- Formal verification (over infinite traces: LTL)
- AI (over finite traces : LTL_f)

Linear Temporal Logic

Syntax for LTL and LTL_f

$$\phi ::= p \mid \neg\phi \mid \phi \wedge \phi \mid \phi \vee \phi \mid X\phi \mid \phi U \phi \mid \phi R \phi \mid G\phi \mid F\phi;$$

- $\neg(\phi_1 U \phi_2) = \neg\phi_1 R \neg\phi_2$
- $\neg X\phi = N\neg\phi$ (Weak Next), for LTL_f only
- $F\phi = ttU\phi$
- $G\phi = \bot R\phi$

Linear Temporal Logic

Semantics for LTL (LTL_f),

- Let ξ be a trace with $j\xi j = n$ $(n > 0)$
  - $\xi\, j\models p$ if p *2* $\xi[0]$
  - $\xi\, j\models :\phi$ if ξ *j6*$\models \phi$
  - $\xi\, j\models \phi_1 \wedge \phi_2$ if $\xi\, j\models \phi_1$ and $\xi\, j\models \phi_2$
  - $\xi\, j\models X\phi$ if $n > 1$ and $\xi_1\, j\models \phi$
  - $\xi\, j\models \phi_1 U \phi_2$ if (1) there is $0 \quad i < n$ s.t. $\xi_i\, j\models \phi_2$ and (2) for every $0 \quad j < i$ it holds $\xi_j\, j\models \phi_1$.
- LTL semantics: $n = /$
- LTL_f semantics: $n < /$

LTL vs. LTL_f

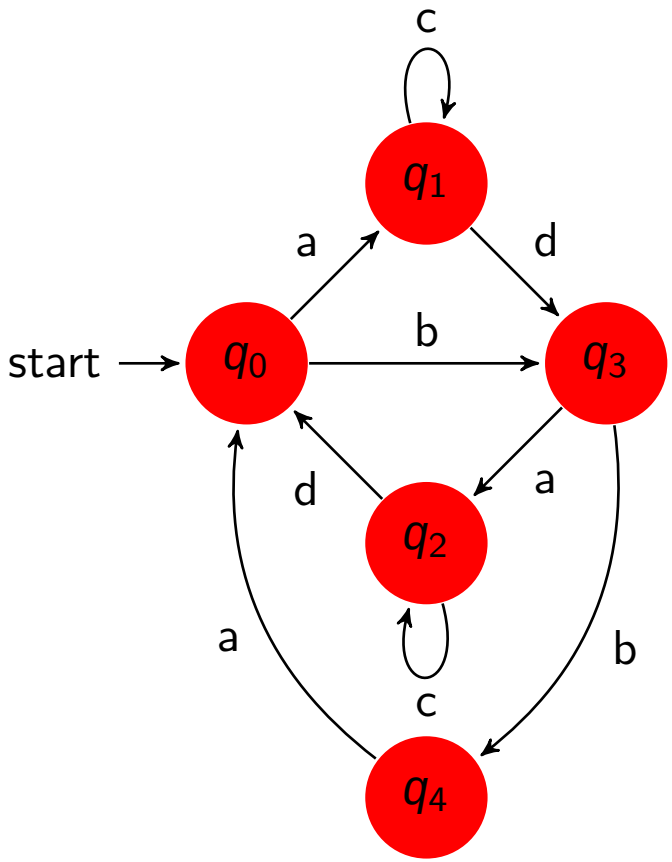
- $X\,tt$ is always true in LTL, but not in LTL_f
- $(a \wedge X\,tt) \not\equiv a$ in LTL_f
- $\neg X\,\phi \not\equiv X \neg \phi$ ($\neg X\,\phi = N \neg \phi$)
- $GX\,\phi$ is never satisfiable in LTL_f

This Talk

- Present an on-the-fly approach to construct automata by SAT solvers
- Show one of its applications to solve LTL_f satisfiability checking problem

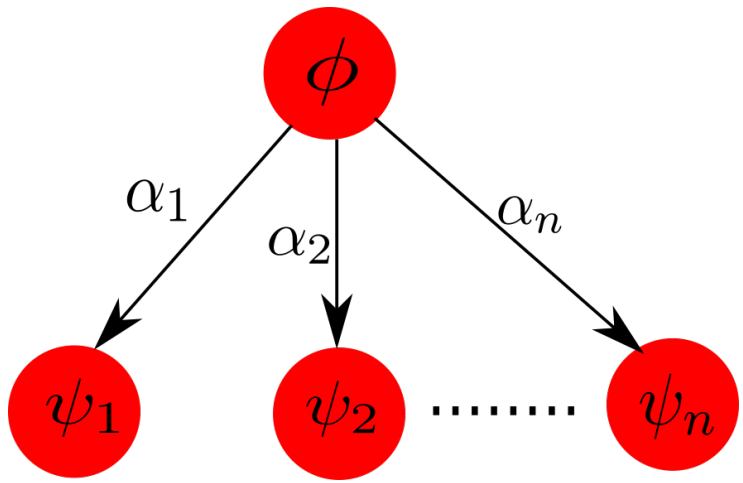
LTL($_f$) to Automata

$\phi \quad =)$



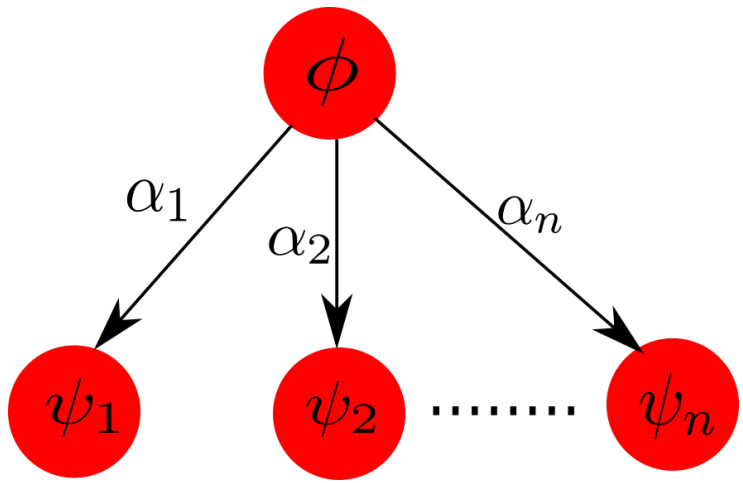
LTL($_f$) to Automata

$\phi \ (\ ) \ \bigvee_i (\alpha_i \wedge X(\psi_i))$



LTL($_f$) to Automata

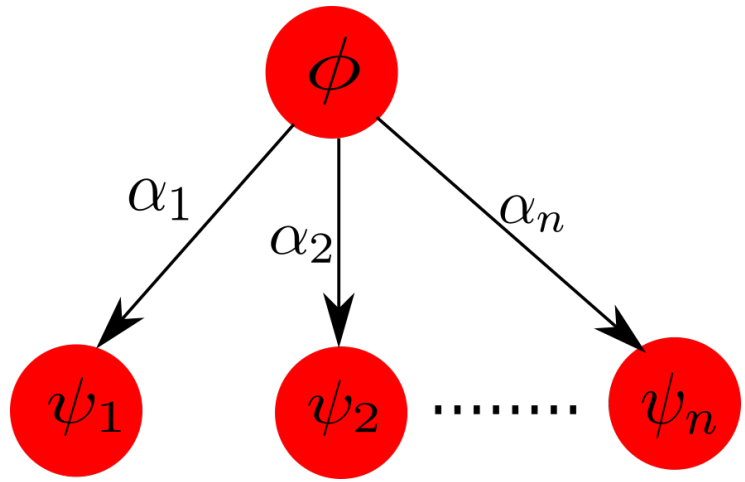
$\phi \ (\ ) \ \bigvee_i (\alpha_i \wedge X(\psi_i))$



Bottleneck: $\phi \ (\ ) \ \bigvee_i (\alpha_i \wedge X(\psi_i))$ is expensive!

LTL($_f$) to Automata

$\phi \ (\ ) \ \bigvee_i(\alpha_i \wedge X(\psi_i))$



Bottleneck: $\phi \ (\ ) \ \bigvee_i(\alpha_i \wedge X(\psi_i))$ is expensive!

Question: Is it possible to generate ONE successor at one time (on the fly)?

neXt Normal Form (XNF)

If considering the temporal subformulas as atoms, an LTL($_f$) formula ϕ becomes a Boolean formula.

We say ϕ is in XNF iff only X/N subformulas can be its atom.

neXt Normal Form (XNF)

Example

- $(a \_ bUc) \wedge cRd$ is not in XNF
- $(a \_ c \_ b \wedge X(bUc) \wedge d \wedge (c \_ X(cRd))$ is in XNF

Invoking SAT solver

1. Input formula ϕ: $(aUb) \wedge (:bU:a)$

Invoking SAT solver

1. Input formula ϕ: $(aUb) \wedge (:bU:a)$
2. $xnf(\phi)$: $(b \vee (a \wedge X(aUb))) \wedge (:a \vee (:b \wedge X(:bU:a)))$

Invoking SAT solver

1. Input formula ϕ: $(aUb) \wedge (\neg bU \neg a)$
2. $xnf(\phi)$: $(b \vee (a \wedge X(aUb))) \wedge (\neg a \vee (\neg b \wedge X(\neg bU \neg a)))$
3. Take $xnf(\phi)$ as the input of a SAT solver

Invoking SAT solver

1. Input formula ϕ: $(aUb) \wedge (\neg bU \neg a)$
2. $xnf(\phi)$: $(b \vee (a \wedge X(aUb))) \wedge (\neg a \vee (\neg b \wedge X(\neg bU \neg a)))$
3. Take $xnf(\phi)$ as the input of a SAT solver
4. SAT solver may return: $\{a, \neg b, X(aUb), X(\neg bU \neg a)\}$
   - A transition in the automaton is $\phi \xrightarrow{a \wedge \neg b} (aUb) \wedge (\neg bU \neg a)$

Invoking SAT solver

1. Input formula ϕ: $(aUb) \wedge (\neg bU \neg a)$
2. $xnf(\phi)$: $(b \vee (a \wedge X(aUb))) \wedge (\neg a \vee (\neg b \wedge X(\neg bU \neg a)))$
3. Take $xnf(\phi)$ as the input of a SAT solver
4. SAT solver may return: $\{a, \neg b, X(aUb), X(\neg bU \neg a)\}$
   - A transition in the automaton is $\phi \xrightarrow{a \wedge \neg b} (aUb) \wedge (\neg bU \neg a)$

To construct the whole automaton, just do enumeration!

LTL_f Satisfiability Checking

Problem

Given an LTL_f formula ϕ, is there a finite trace ξ s.t. $\xi \models \phi$?

- Fa is satisfiable;
- $Fa \wedge G : a$ is unsatisfiable;

Determining Final State

Introduce a new atom *Tail* to associate with X

$$X\psi =) \quad : Tail \wedge X\psi$$

Example

- $b\_X(aUb) =) \;\; b\_ \; : Tail \wedge X(aUb)$
- $XX\phi =) \quad : Tail \wedge X(\, : Tail \wedge X\phi)$

Determining Final State

Theorem

ϕ is a final state iff $\mathit{Tail} \wedge \mathit{xnf}(\phi)$ is satisfiable (Boolean formula).

BLSC: Basic SAT-based Checking

Algorithm 1 BLSC: Basic on-the-fly LTL_f Satisfiability Checking

Require: An LTL_f formula ϕ.

Ensure: SAT or UNSAT.

1: **if** *Tail* ^ *xnf* (ϕ) is satisfiable **then**
2: **return** SAT;
3: **end if**
4: Add ϕ into *block_list*;
5: Let $\psi =$ *xnf* (ϕ) ^ *:X* (*block_list*);
6: **while** ψ is satisfiable **do**
7: Let t be a propositional assignment for ψ;
8: Let $\phi^0 = \bigwedge f\theta jX\theta\ 2\ tg$;
9: **if** BLSC (ϕ^0) returns SAT **then**
10: **return** SAT;
11: **end if**
12: **end while**
13: **return** UNSAT;

BLSC Overview

- BLSC performs very well in small instances, but lacks of scalability

BLSC Overview

- BLSC performs very well in small instances, but lacks of scalability
- Computing states via SAT solvers does not save the space!

BLSC Overview

- BLSC performs very well in small instances, but lacks of scalability
- Computing states via SAT solvers does not save the space!
- Question: Can BLSC be improved?

BLSC Overview

- BLSC performs very well in small instances, but lacks of scalability
- Computing states via SAT solvers does not save the space!
- Question: Can BLSC be improved?
  - BLSC leverages the satisfiability from SAT solvers

BLSC Overview

- BLSC performs very well in small instances, but lacks of scalability
- Computing states via SAT solvers does not save the space!
- Question: Can BLSC be improved?
  - BLSC leverages the satisfiability from SAT solvers
  - BLSC discards the unsatisfiability from SAT solvers

CDLSC: Conflict-Driven Checking

Tail ^ *xnf* (ϕ) is UNSAT

$=)$

there is ψ s.t ψ ϕ[1] and *Tail* ^ *xnf* (ψ) is still unsatisfiable
(UNSAT core)

[1] Consider have the form of $\bigwedge_i$

CDLSC: Conflict-Driven Checking

$$\textit{Tail} \wedge \textit{xnf}(\phi) \text{ is UNSAT}$$

$$\Longrightarrow$$

there is ψ s.t ψ ϕ[1] and $\textit{Tail} \wedge \textit{xnf}(\psi)$ is still unsatisfiable (UNSAT core)

- ψ can be provided by a SAT solver
- ψ represents a set of states that cannot reach a final state in 0 step

[1] Consider have the form of $\bigwedge_i$

CDLSC: Conflict-Driven Checking

Conflict Sequence: $C = C_0, C_1, C_2, \ldots, C_k (k \quad 0)$

- C_i : a set of states that cannot reach a final state in i steps

CDLSC: Conflict-Driven Checking

Conflict Sequence: $C = C_0, C_1, C_2, \ldots, C_k (k \quad 0)$

- C_i : a set of states that cannot reach a final state in i steps
- Guided search:
  - ψ is known in some C_i $=)$ a next state of ψ not in C_i is preferred: $xnf(\phi) \wedge : XC_i$
  - Otherwise, check final state: $Tail \wedge xnf(\psi)$

CDLSC: Conflict-Driven Checking

Conflict Sequence: $C = C_0, C_1, C_2, \ldots, C_k (k \quad 0)$

- C_i : a set of states that cannot reach a final state in i steps
- Guided search:
  - ψ is known in some C_i $=)$ a next state of ψ not in C_i is preferred: $xnf(\phi) \wedge : XC_i$
  - Otherwise, check final state: $Tail \wedge xnf(\psi)$
- Unsatisfiability check: $\bigcap_{0 \quad j \quad i} C_j \quad C_{i+1}$

Evaluation

- Benchmarks: 7446 LTL-as-LTL_f formulas in previous work
- Platform: Rice Davinci cluster
- Timeout: 60 seconds for each instance
- Tools:
  1. Aalta-finite [LZPVH14]
  2. Aalta-infinite [LZPV15]
  3. ltl2sat [FG16]
  4. nuXmv (IC3+KLIVE) [CCDGMMMRT14]
  5. BLSC
  6. CDLSC

Evaluation

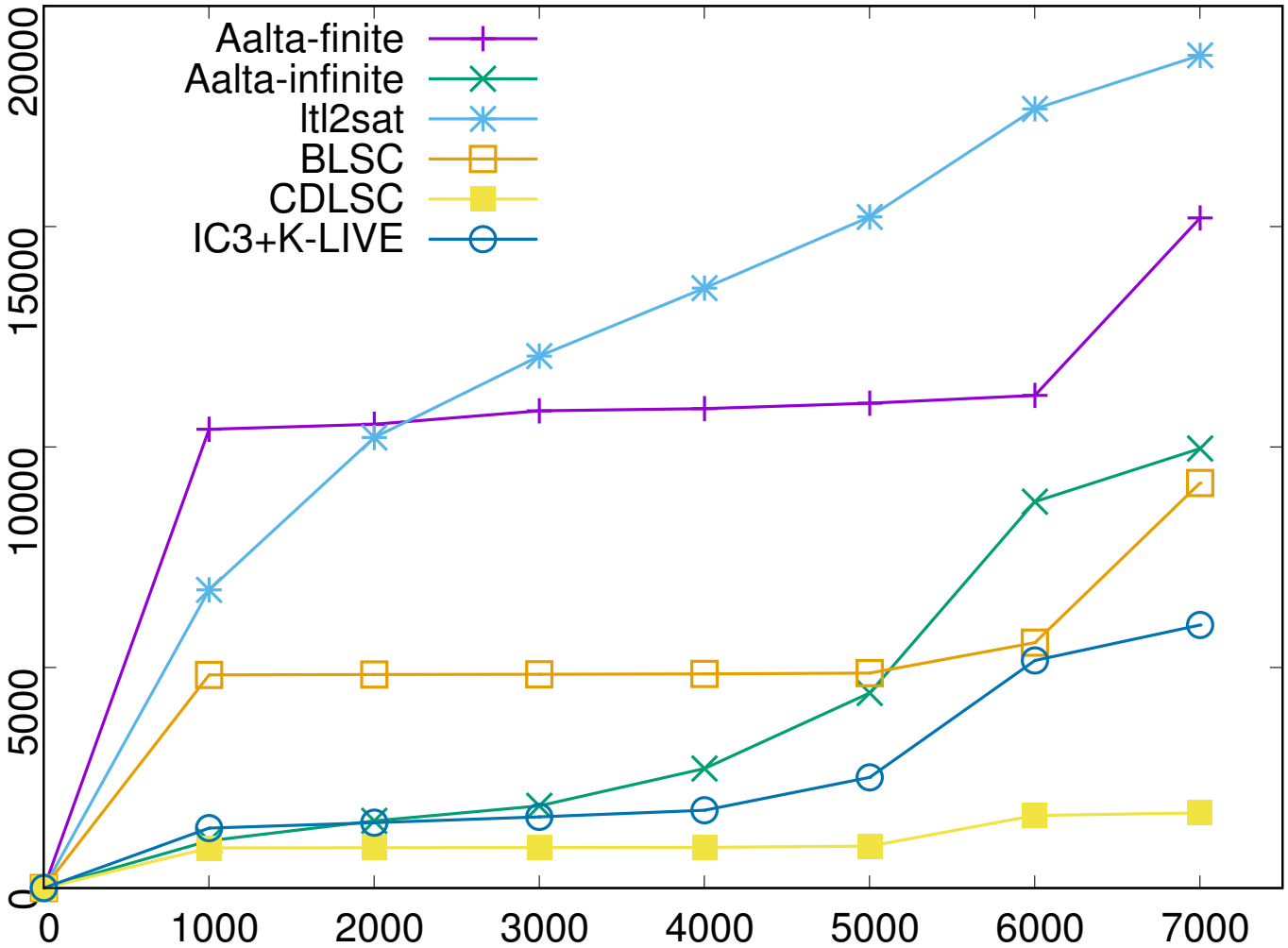


Figure 1: Cactus plot for LTL_f Satisfiability Checking on LTL-as-LTL_f Benchmarks.

Beyond Satisfiability Checking

- On-the-fly Synthesis for LTL over finite traces [AAAI2019]
- Satisfiability checking for LTL over infinite traces[FMSD 2019]
- Synthesis for LTL over infinite traces?
- On-the-fly LTL model checking?
- Extension to word-level?

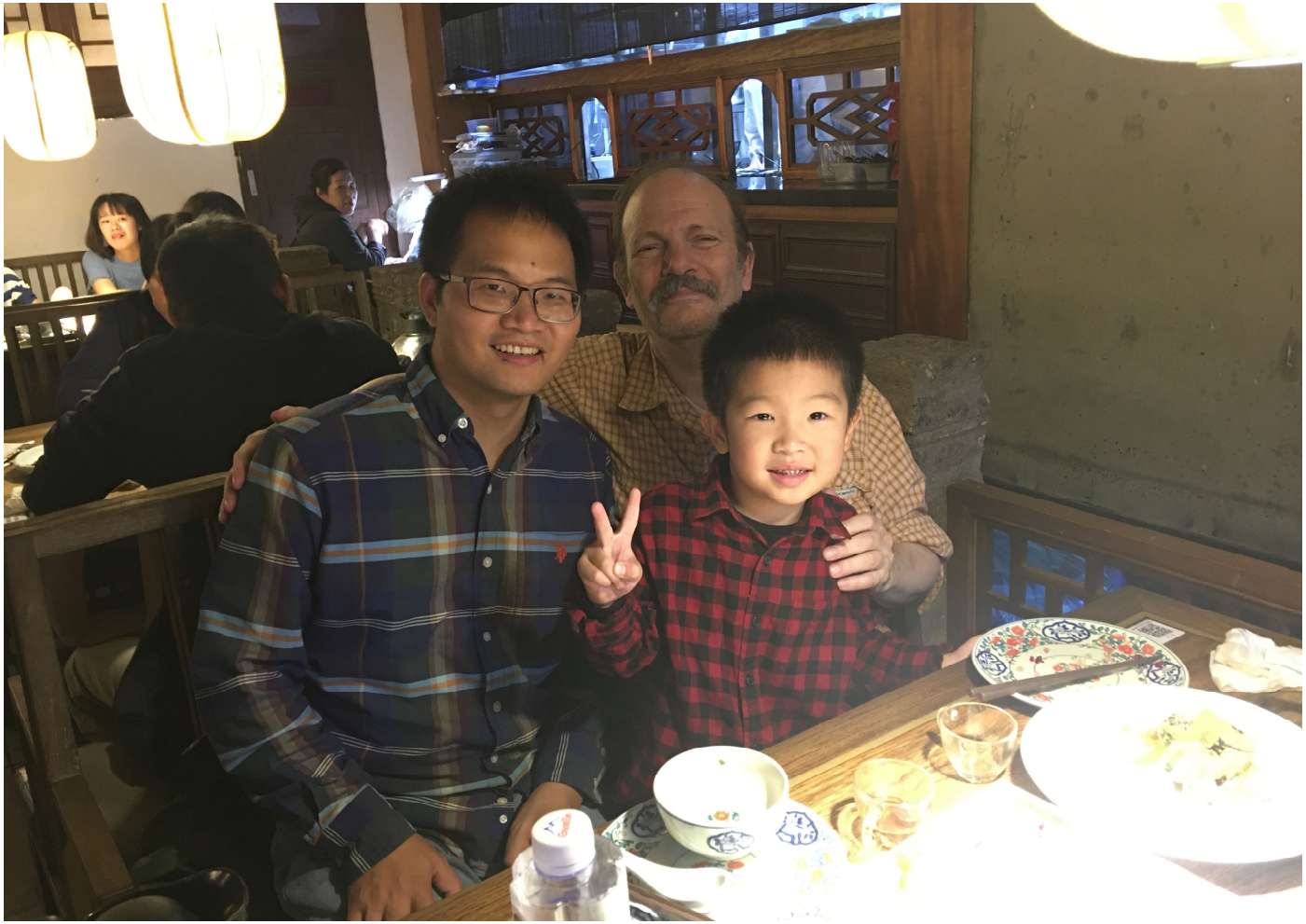


Figure 2: Shanghai, Oct. 2019.

Hope to see you again VERY soon!