VLDL Satisfiability and ~~Model Checking~~ via Tree Automata

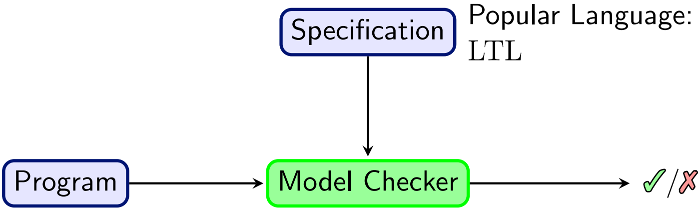
Alexander Weinert

Saarland University

December 12th, 2017

FSTTCS 2017 - Kanpur, India

Setting: Program Verification



An Example

```
def f():
  if(/*...*/):
    cd("folder")
  else:
    cd("..")

def main():
  /*...*/
```

"Program never leaves its original working directory"

Not expressible in LTL!

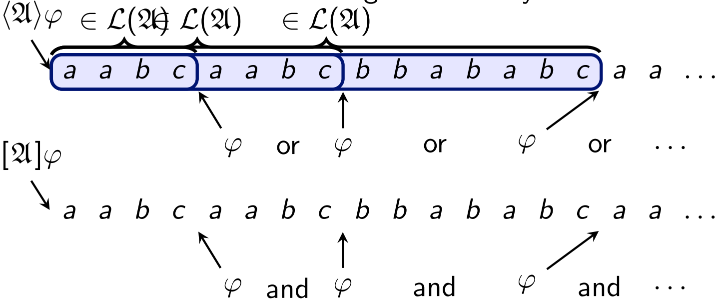
Visibly Linear Dynamic Logic (VLDL)[1]

Syntax of VLDL:

guarded eventually

$$\varphi = p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \langle\mathfrak{A}\rangle\varphi \mid [\mathfrak{A}]\varphi$$

guarded always



[1](W. and Zimmermann, 2016)

Visibly Pushdown Automata[2]

Visibly Pushdown Automata are restricted Pushdown Automata

$$\overbrace{\boxed{\text{Calls}}\boxed{\text{Local Actions}}\boxed{\text{Returns}}}^{\Sigma}$$

- When reading call, automaton has to push onto the stack
- When reading return, automaton has to pop off the stack
- When reading local action, automaton has to ignore the stack

$\Rightarrow$ Closed under intersection!

[2] (Alur and Madhusudan, 2005)

VLDL Complexity[3]

VLDL: Extension of LTL,
temporal operators guarded by visibly pushdown automata

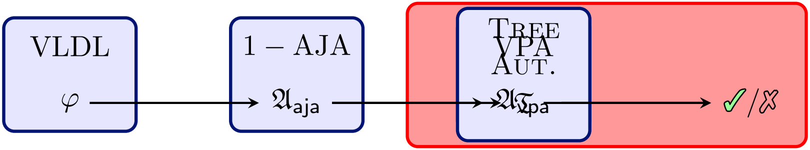
Satisfiability	ExpTime-complete
Model Checking	ExpTime-complete
Games	3ExpTime-complete

Contribution:
Novel, conceptually simple algorithms

[3] (W. and Zimmermann, 2016)

Theorem (W. and Zimmermann, 2016)

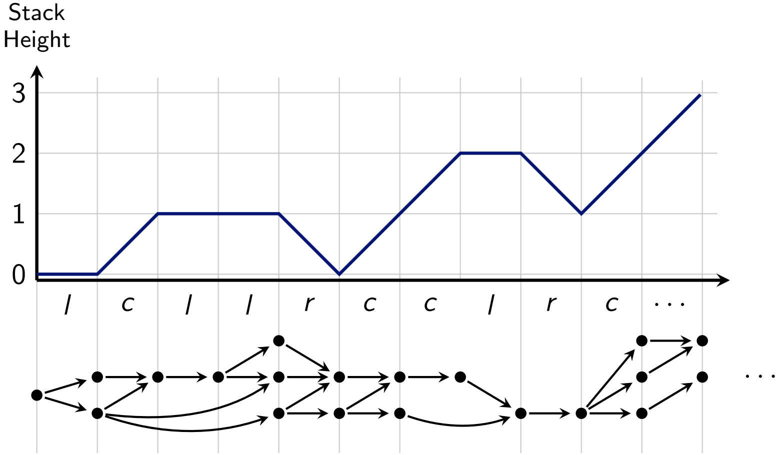
VLDL *Satisfiability is* EXPTIME-*complete.*



1. What are $1 - \mathrm{AJAs}$?
2. How to translate $1 - \mathrm{AJAs}$ into tree automata?

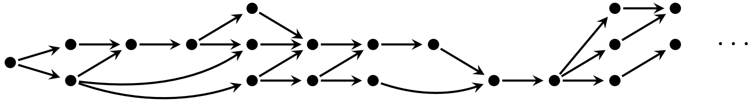
Intermediate Automata: 1 − AJA[4]

Extension of alternating automata: $\delta(q, a) = q_1 \wedge (q_2 \vee q_3)$.



[4](Bozzelli, 2007)

Intermediate Automata: $1 - \mathrm{AJA}$



Acceptance: Each branch visits accepting states infinitely often.

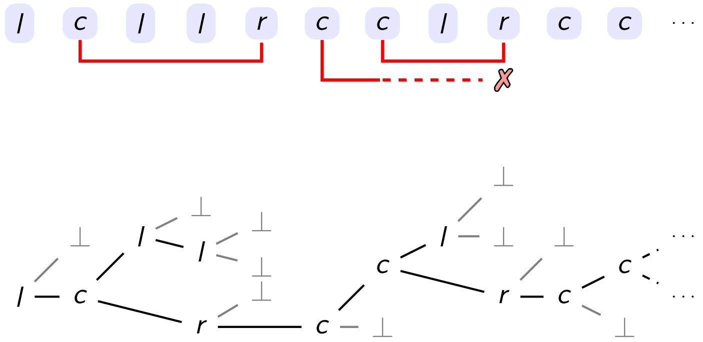
Theorem (Ext. of (W. and Zimmermann, 2016))

For each VLDL *formula there exists an equivalent* $1 - \mathrm{AJA}$ *of polynomial size.*

Guiding Questions

1. What are $1 - \mathrm{AJAs}$? ✓
2. How to translate $1 - \mathrm{AJAs}$ into tree automata?
   - How to translate words into trees?

Encoding Words as Trees

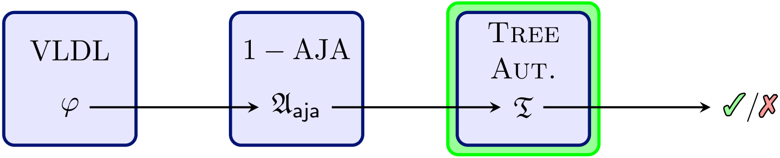


Adapted from (Alur and Madhusudan, 2004)

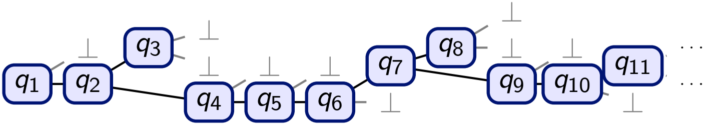
Guiding Questions

1. What are $1 - \mathrm{AJAs}$? ✓
2. How to translate $1 - \mathrm{AJAs}$ into tree automata?
   - How to translate words into trees? ✓

Overview



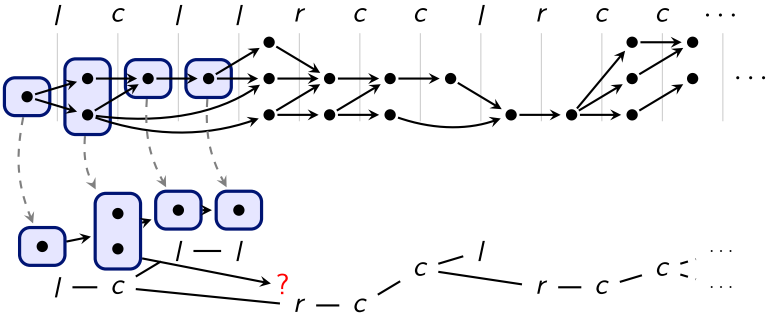
Tree Automata



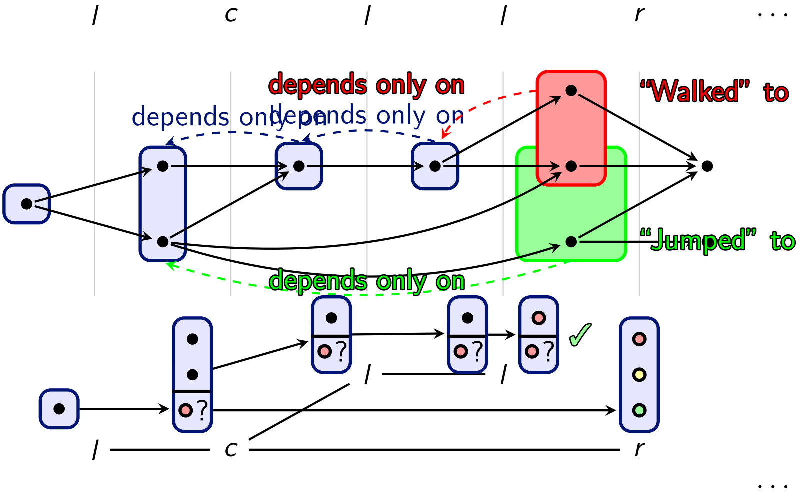
Acceptance: Every branch has infinitely many accepting vertices

Component	Technique
States	?
Transitions	?
Accepting States	?

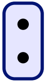
Translating 1 – AJAs into Tree Automata

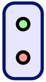


Construction in Detail



Summary of Construction

1. Subset construction 

2. Split into "jump"- and "walk"-states 

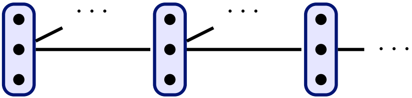
3. Guess and verify 

Constructing a Tree Automaton

Component	Technique	
States	Subset Construction	✓
Transitions	Guess/Verify	✓
Accepting States	?	

Accepting States

Run of the tree automaton:



Recall:

- $1 - \mathrm{AJA}$ accepts if all paths are accepting.
- Tree automaton accepts if accepting states are visited infinitely often.

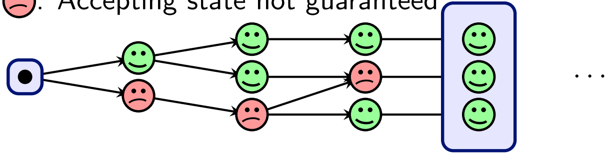
Solution: Lift acceptance condition from paths to states

$\Rightarrow$ Breakpoint Technique (Miyano and Hayashi, 1984)

Breakpoint Construction

: Accepting state guaranteed

: Accepting state not guaranteed



Breakpoint: All paths since last breakpoint visit accepting state

Lemma (Miyano and Hayashi, 1984)

A run of an alternating automaton is accepting if there exists a breakpoint sequence over it.

Lemma

A run of a $1 - \mathrm{AJA}$ *is accepting if there exists a breakpoint sequence** *over it.*

Accepting States

- Keep track of 🙂 and 🙁 states in state space
- Update 🙂 and 🙁 states on the fly
- Accept and restart if all states are 🙂

Accepting States ✓

Constructing a Tree Automaton

Component	Technique	
States	Subset Construction	✓
Transitions	Guess/Verify	✓
Accepting States	Breakpoint Construction	✓

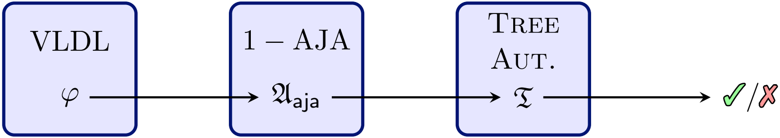
Theorem

For every VLDL formula φ we can construct a tree automaton $\mathfrak{T}$ of exponential size that recognizes the same language.

Guiding Questions

1. What are $1 - \mathrm{AJAs}$? ✓
2. How to translate $1 - \mathrm{AJAs}$ into tree automata? ✓
   - How to translate words into trees? ✓

New Approach



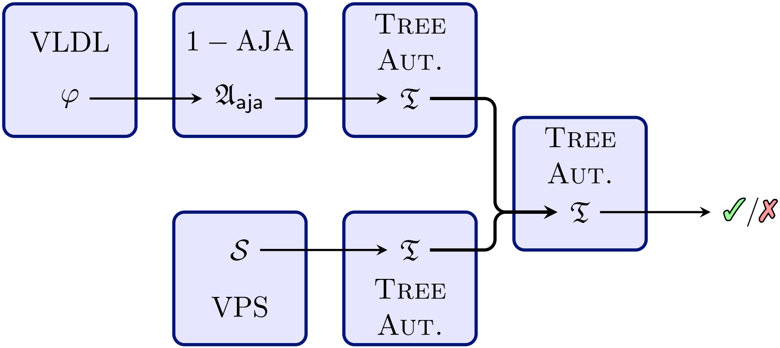
Lemma

The following problem is in PTime*: "Given a tree automaton* $\mathfrak{T}$*, does* $\mathfrak{T}$ *recognize the empty language?"*

Theorem

The following problem is in ExpTime*: "Given a* VLDL *formula* φ*, does* φ *define the empty language?"*

Model Checking



Theorem

The following problem is in EXPTIME*: "Given a* VLDL *formula* φ *and a visibly pushdown system* $\mathcal{S}$*, do all traces of* $\mathcal{S}$ *satisfy* φ*?"*

Conclusion

Conclusion

- Connection between visibly pushdown words and stack trees
- Breakpoint technique is very versatile
- Putting VLDL on solid algorithmic foundation of Büchi games

Future Work

- Games with VLDL winning conditions
- Prototypical Implementation, Comparison