VLDL Satisfiability and Model Checking via Tree Automata

Alexander Weinert

Saarland University

December 12th, 2017

FSTTCS 2017 - Kanpur, India

VLDL Satisfiability and Model Checking via Tree Automata

Alexander Weinert

Saarland University

December 12th, 2017

FSTTCS 2017 - Kanpur, India

VLDL Satisfiability and Model Checking via Tree Automata

Alexander Weinert

Saarland University

December 12th, 2017

FSTTCS 2017 - Kanpur, India

VLDL Satisfiability and Model Checking via Tree Automata

Alexander Weinert

Saarland University

December 12th, 2017

FSTTCS 2017 - Kanpur, India

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

Alexander Weinert

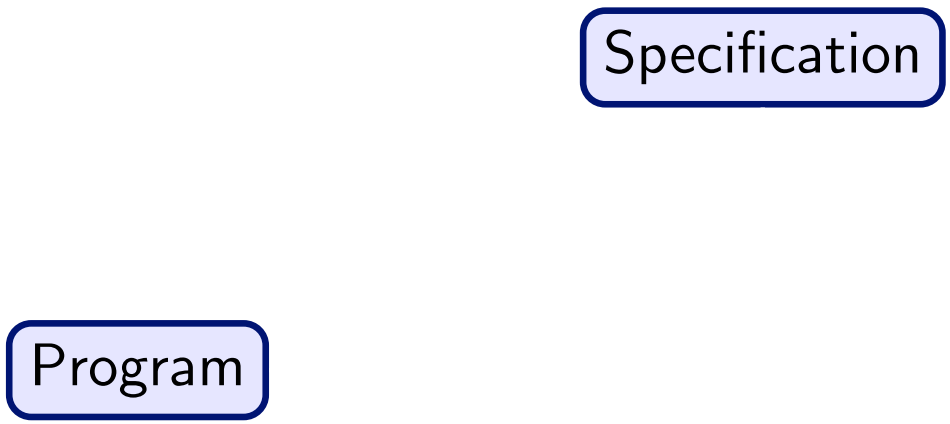
Saarland University

December 12th, 2017

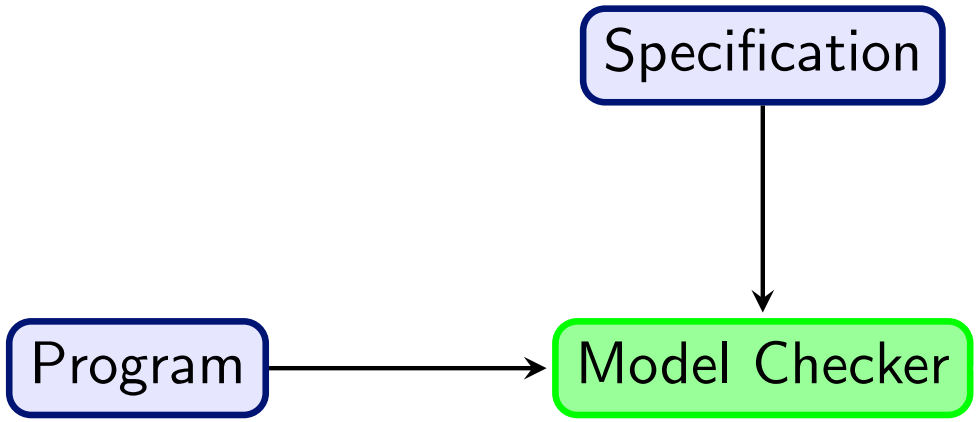
FSTTCS 2017 - Kanpur, India

Setting: Program Verification

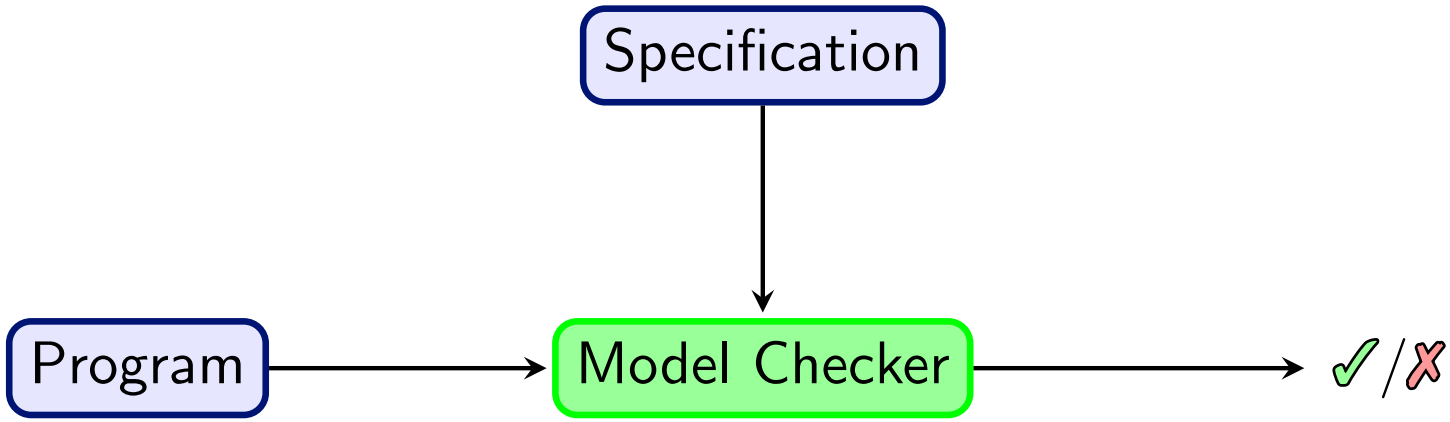
Setting: Program Verification



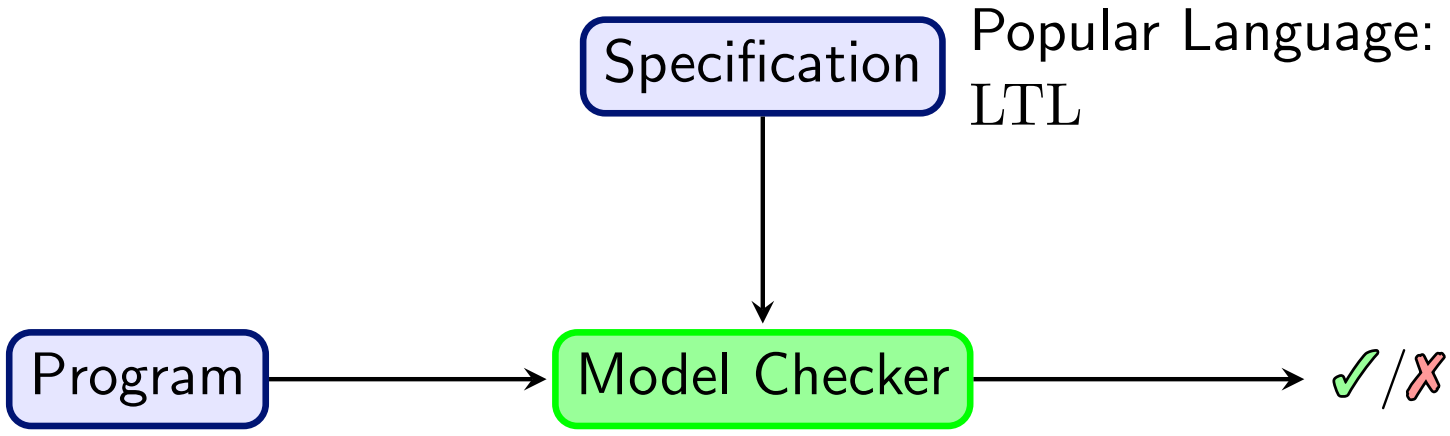
Setting: Program Verification



Setting: Program Verification



Setting: Program Verification



An Example

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

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

An Example

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

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

"Program never leaves its original working directory"

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:

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

[1](W. and Zimmermann, 2016)

Visibly Linear Dynamic Logic (VLDL)[1]

Syntax of VLDL:

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

[1](W. and Zimmermann, 2016)

Visibly Linear Dynamic Logic (VLDL)[1]

Syntax of VLDL:

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

[1](W. and Zimmermann, 2016)

Visibly Linear Dynamic Logic (VLDL)[1]

Syntax of VLDL:

guarded eventually

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

[1](W. and Zimmermann, 2016)

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 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

$$a \quad a \quad b \quad c \quad a \quad a \quad b \quad c \quad b \quad b \quad a \quad b \quad a \quad b \quad c \quad a \quad a \quad \ldots$$

[1] (W. and Zimmermann, 2016)

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

$\langle\mathfrak{A}\rangle\varphi$

$a \;\; a \;\; b \;\; c \;\; a \;\; a \;\; b \;\; c \;\; b \;\; b \;\; a \;\; b \;\; a \;\; b \;\; c \;\; a \;\; a \;\; \ldots$

[1] (W. and Zimmermann, 2016)

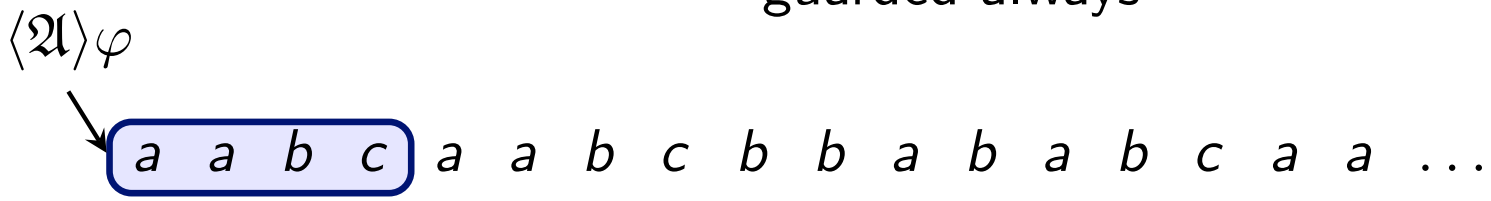
Visibly Linear Dynamic Logic (VLDL)[1]

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)

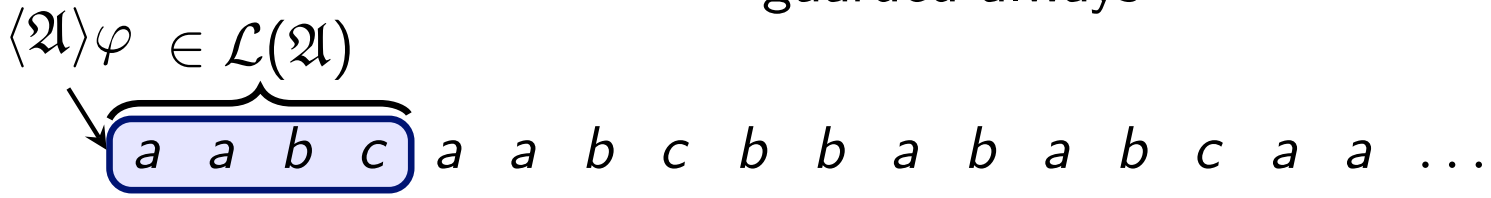
Visibly Linear Dynamic Logic (VLDL)[1]

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)

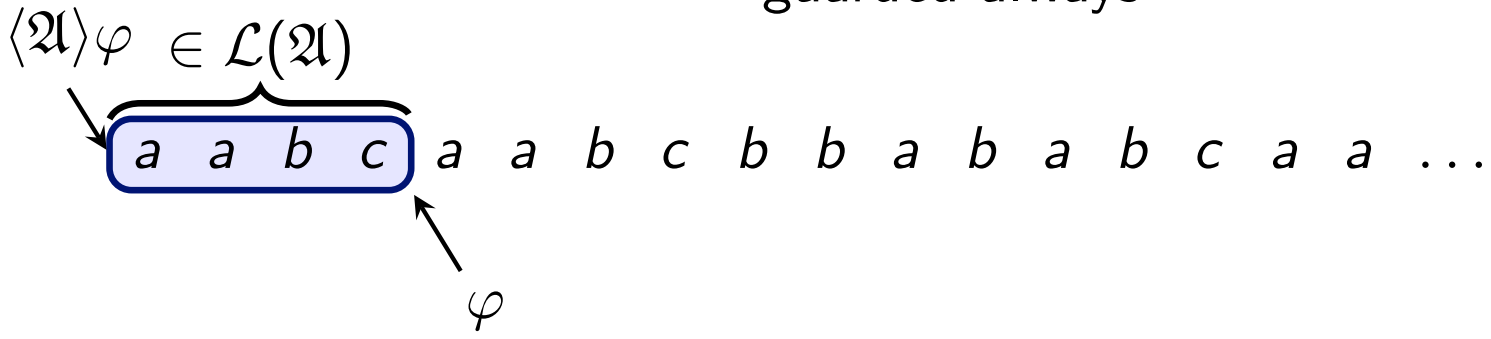
Visibly Linear Dynamic Logic (VLDL)[1]

Syntax of VLDL:

guarded eventually

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

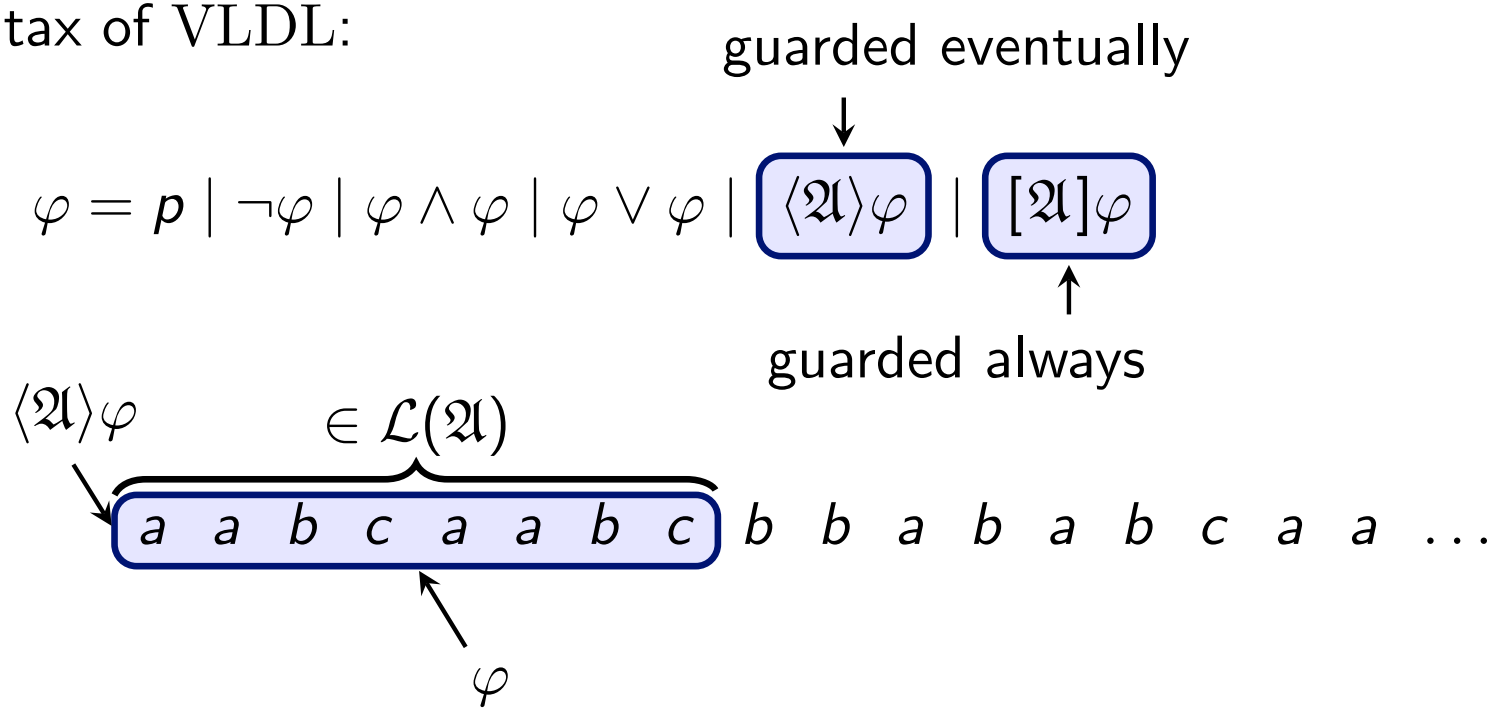
guarded always



[1](W. and Zimmermann, 2016)

Visibly Linear Dynamic Logic (VLDL)[1]

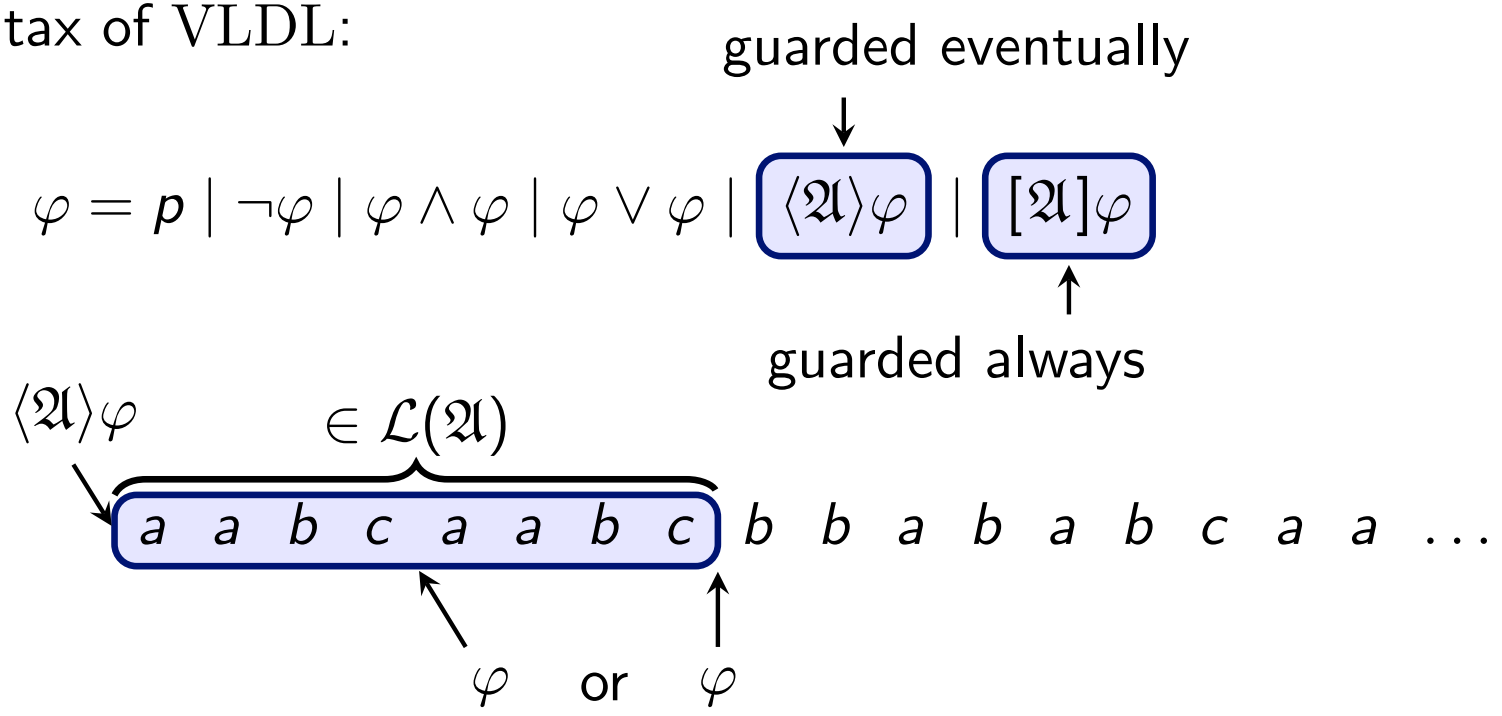
Syntax of VLDL:



[1](W. and Zimmermann, 2016)

Visibly Linear Dynamic Logic (VLDL)[1]

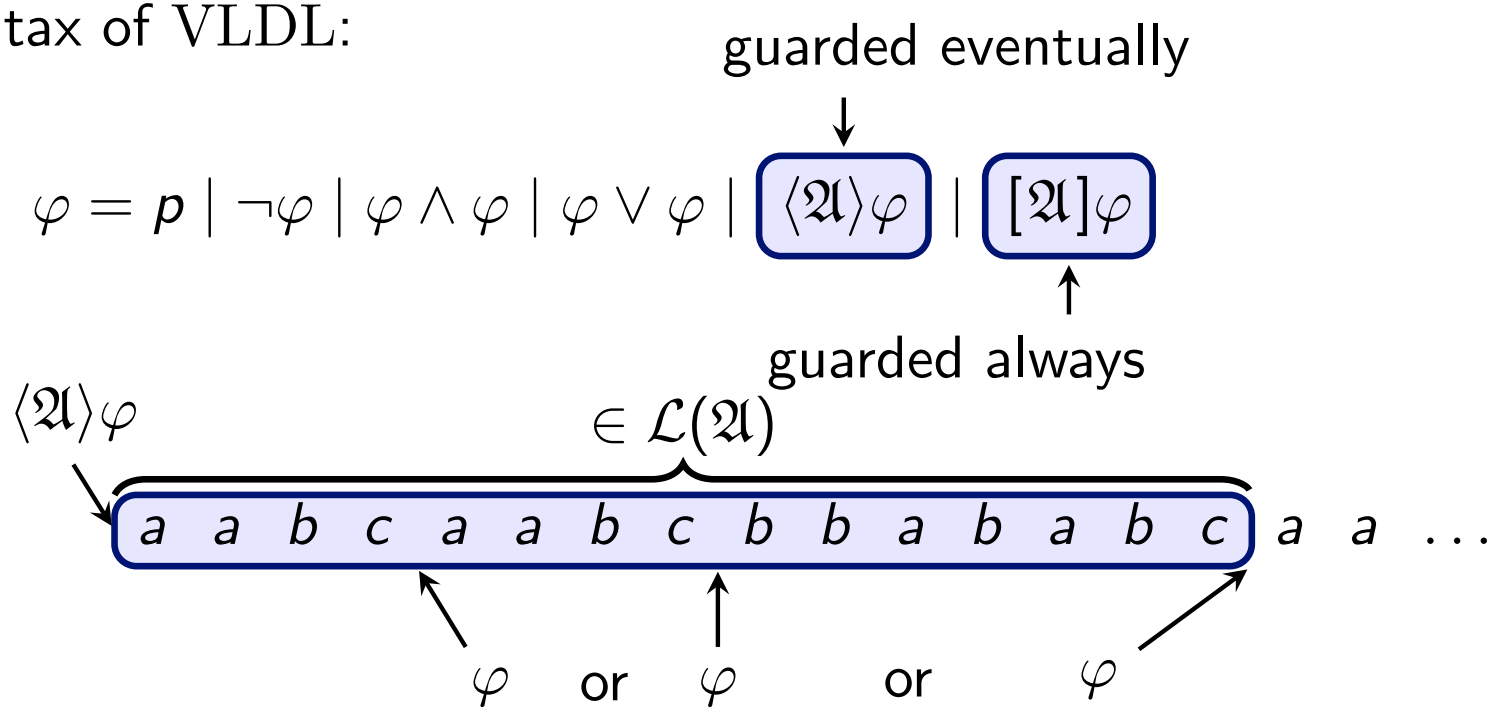
Syntax of VLDL:



[1](W. and Zimmermann, 2016)

Visibly Linear Dynamic Logic (VLDL)[1]

Syntax of VLDL:



[1](W. and Zimmermann, 2016)

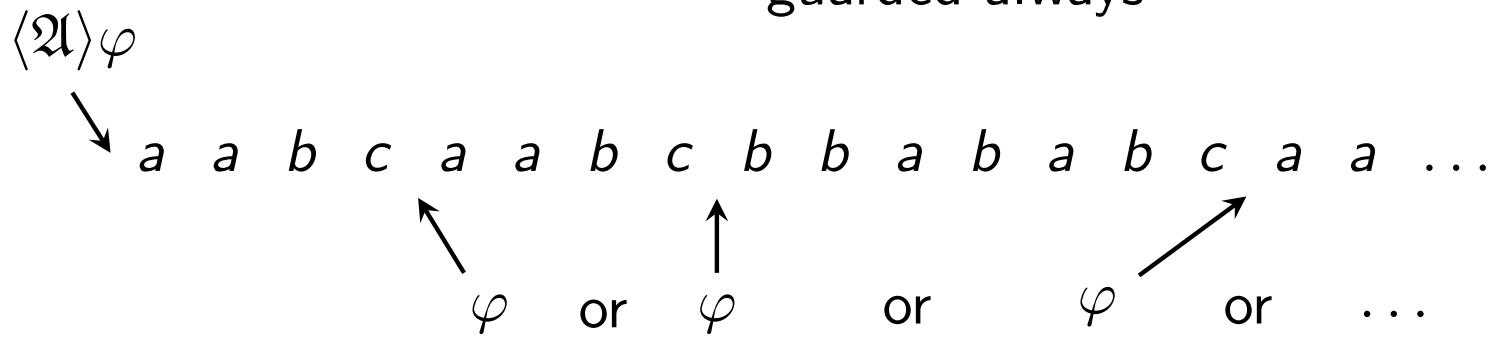
Visibly Linear Dynamic Logic (VLDL)[1]

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)

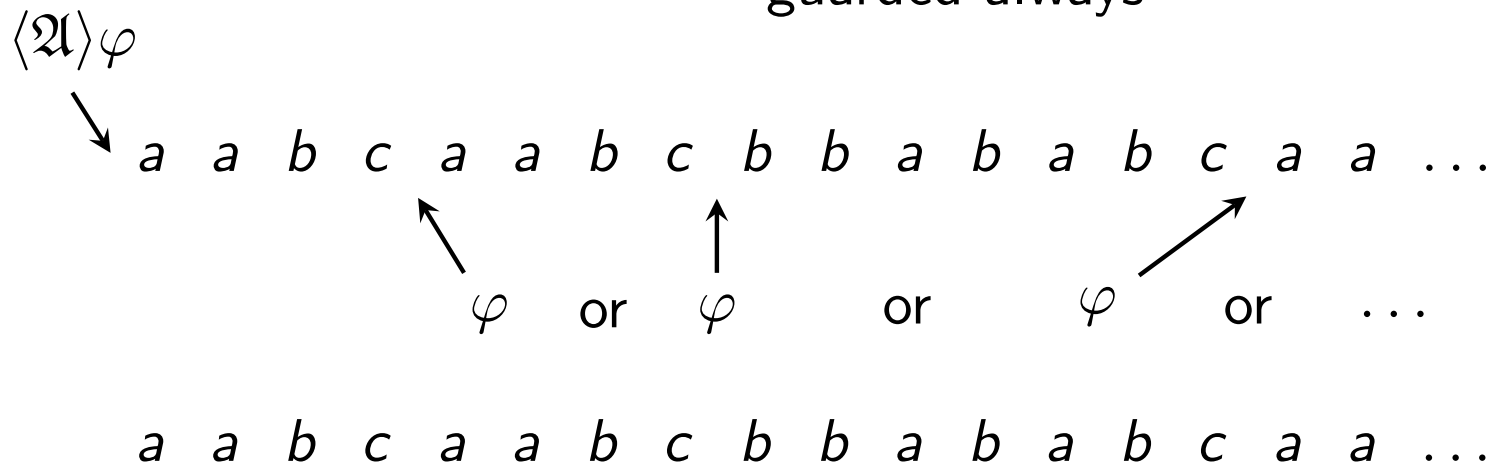
Visibly Linear Dynamic Logic (VLDL)[1]

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)

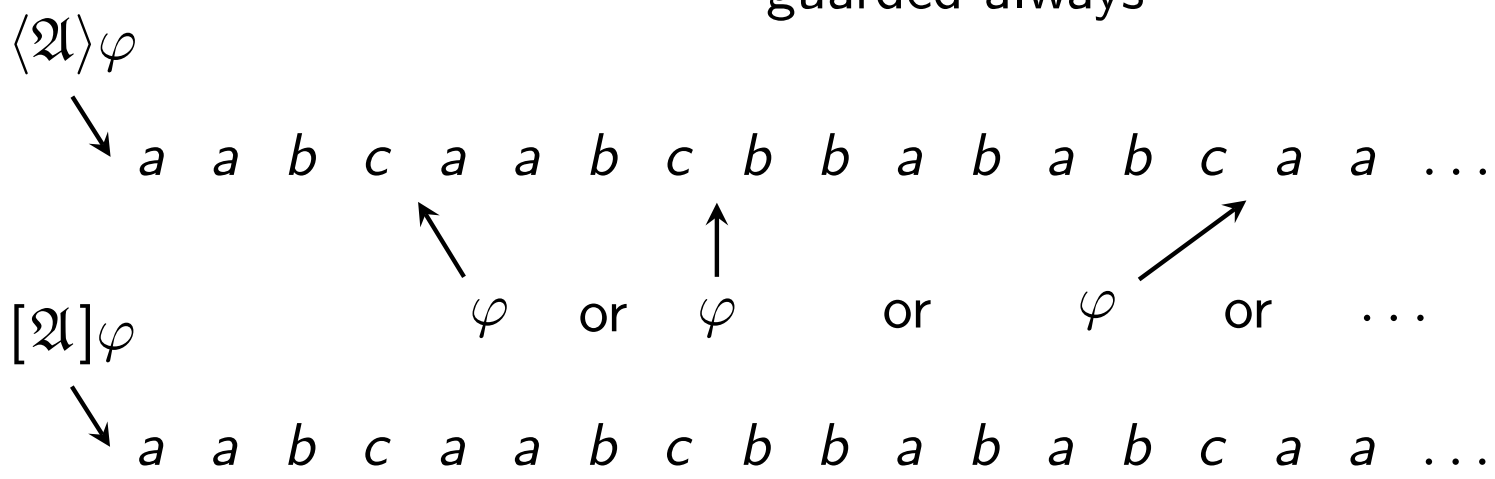
Visibly Linear Dynamic Logic (VLDL)[1]

Syntax of VLDL:

guarded eventually

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

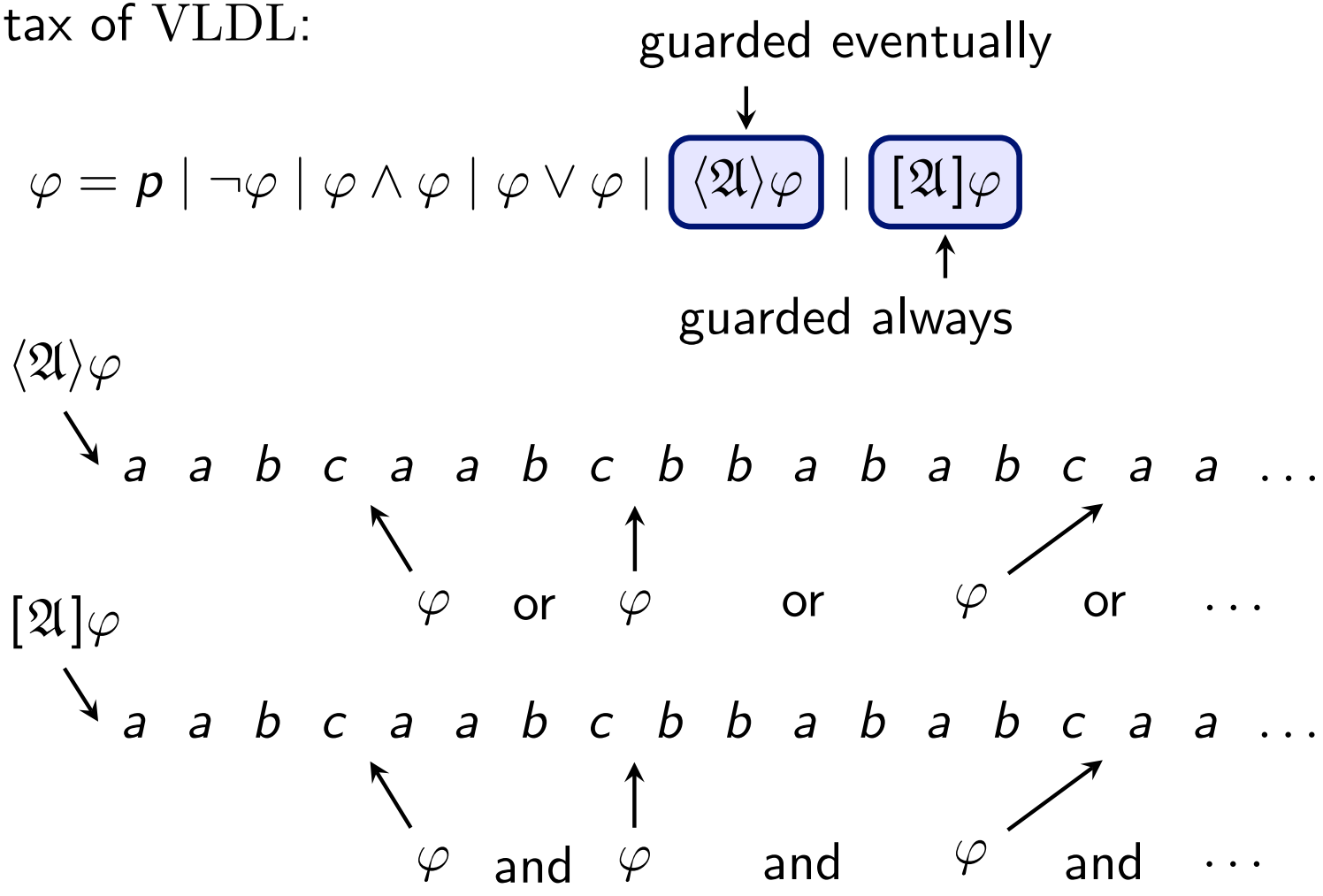
guarded always



[1] (W. and Zimmermann, 2016)

Visibly Linear Dynamic Logic (VLDL)[1]

Syntax of VLDL:



[1](W. and Zimmermann, 2016)

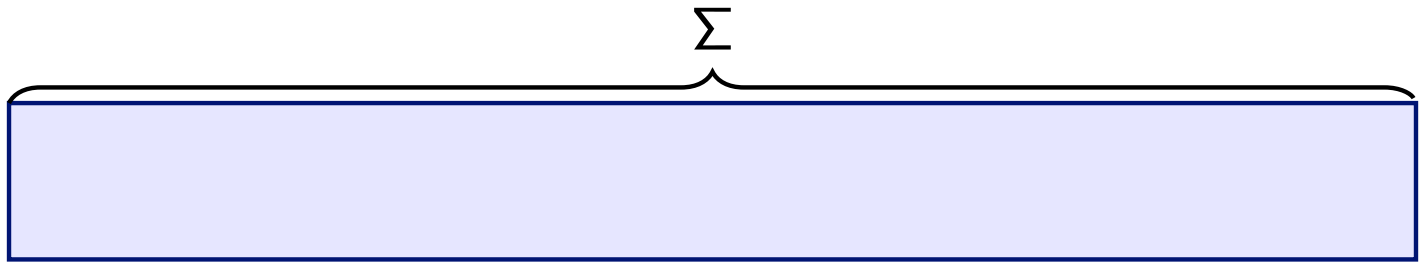
Visibly Pushdown Automata[2]

Visibly Pushdown Automata are restricted Pushdown Automata

[2](Alur and Madhusudan, 2005)

Visibly Pushdown Automata[2]

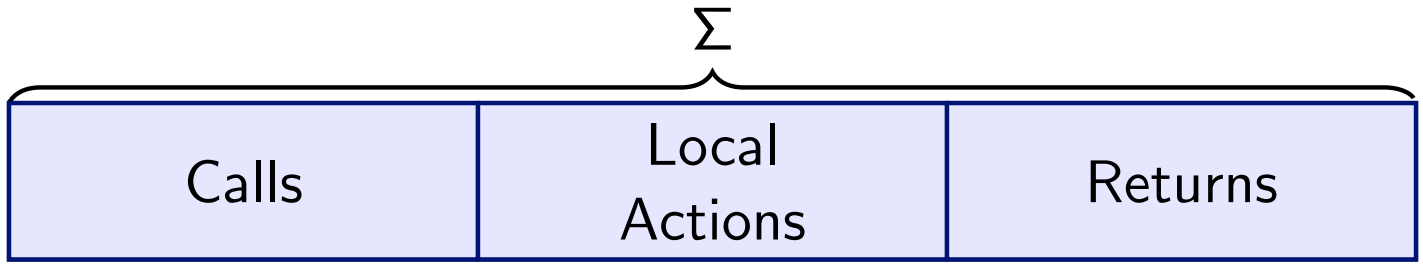
Visibly Pushdown Automata are restricted Pushdown Automata



[2](Alur and Madhusudan, 2005)

Visibly Pushdown Automata[2]

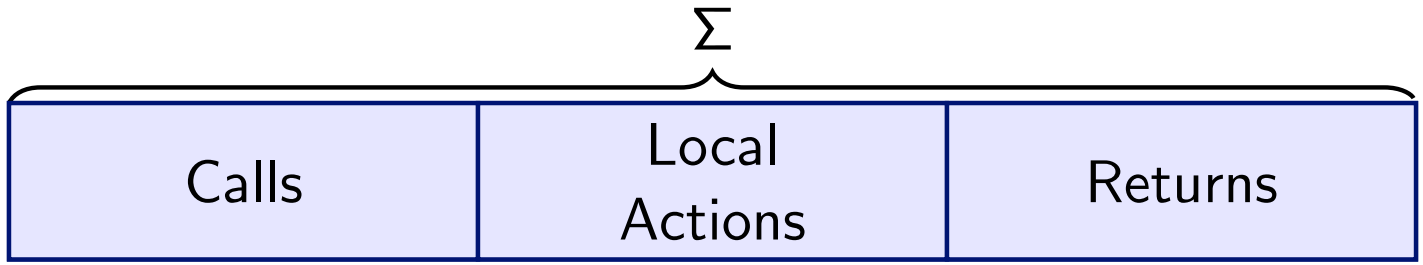
Visibly Pushdown Automata are restricted Pushdown Automata



[2](Alur and Madhusudan, 2005)

Visibly Pushdown Automata[2]

Visibly Pushdown Automata are restricted Pushdown Automata

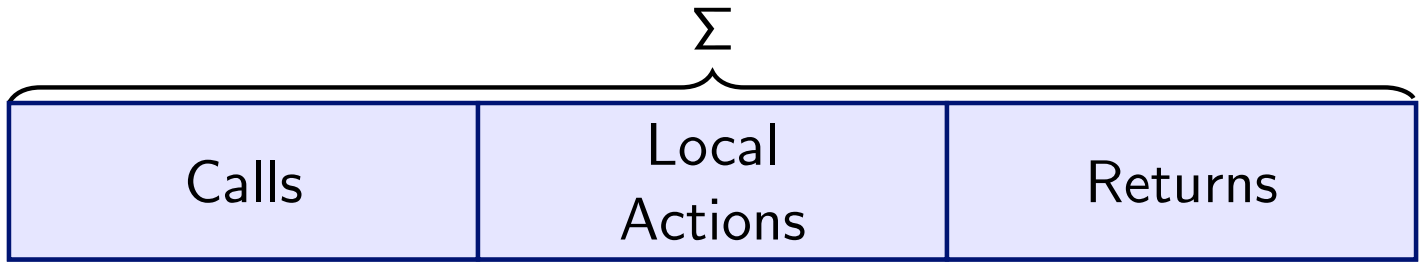


- When reading call, automaton has to push onto the stack

[2](Alur and Madhusudan, 2005)

Visibly Pushdown Automata[2]

Visibly Pushdown Automata are restricted Pushdown Automata

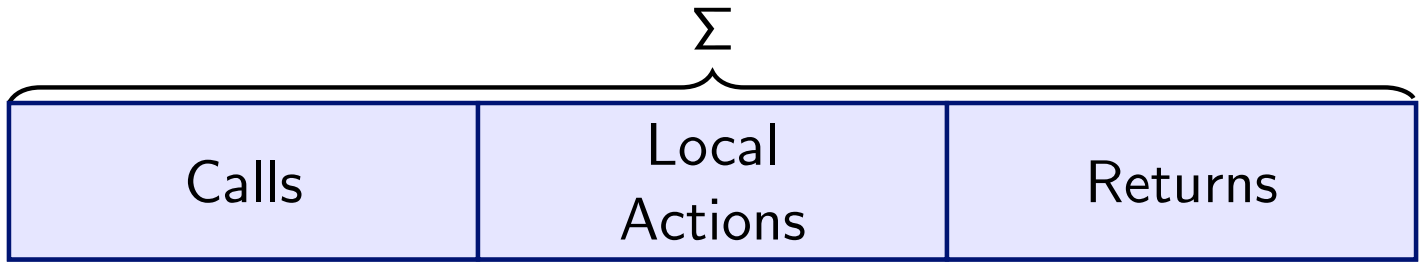


- When reading call, automaton has to push onto the stack
- When reading return, automaton has to pop off the stack

[2](Alur and Madhusudan, 2005)

Visibly Pushdown Automata[2]

Visibly Pushdown Automata are restricted Pushdown Automata



- 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

[3](W. and Zimmermann, 2016)

VLDL Complexity[3]

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

Satisfiability	ExpTime-complete

[3] (W. and Zimmermann, 2016)

VLDL Complexity[3]

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

Satisfiability	ExpTime-complete
Model Checking	ExpTime-complete

[3](W. and Zimmermann, 2016)

VLDL Complexity[3]

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

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

[3](W. and Zimmermann, 2016)

VLDL Complexity[3]

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

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

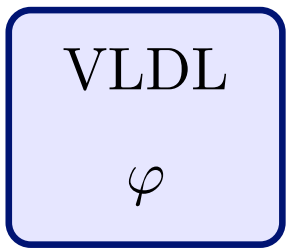
Contribution:
Novel, conceptually simple algorithms

[3](W. and Zimmermann, 2016)

VLDL Satisfiability and Model Checking

Theorem (W. and Zimmermann, 2016)

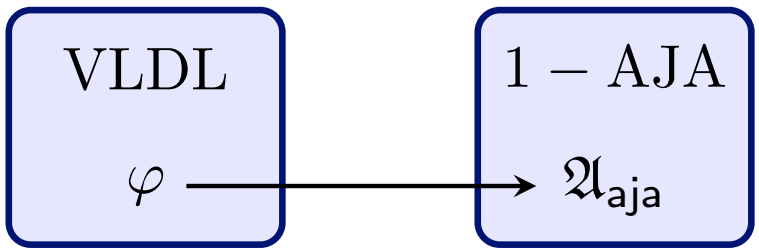
VLDL *Satisfiability is* ExpTime*-complete.*



VLDL Satisfiability and Model Checking

Theorem (W. and Zimmermann, 2016)

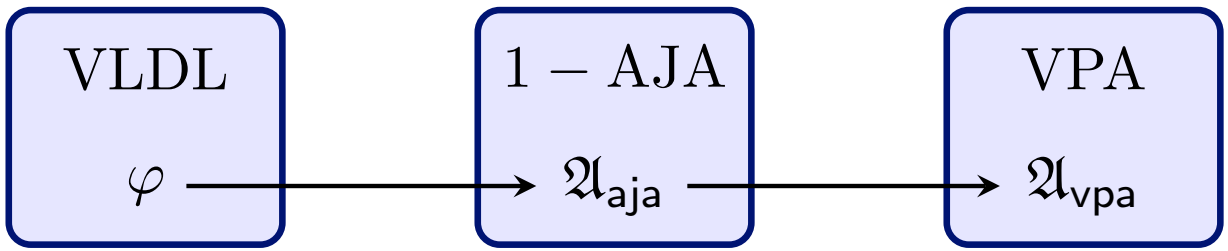
VLDL *Satisfiability is* ExpTime*-complete.*



VLDL Satisfiability and Model Checking

Theorem (W. and Zimmermann, 2016)

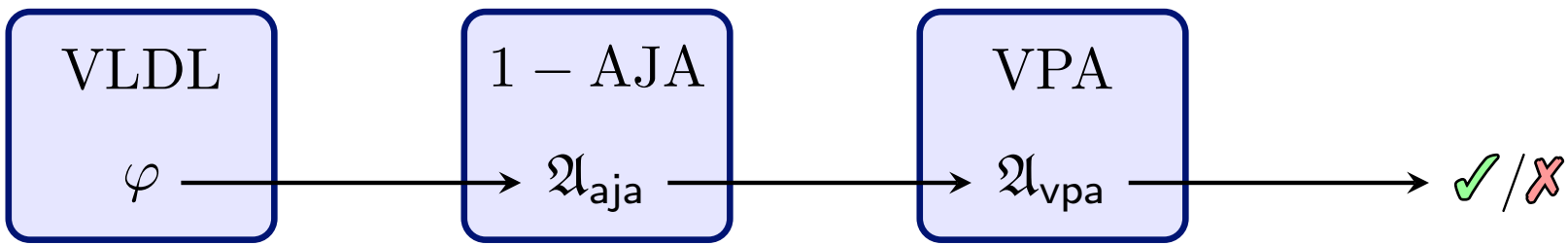
VLDL *Satisfiability is* ExpTime*-complete.*



VLDL Satisfiability and Model Checking

Theorem (W. and Zimmermann, 2016)

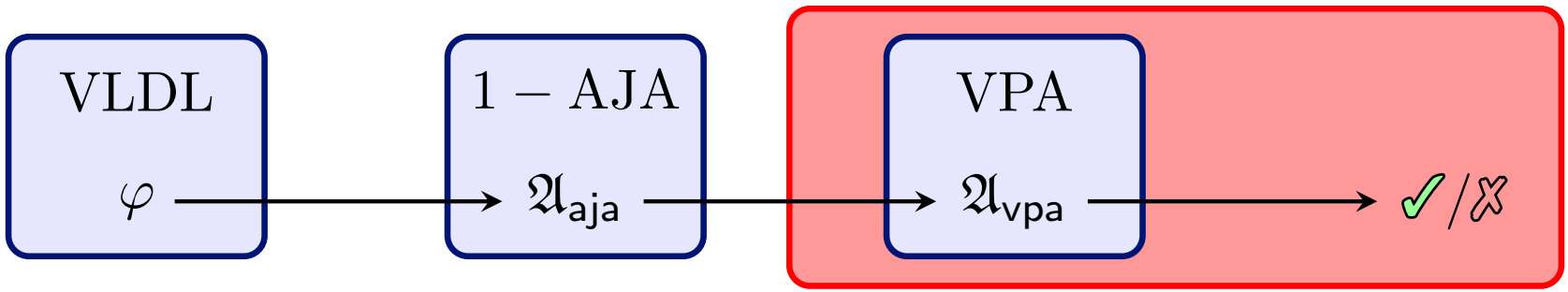
VLDL *Satisfiability is* ExpTime*-complete.*



VLDL Satisfiability and Model Checking

Theorem (W. and Zimmermann, 2016)

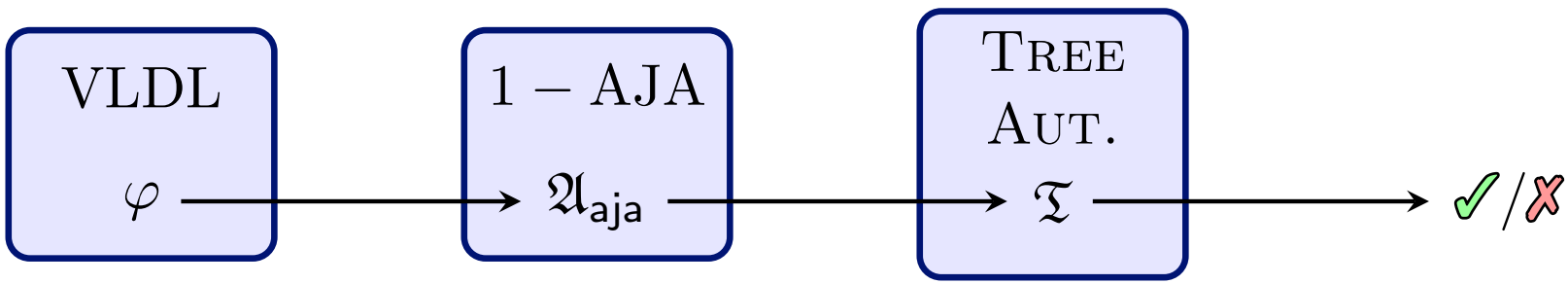
VLDL *Satisfiability is* ExpTime*-complete.*



VLDL Satisfiability and Model Checking

Theorem (W. and Zimmermann, 2016)

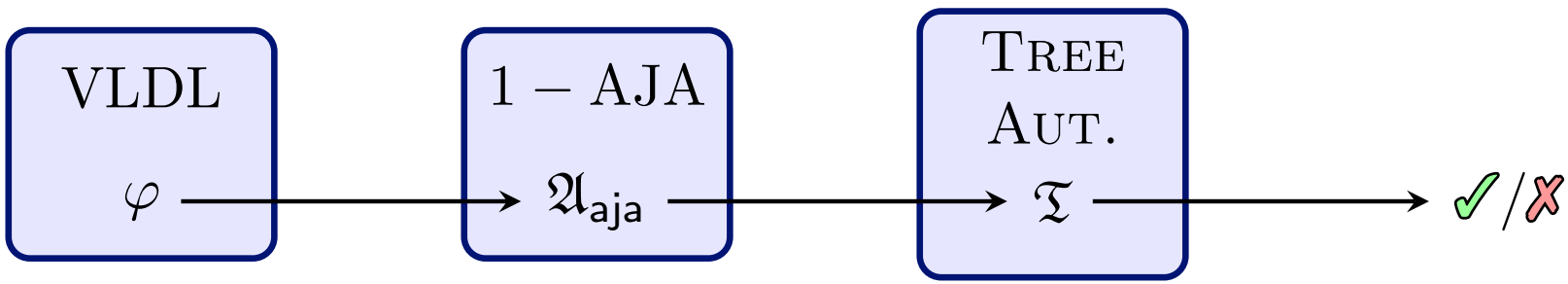
VLDL *Satisfiability is* ExpTime*-complete.*



VLDL Satisfiability and Model Checking

Theorem (W. and Zimmermann, 2016)

VLDL *Satisfiability is* ExpTime*-complete.*

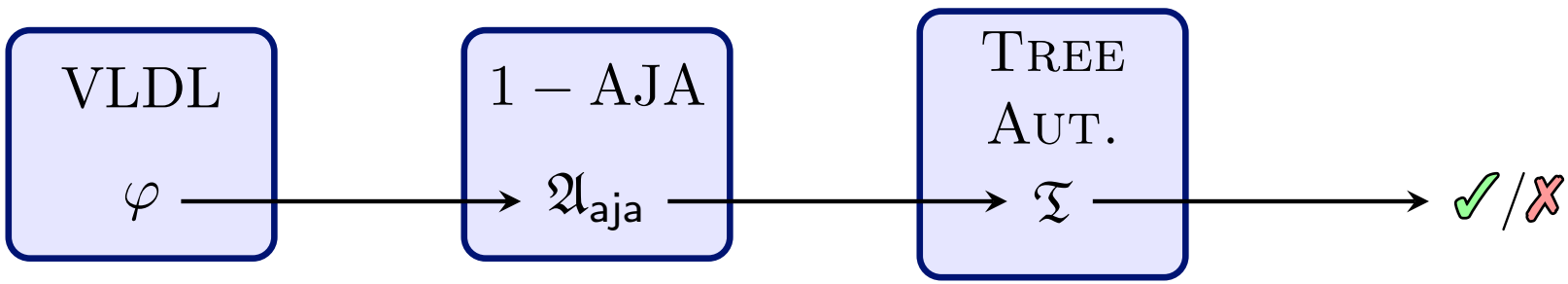


1. What are $1-\text{AJAs}$?

VLDL Satisfiability and Model Checking

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]

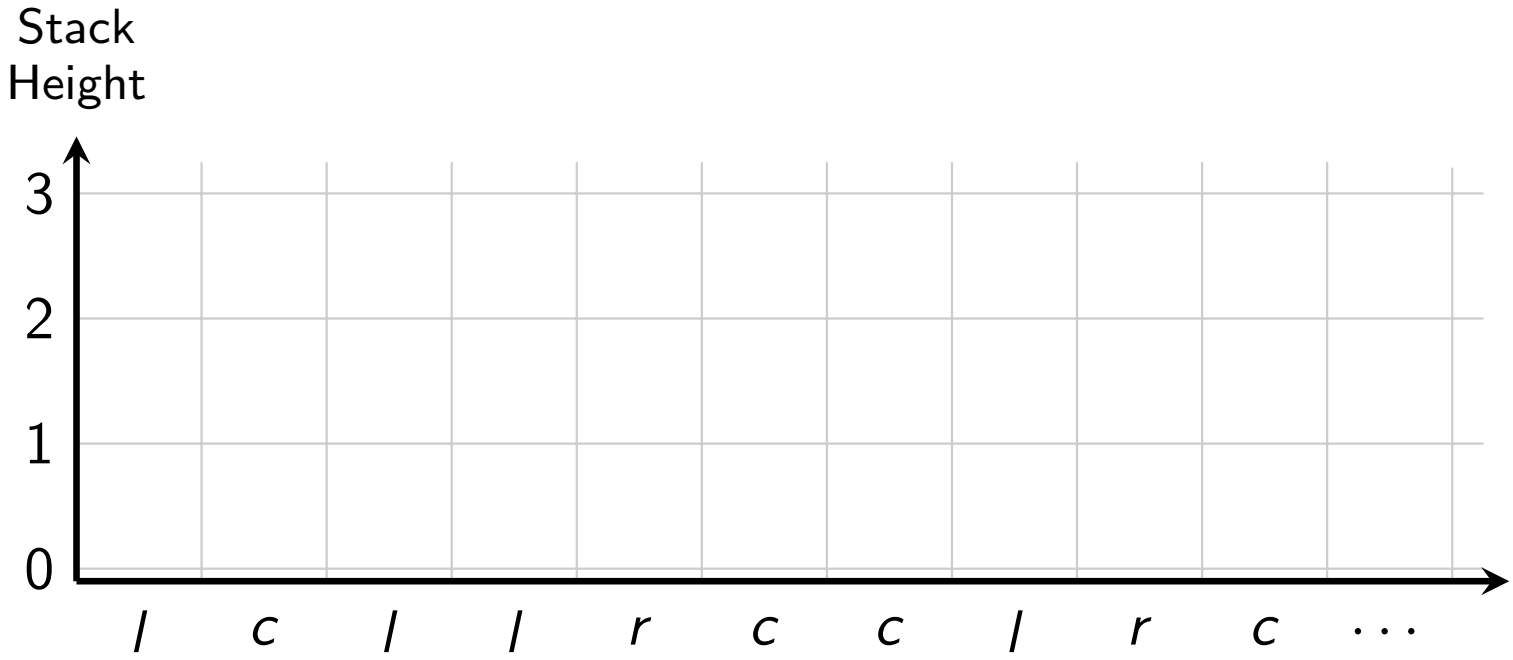
Extension of alternating automata: $\delta(q, a) = q_1 \wedge (q_2 \vee q_3)$.

$$l \quad c \quad l \quad l \quad r \quad c \quad c \quad l \quad r \quad c \quad \cdots$$

[4](Bozzelli, 2007)

Intermediate Automata: 1 – AJA[4]

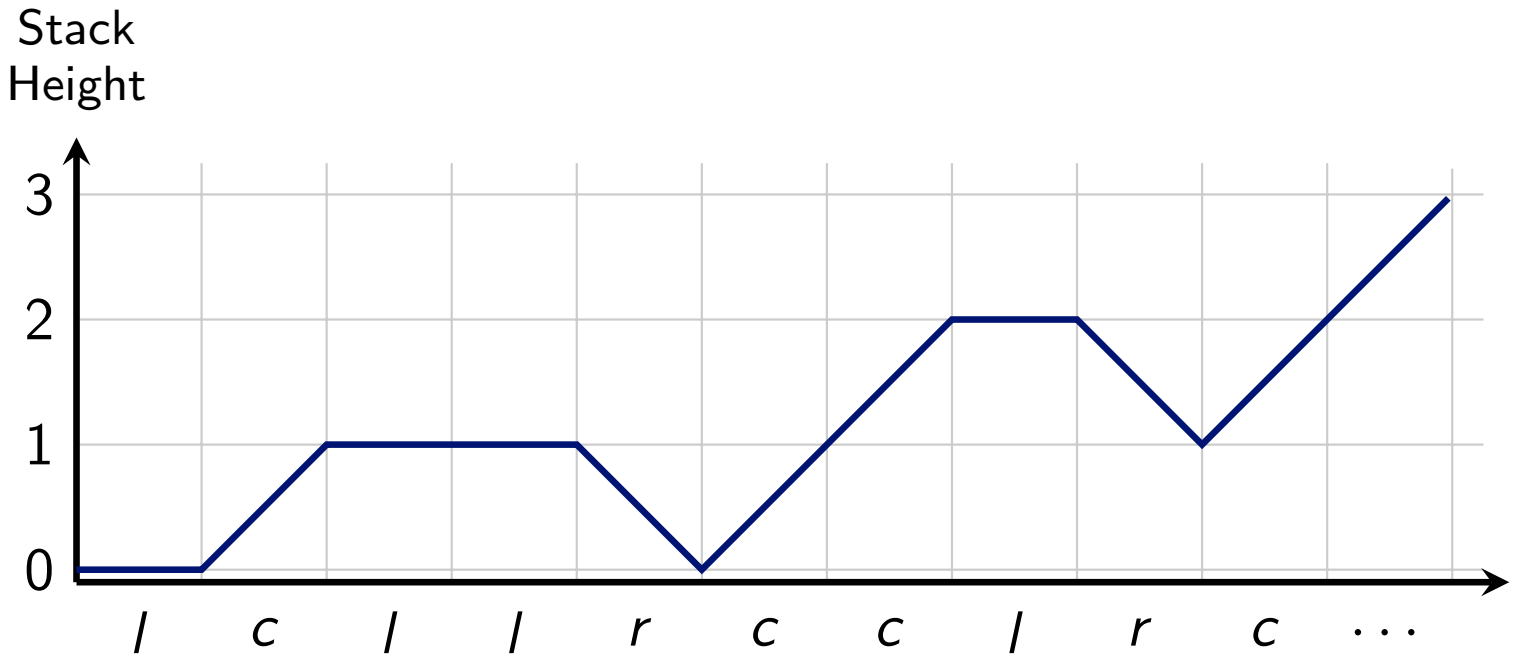
Extension of alternating automata: $\delta(q, a) = q_1 \wedge (q_2 \vee q_3)$.



[4](Bozzelli, 2007)

Intermediate Automata: 1 – AJA[4]

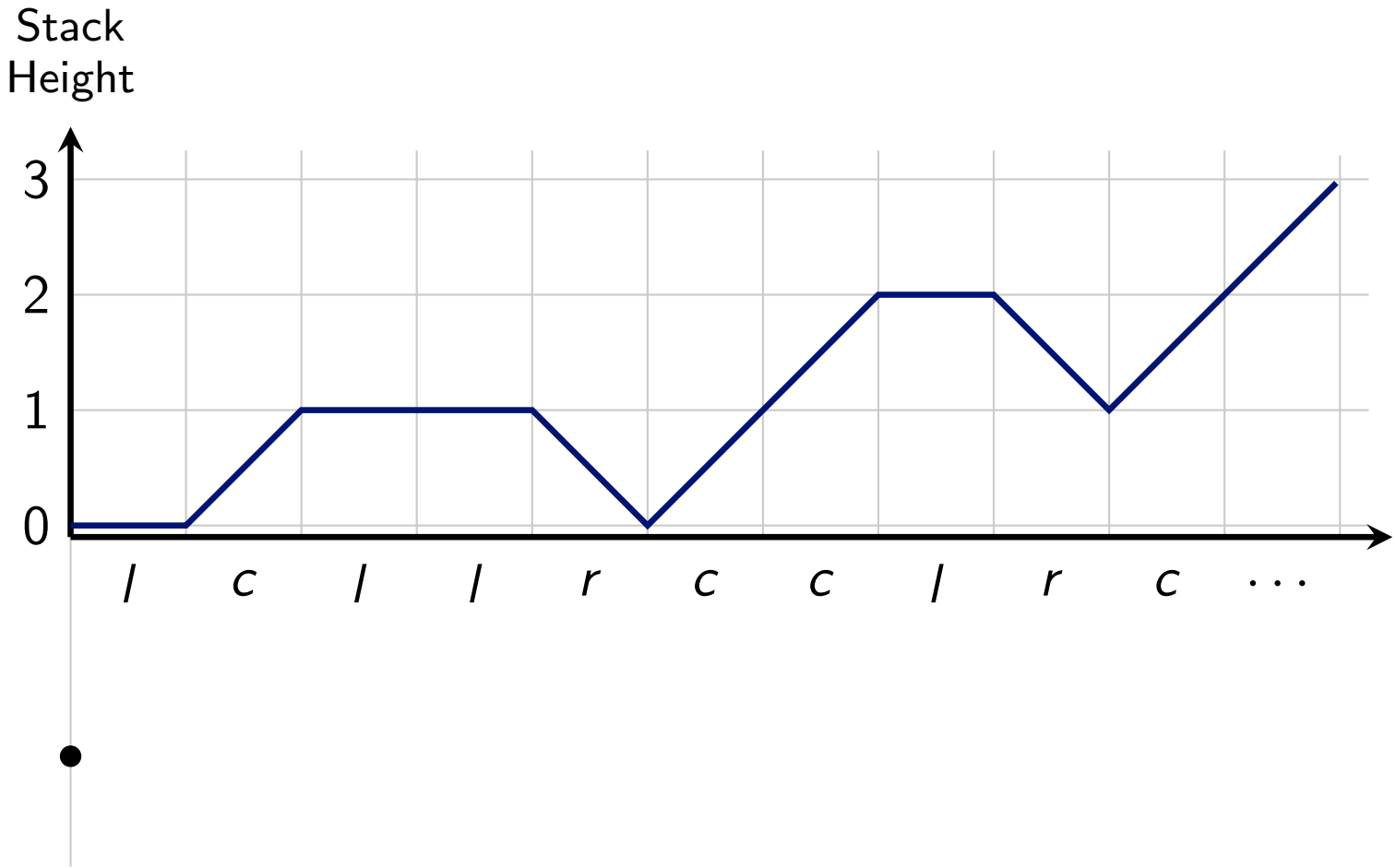
Extension of alternating automata: $\delta(q, a) = q_1 \wedge (q_2 \vee q_3)$.



[4](Bozzelli, 2007)

Intermediate Automata: 1 – AJA[4]

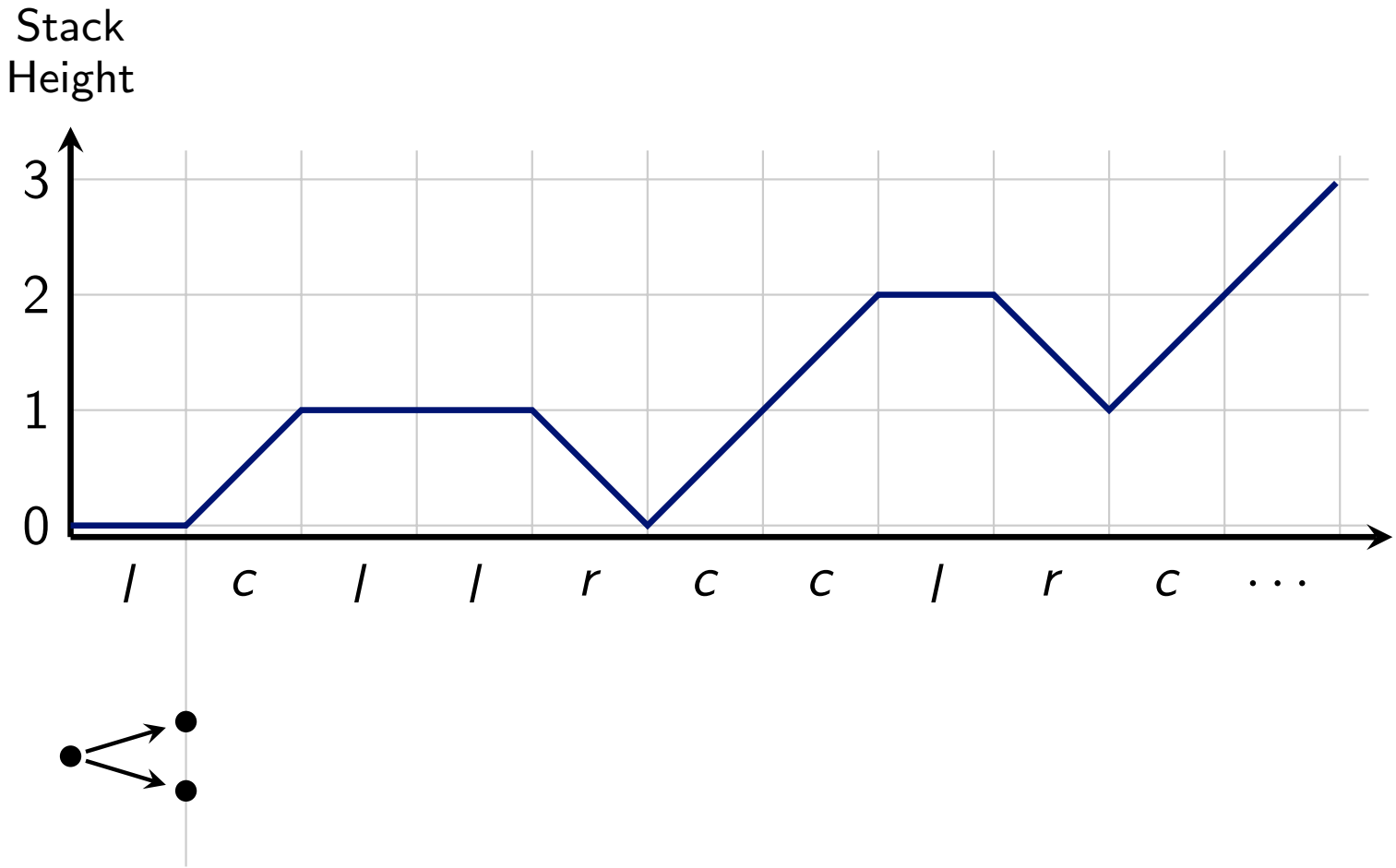
Extension of alternating automata: $\delta(q, a) = q_1 \wedge (q_2 \vee q_3)$.



[4](Bozzelli, 2007)

Intermediate Automata: 1 – AJA[4]

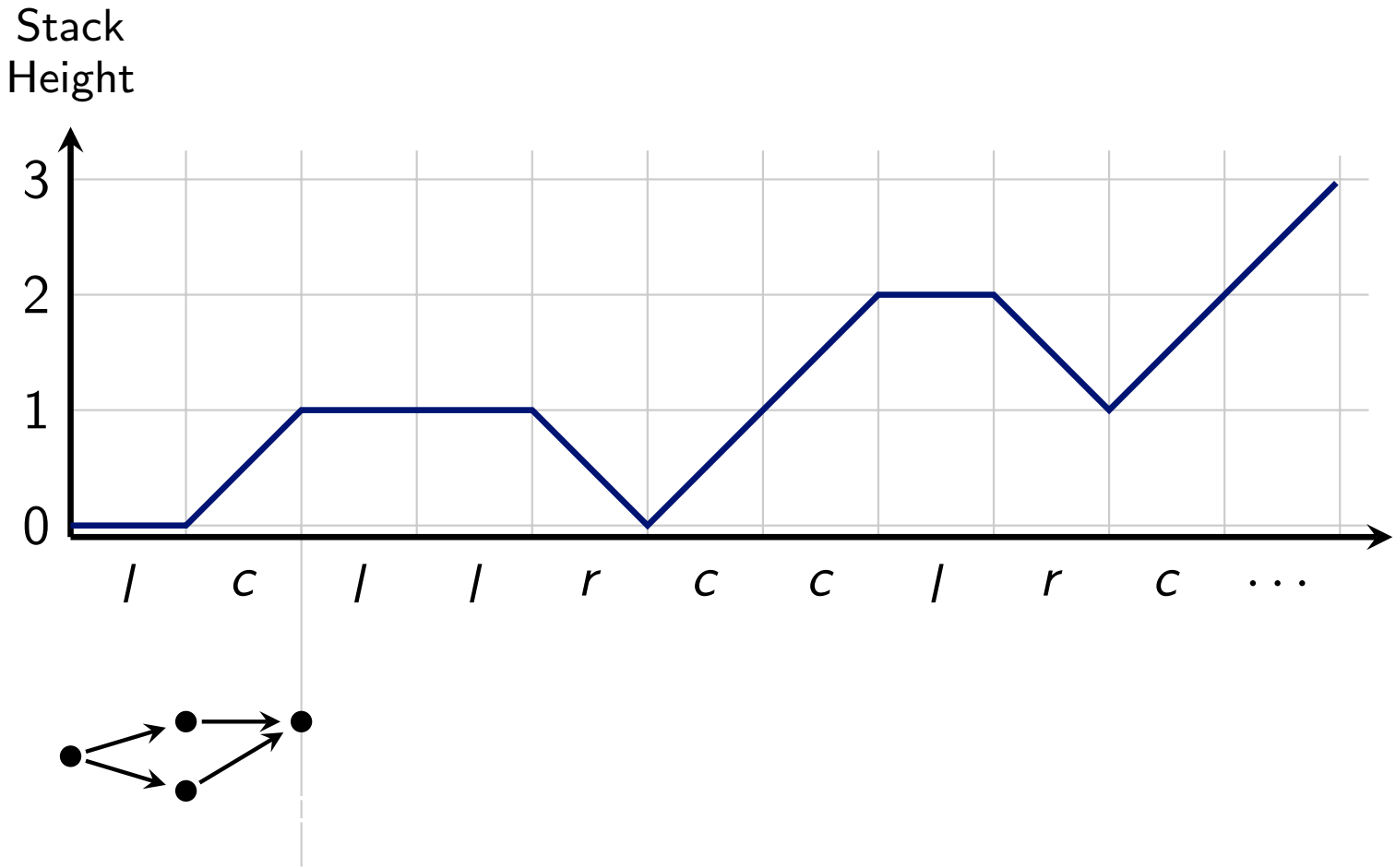
Extension of alternating automata: $\delta(q, a) = q_1 \wedge (q_2 \vee q_3)$.



[4](Bozzelli, 2007)

Intermediate Automata: 1 – AJA[4]

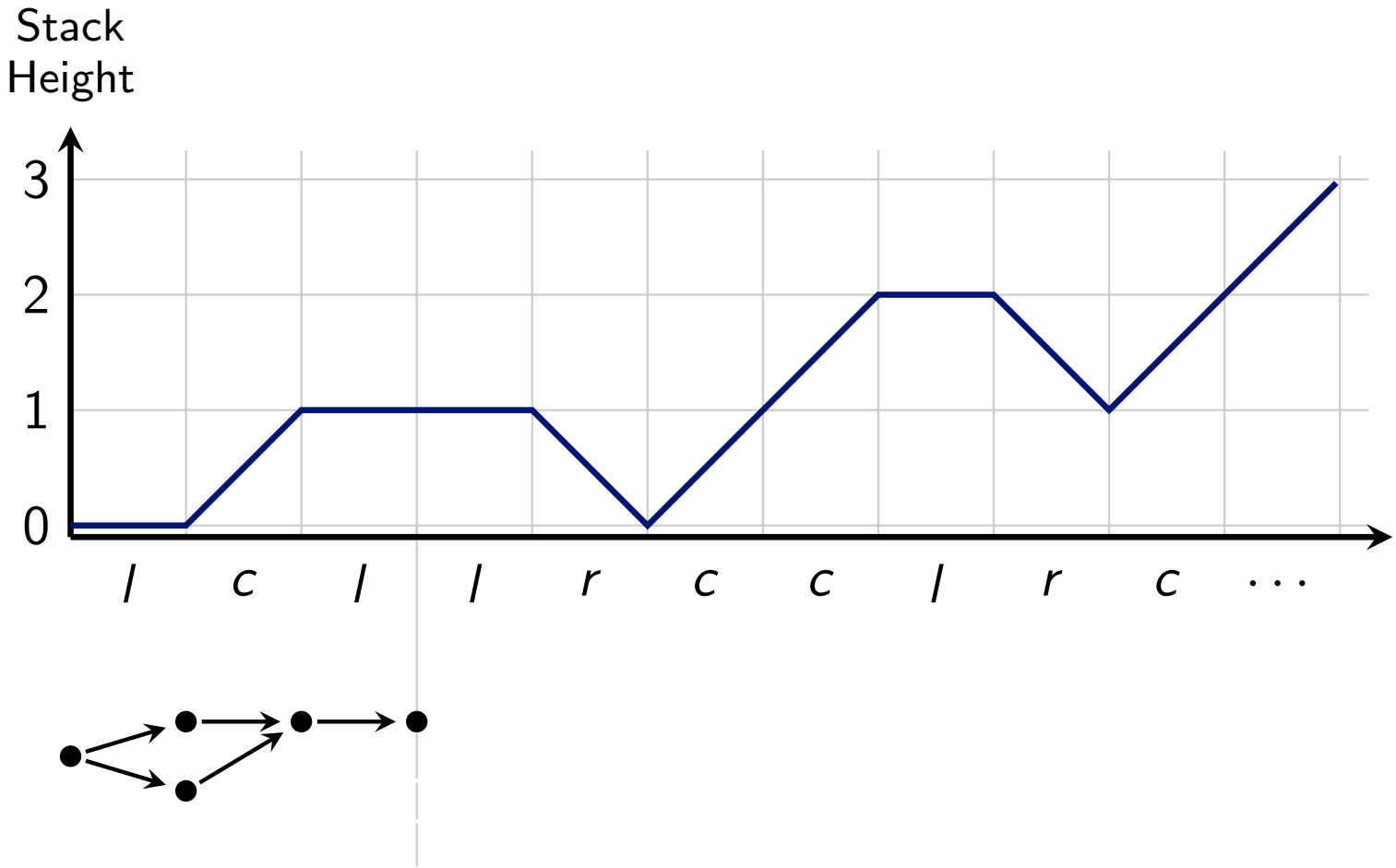
Extension of alternating automata: $\delta(q, a) = q_1 \wedge (q_2 \vee q_3)$.



[4](Bozzelli, 2007)

Intermediate Automata: 1 – AJA[4]

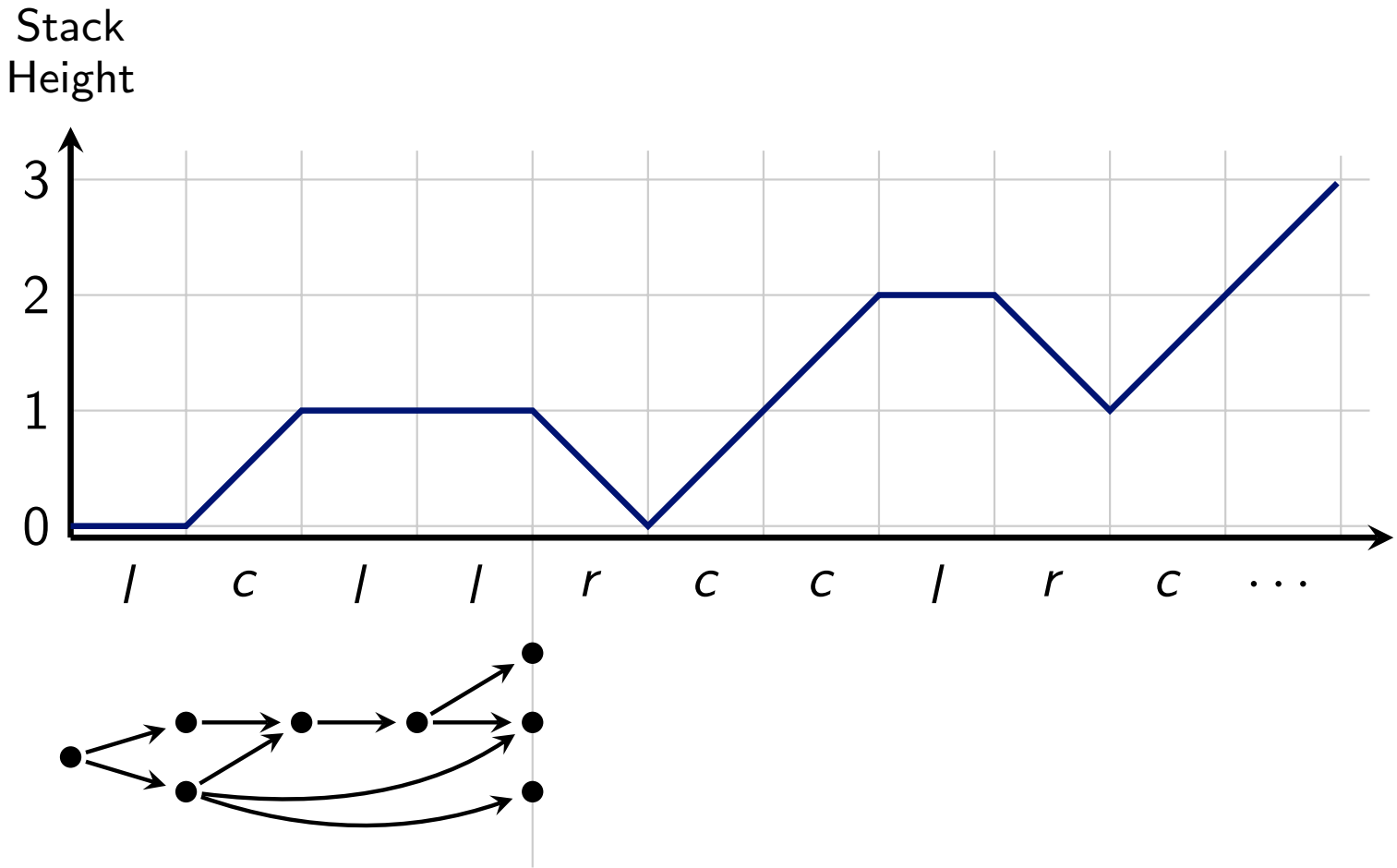
Extension of alternating automata: $\delta(q, a) = q_1 \wedge (q_2 \vee q_3)$.



[4](Bozzelli, 2007)

Intermediate Automata: 1 – AJA[4]

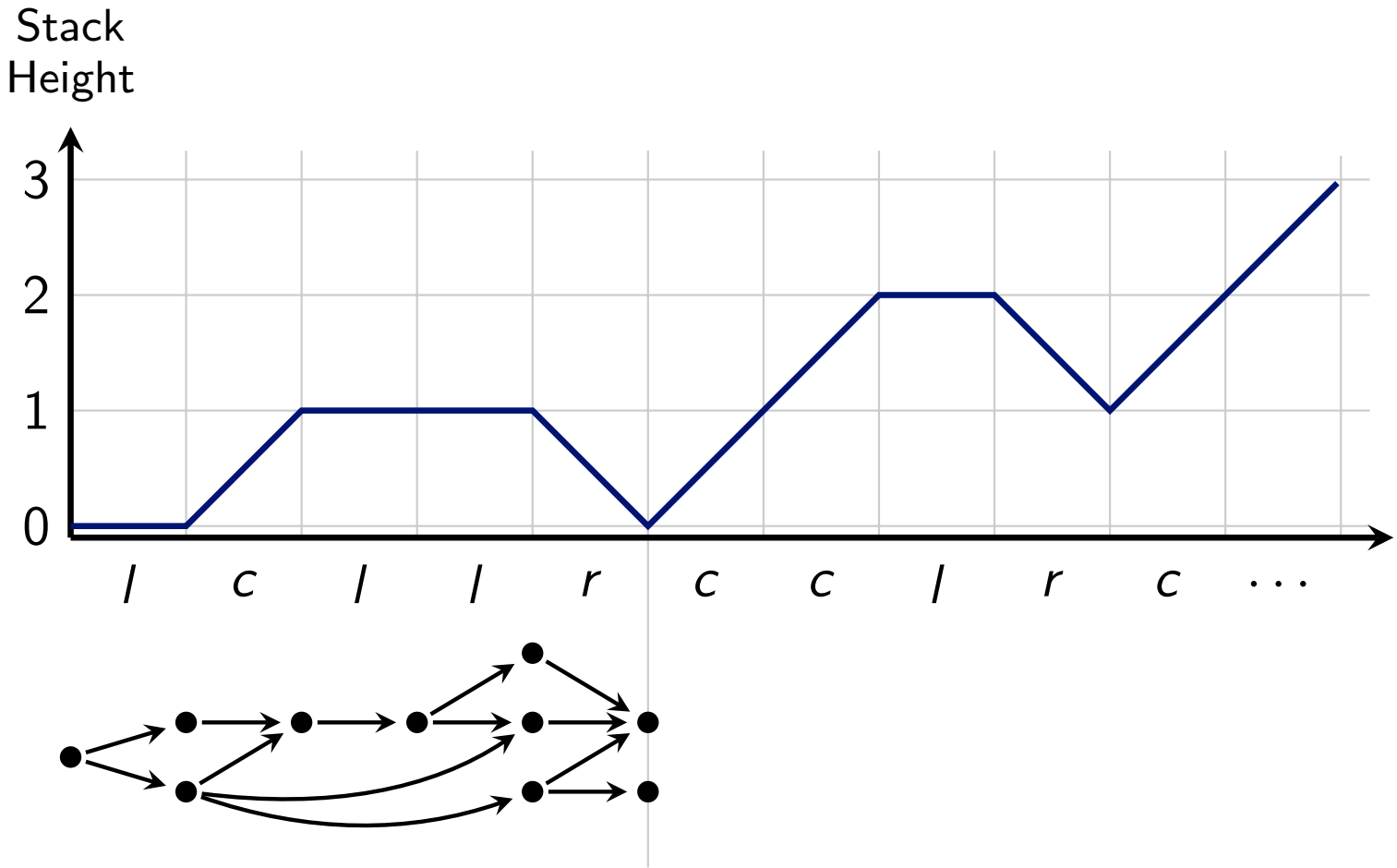
Extension of alternating automata: $\delta(q, a) = q_1 \wedge (q_2 \vee q_3)$.



[4](Bozzelli, 2007)

Intermediate Automata: 1 – AJA[4]

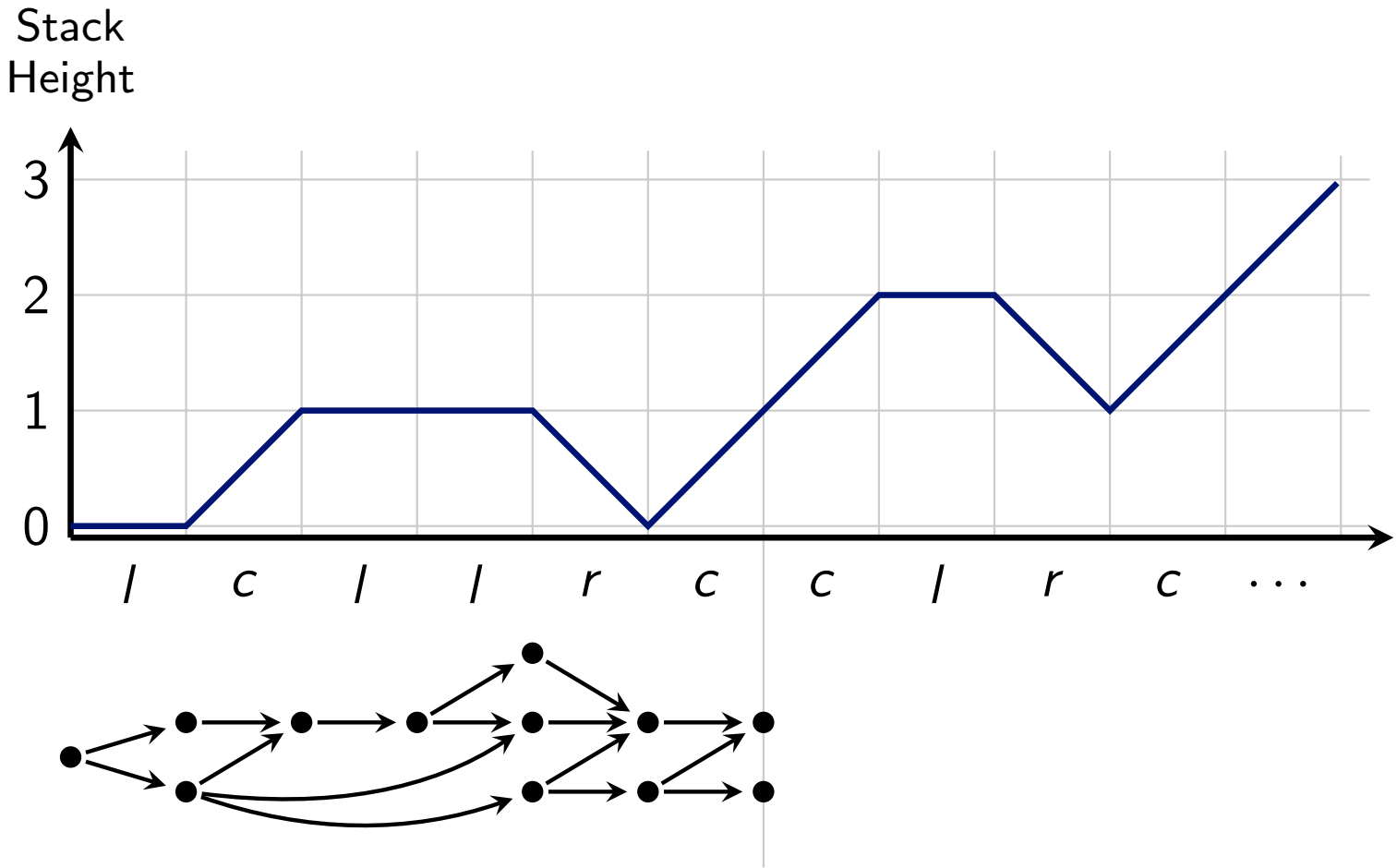
Extension of alternating automata: $\delta(q, a) = q_1 \wedge (q_2 \vee q_3)$.



[4](Bozzelli, 2007)

Intermediate Automata: 1 – AJA[4]

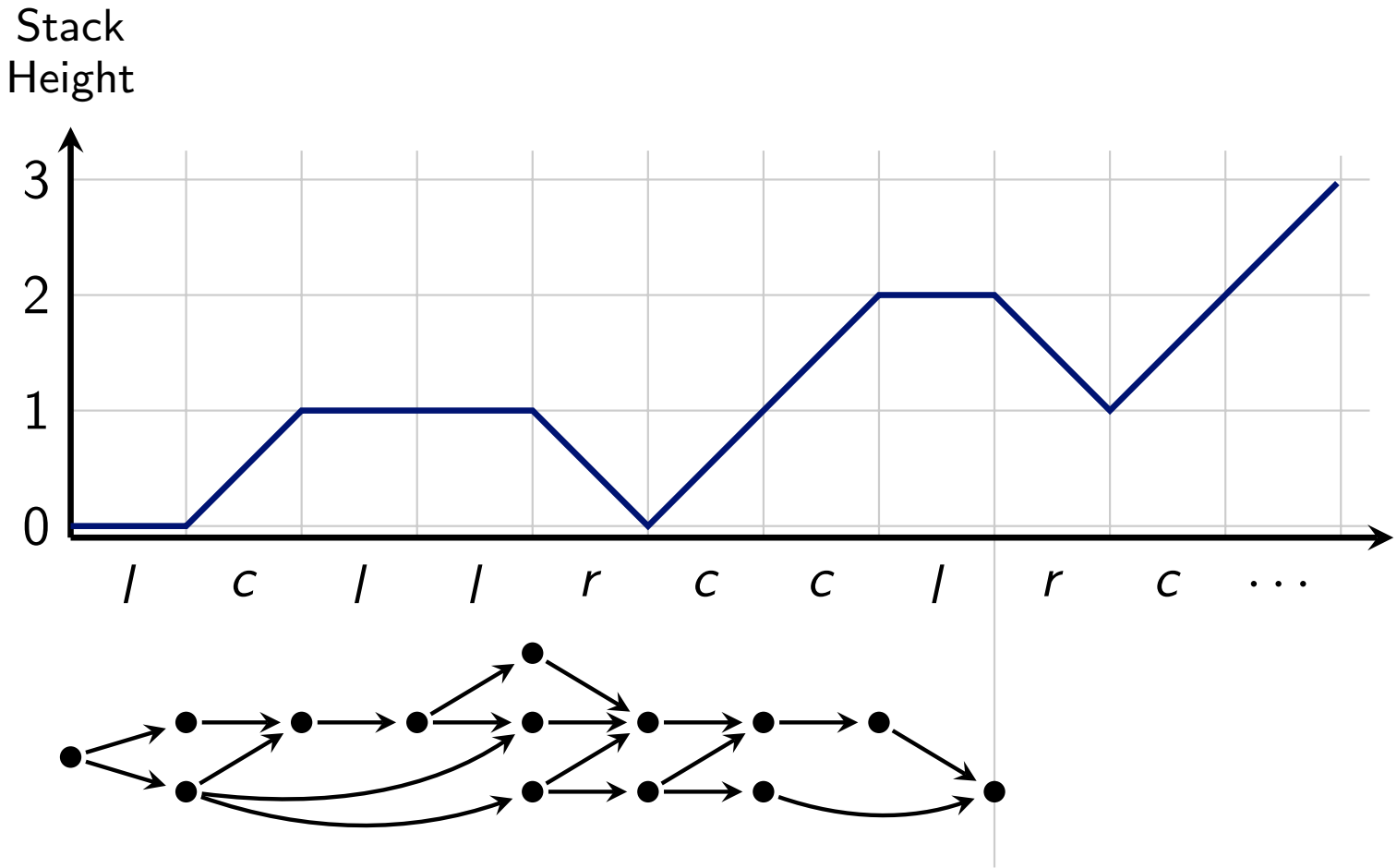
Extension of alternating automata: $\delta(q, a) = q_1 \wedge (q_2 \vee q_3)$.



[4](Bozzelli, 2007)

Intermediate Automata: 1 – AJA[4]

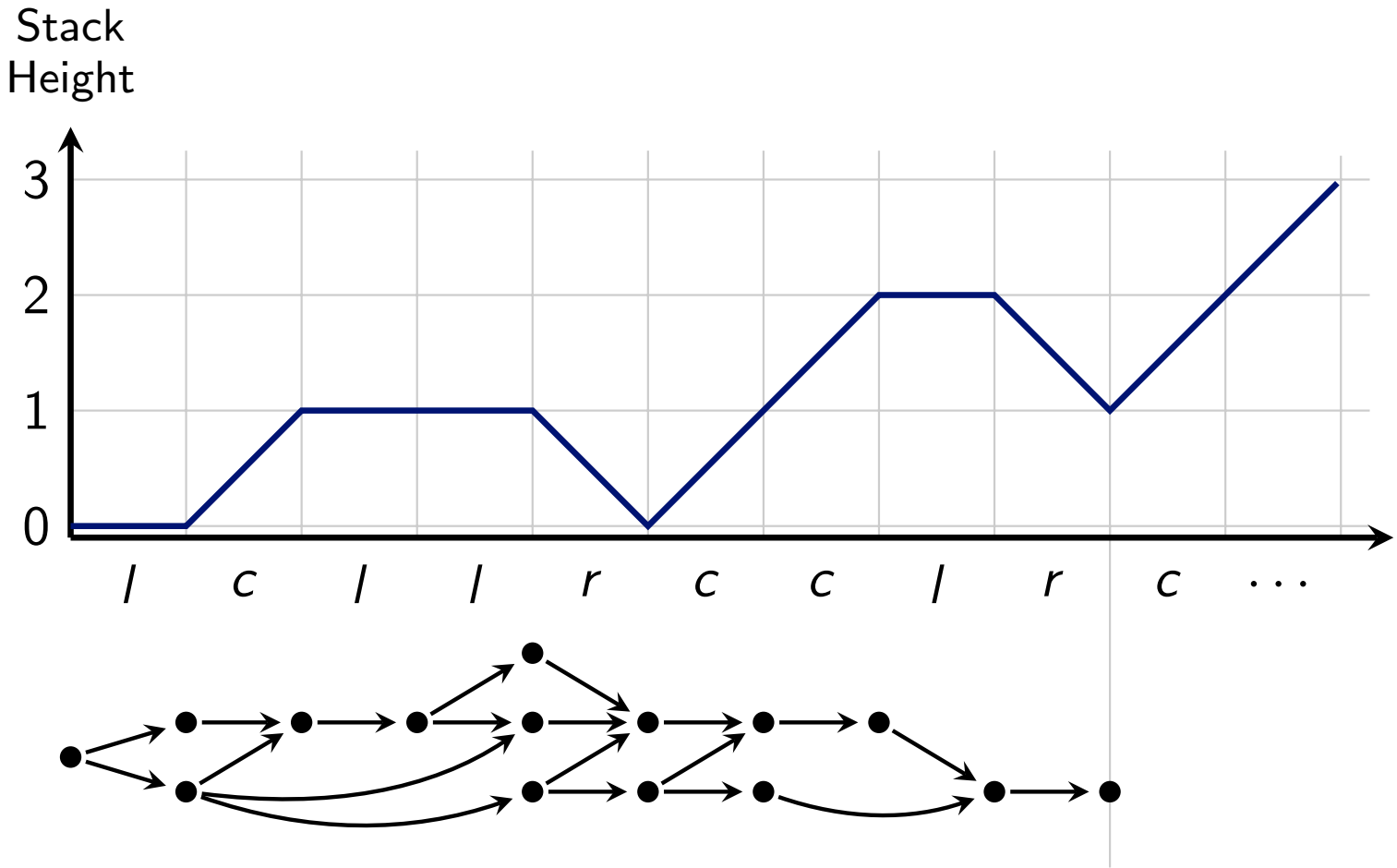
Extension of alternating automata: $\delta(q, a) = q_1 \wedge (q_2 \vee q_3)$.



[4](Bozzelli, 2007)

Intermediate Automata: 1 – AJA[4]

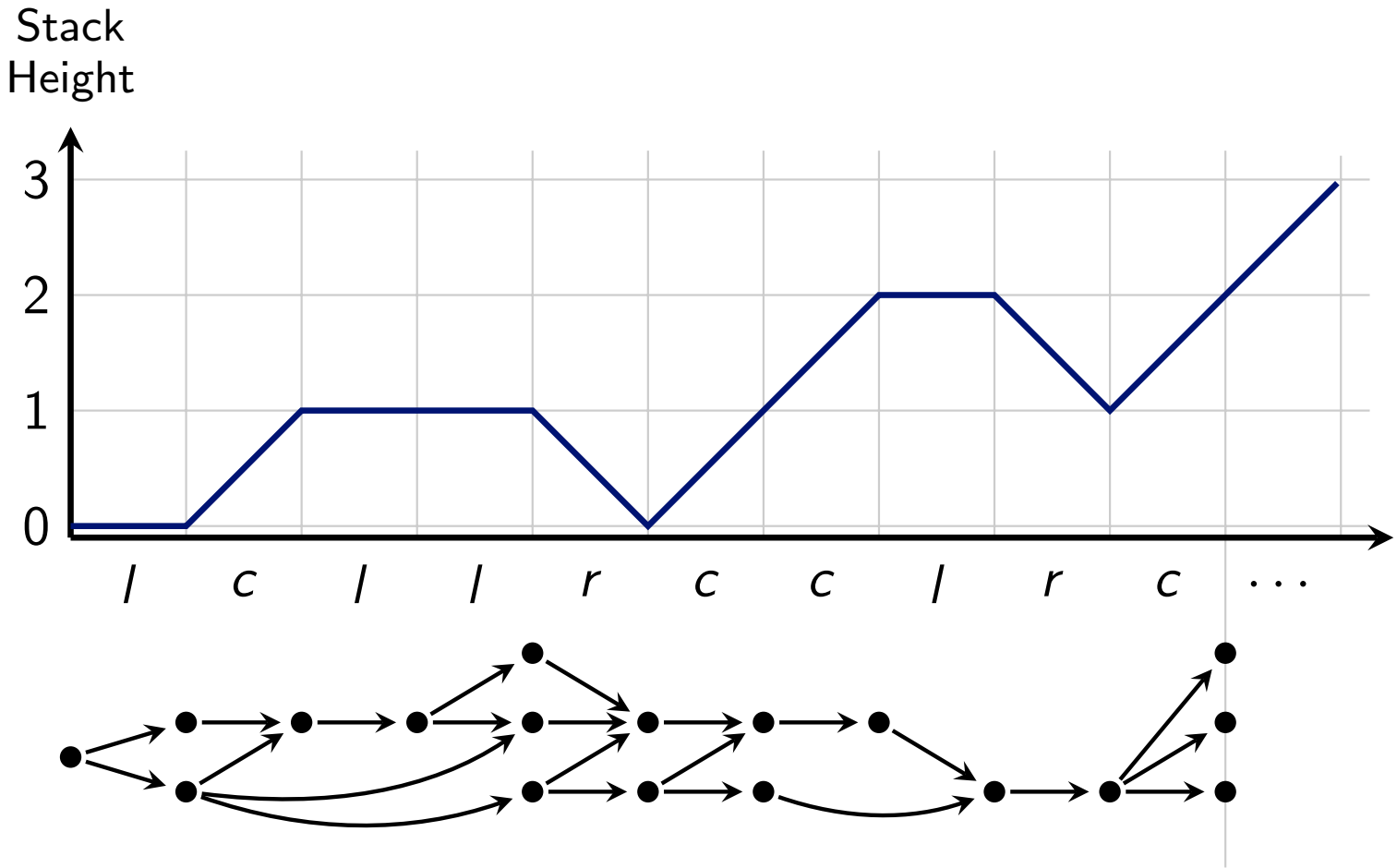
Extension of alternating automata: $\delta(q, a) = q_1 \wedge (q_2 \vee q_3)$.



[4](Bozzelli, 2007)

Intermediate Automata: 1 – AJA[4]

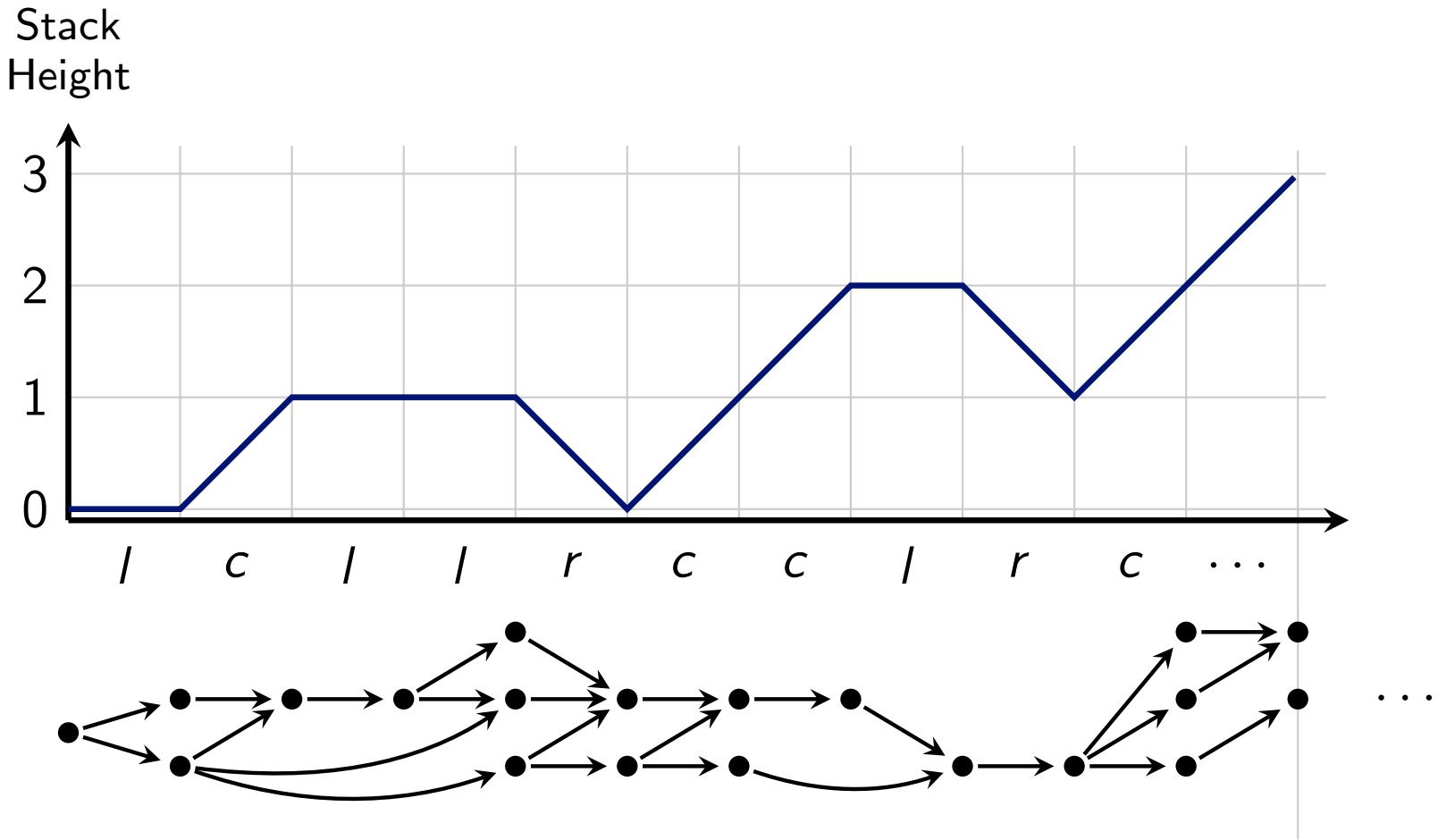
Extension of alternating automata: $\delta(q, a) = q_1 \wedge (q_2 \vee q_3)$.



[4](Bozzelli, 2007)

Intermediate Automata: 1 – AJA[4]

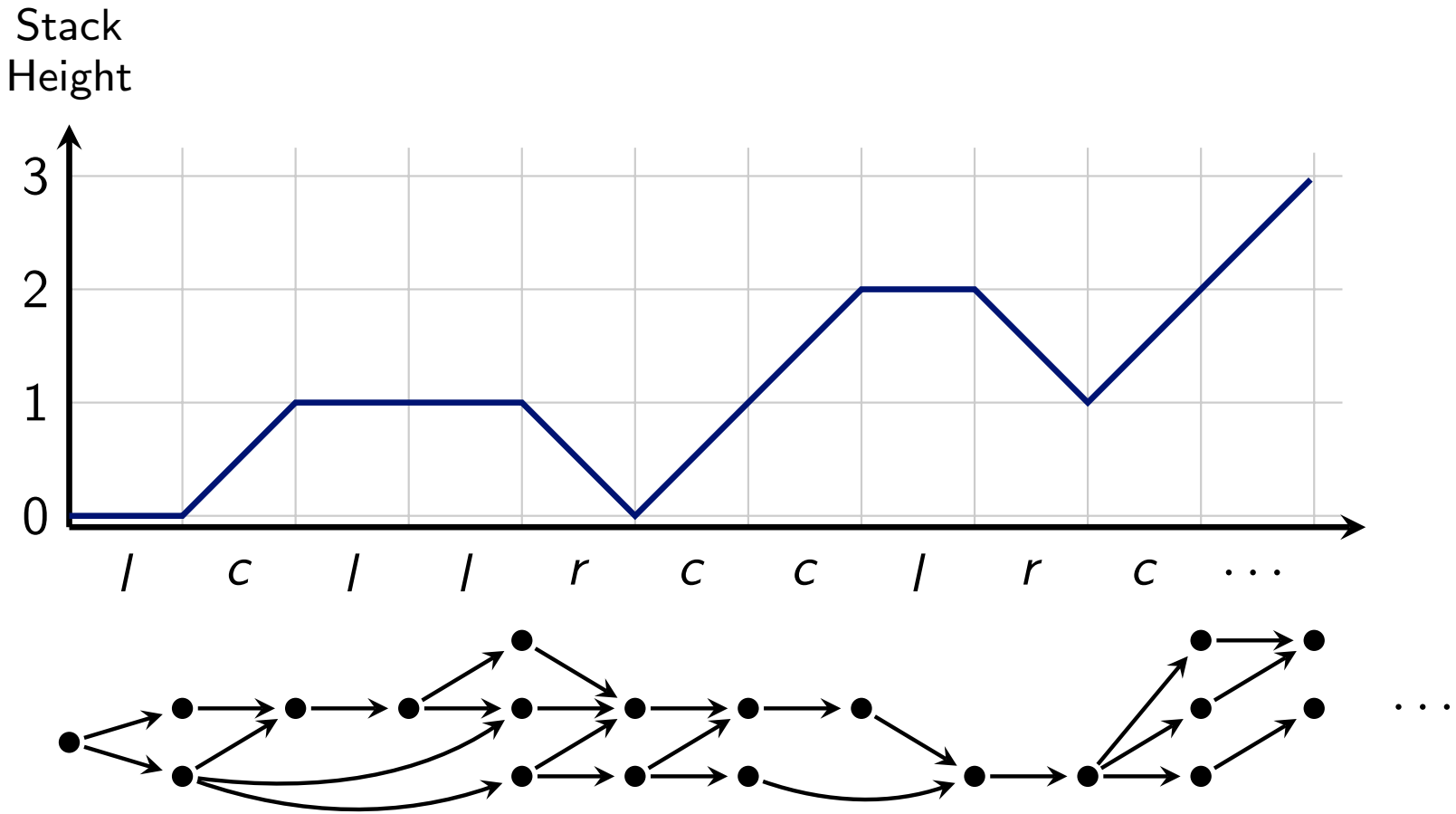
Extension of alternating automata: $\delta(q, a) = q_1 \wedge (q_2 \vee q_3)$.



[4](Bozzelli, 2007)

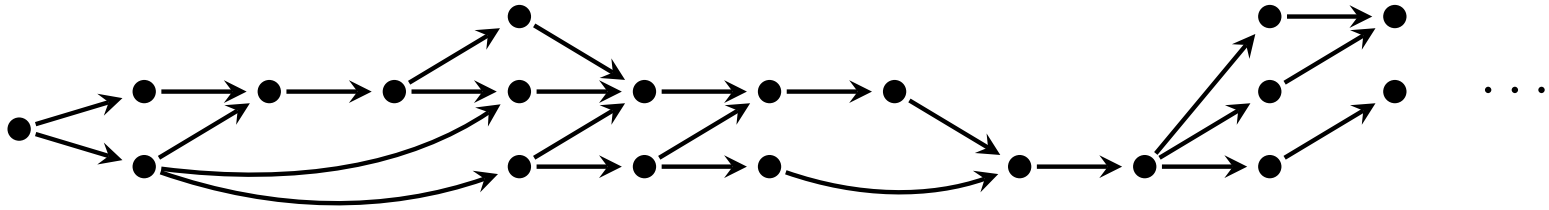
Intermediate Automata: 1 – AJA[4]

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

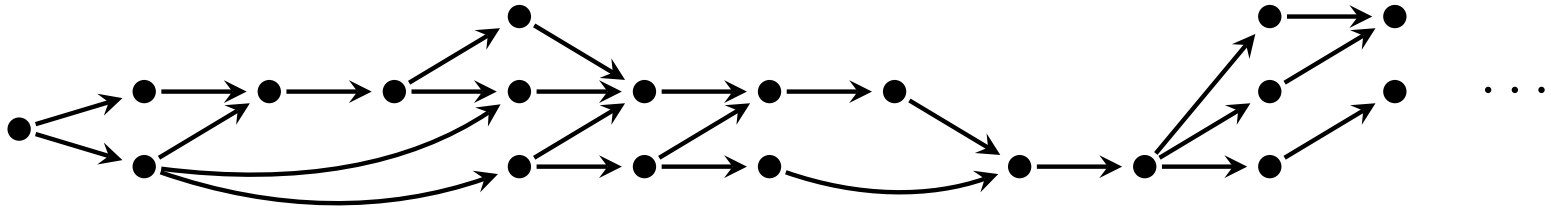


[4](Bozzelli, 2007)

Intermediate Automata: 1 – AJA

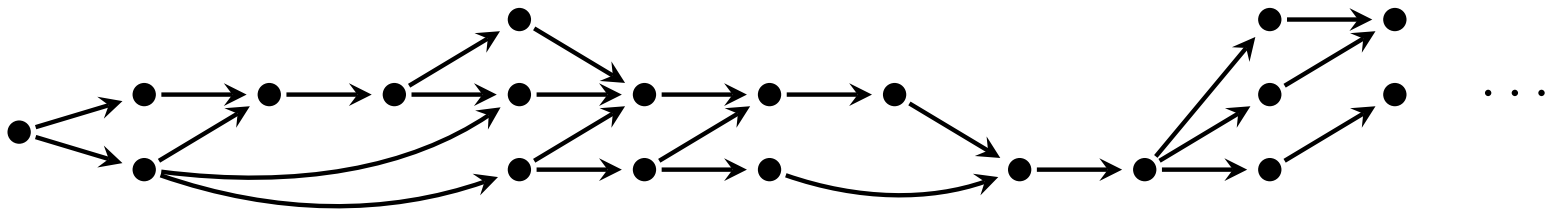


Intermediate Automata: 1 – AJA



Acceptance: Each branch visits accepting states infinitely often.

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?

Guiding Questions

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

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

$$/ \quad c \quad / \quad / \quad r \quad c \quad c \quad / \quad r \quad c \quad c \quad \cdots$$

Adapted from (Alur and Madhusudan, 2004)

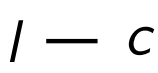
Encoding Words as Trees

$l \quad c \quad l \quad l \quad r \quad c \quad c \quad l \quad r \quad c \quad c \quad \cdots$

Adapted from (Alur and Madhusudan, 2004)

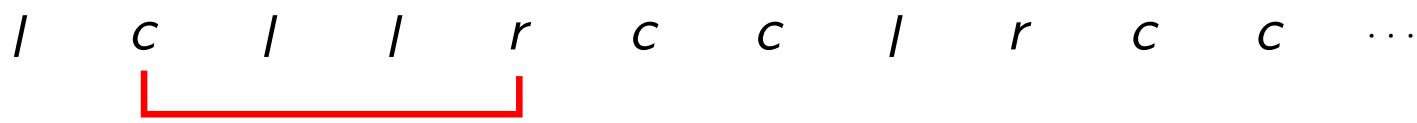
Encoding Words as Trees

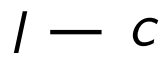
$l \quad c \quad l \quad l \quad r \quad c \quad c \quad l \quad r \quad c \quad c \quad \cdots$



Adapted from (Alur and Madhusudan, 2004)

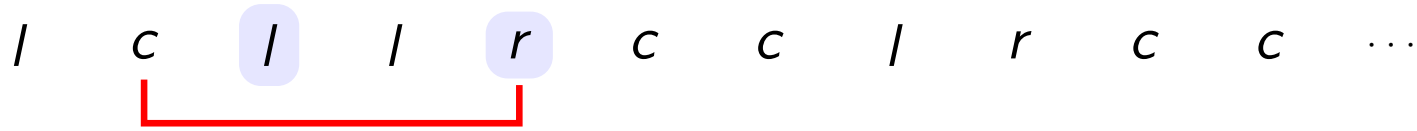
Encoding Words as Trees

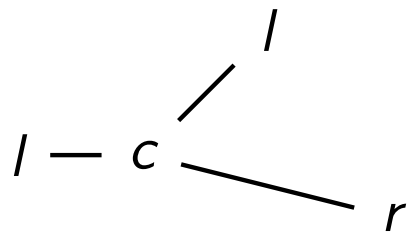




Adapted from (Alur and Madhusudan, 2004)

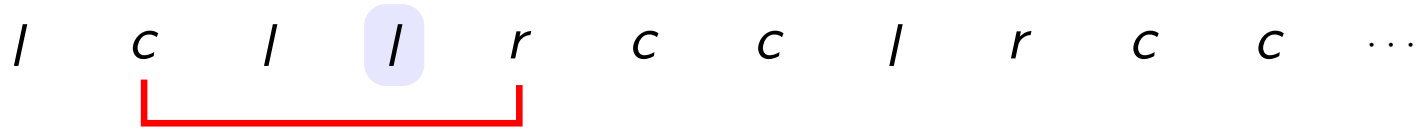
Encoding Words as Trees

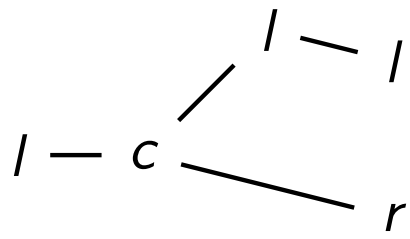




Adapted from (Alur and Madhusudan, 2004)

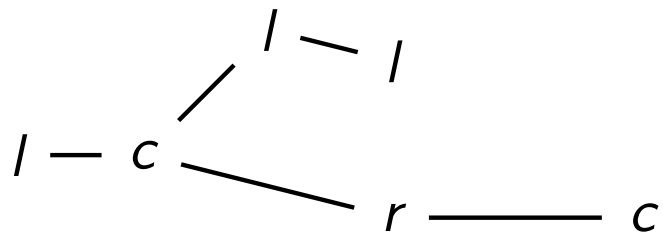
Encoding Words as Trees





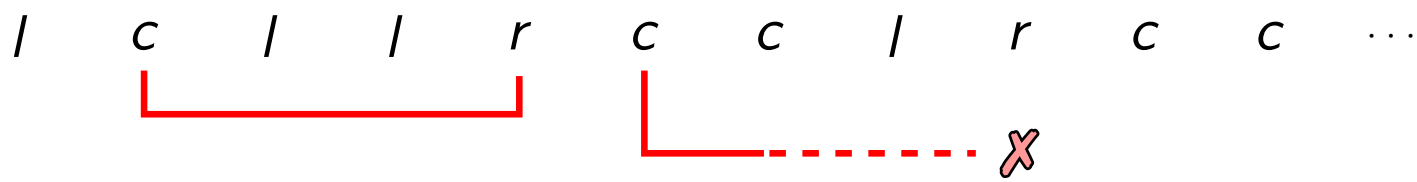
Adapted from (Alur and Madhusudan, 2004)

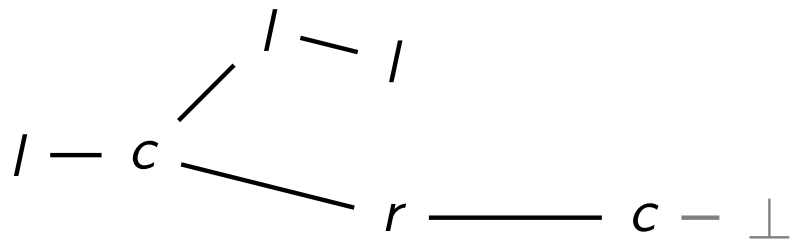
Encoding Words as Trees



Adapted from (Alur and Madhusudan, 2004)

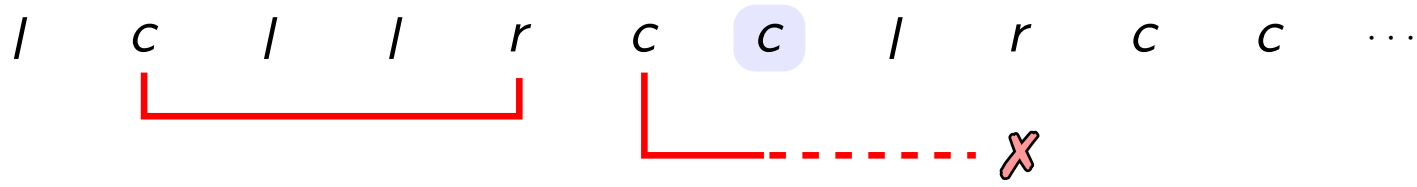
Encoding Words as Trees

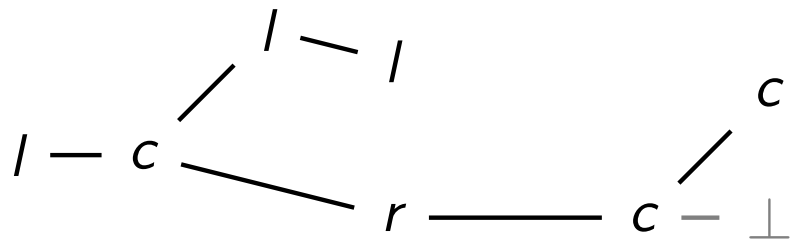




Adapted from (Alur and Madhusudan, 2004)

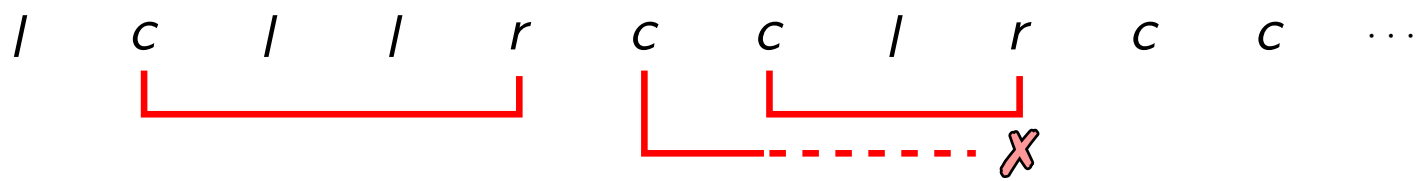
Encoding Words as Trees

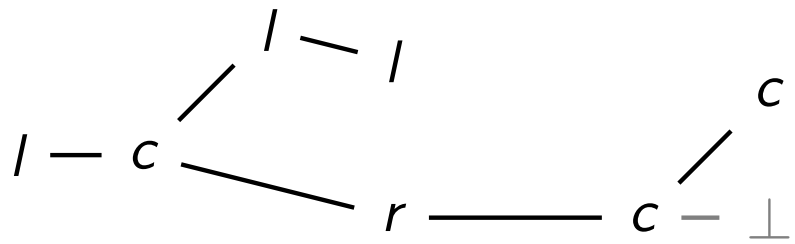




Adapted from (Alur and Madhusudan, 2004)

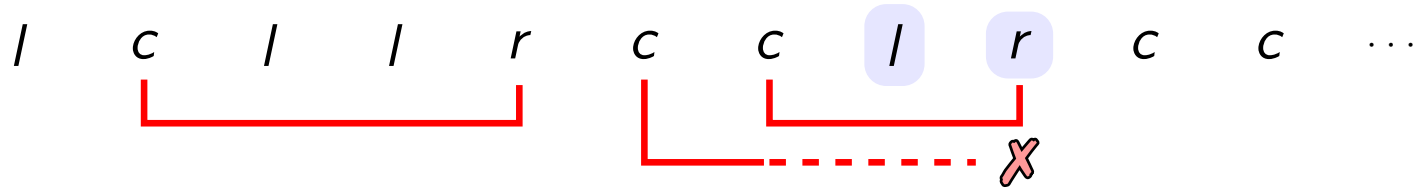
Encoding Words as Trees

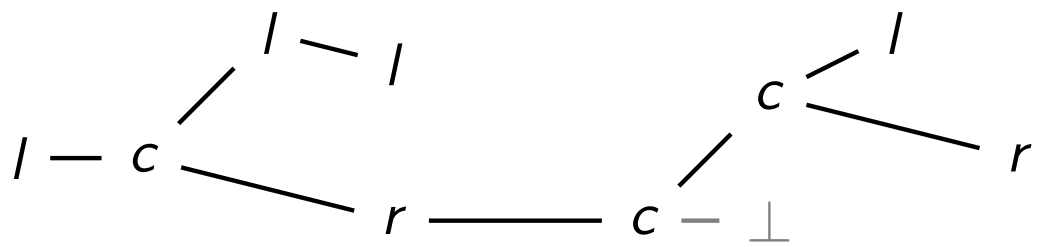




Adapted from (Alur and Madhusudan, 2004)

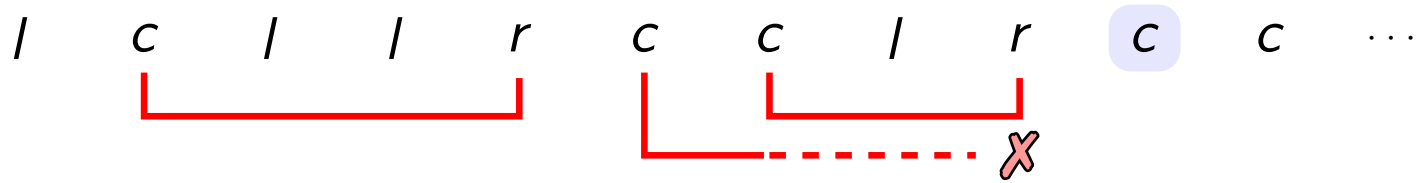
Encoding Words as Trees

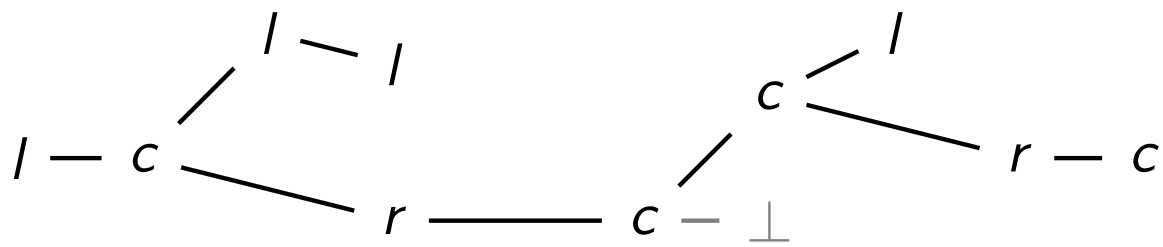




Adapted from (Alur and Madhusudan, 2004)

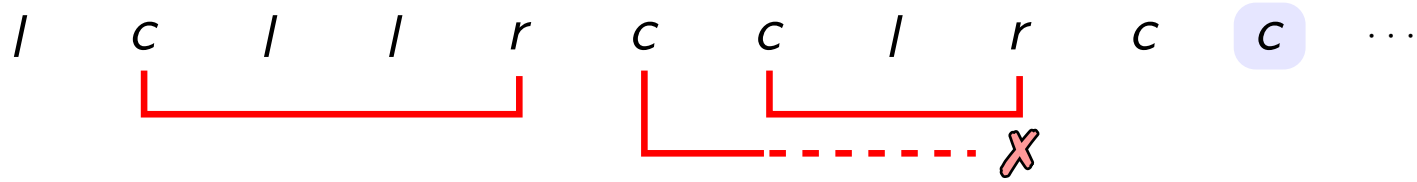
Encoding Words as Trees

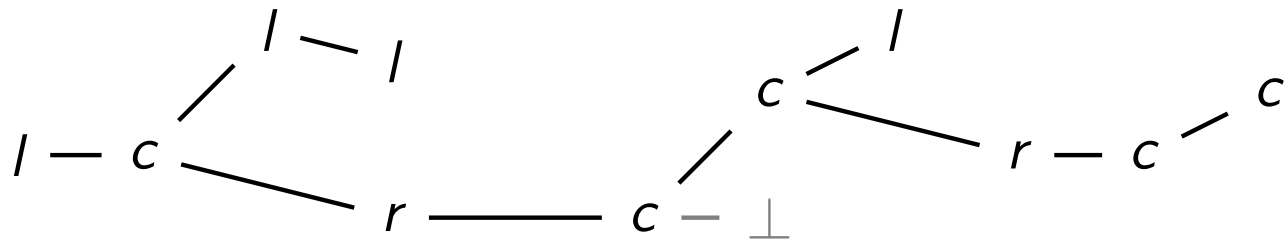




Adapted from (Alur and Madhusudan, 2004)

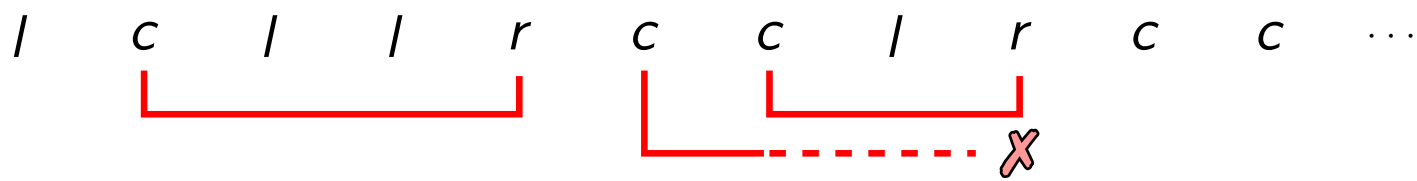
Encoding Words as Trees

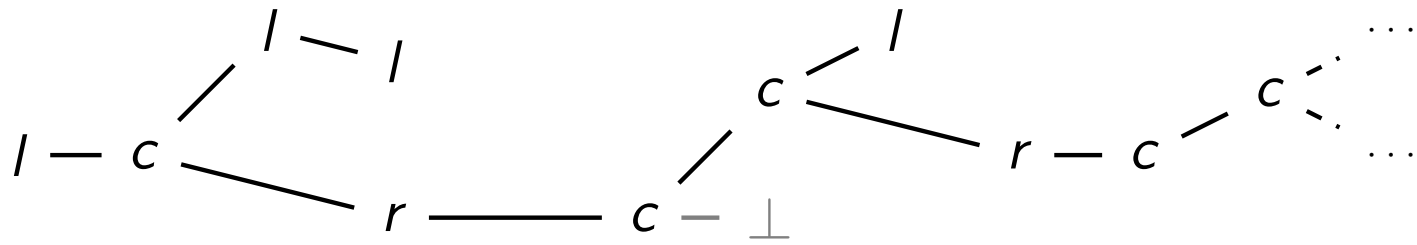




Adapted from (Alur and Madhusudan, 2004)

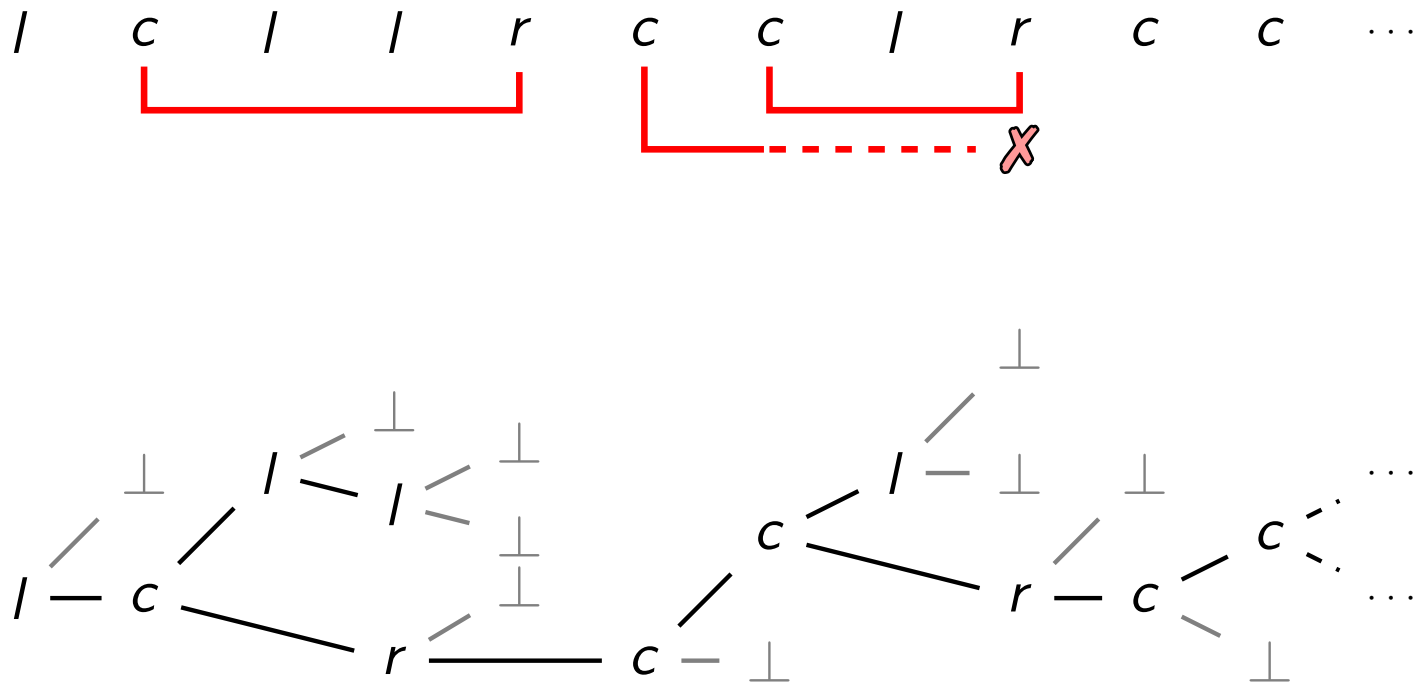
Encoding Words as Trees





Adapted from (Alur and Madhusudan, 2004)

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?

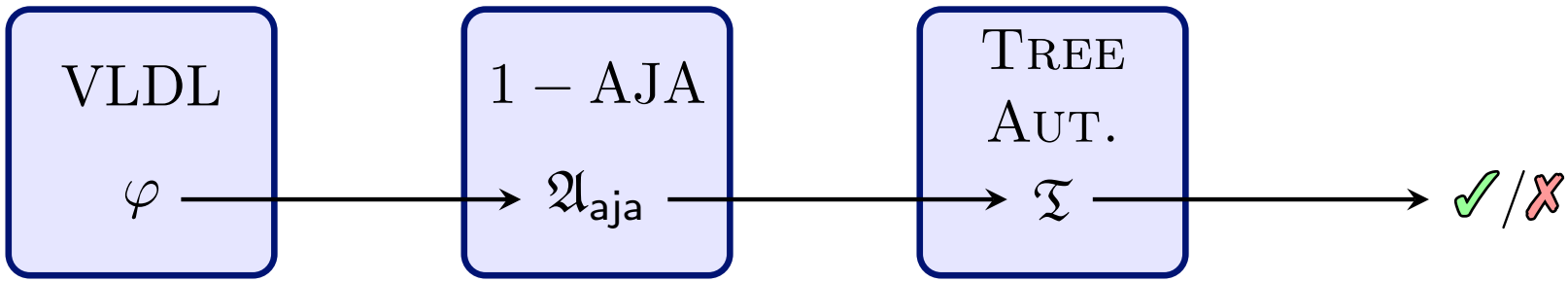
Guiding Questions

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

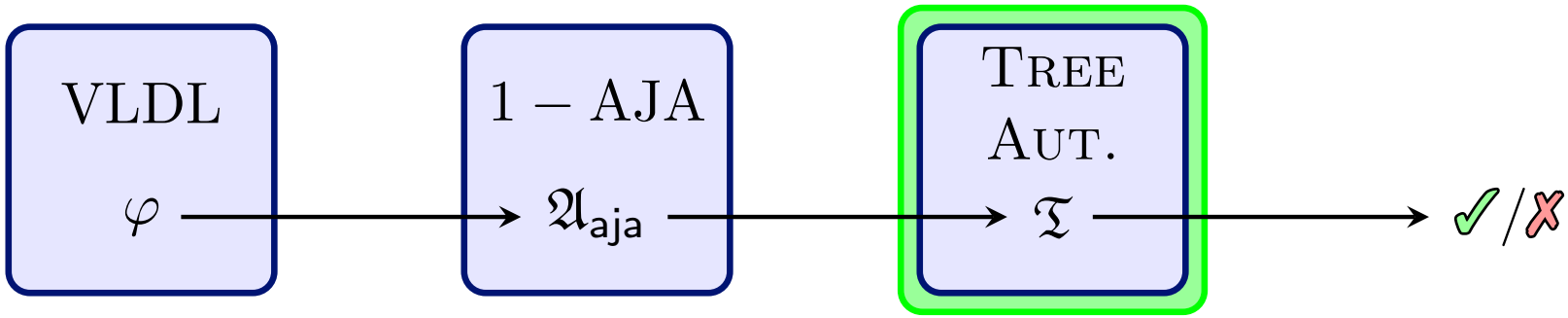
Guiding Questions

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

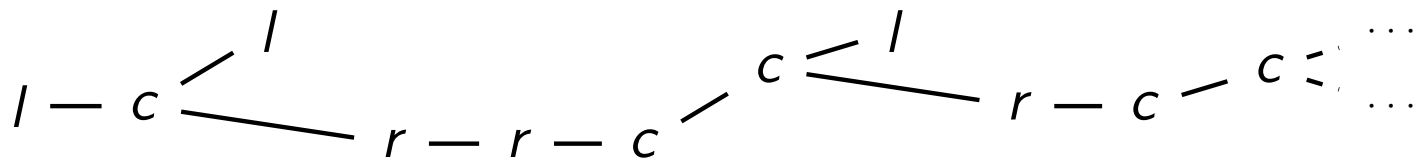
Overview


VLDL
φ
$1 - \mathrm{AJA}$
$\mathfrak{A}_{\mathsf{aja}}$
Tree Aut.
$\mathfrak{T}$
✓/✗

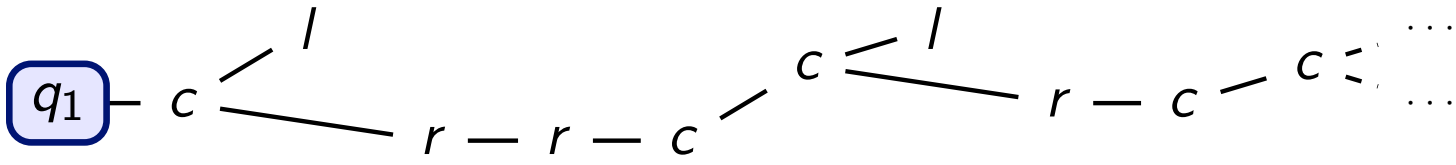
Overview


VLDL
φ
$1 - \mathrm{AJA}$
$\mathfrak{A}_{\mathsf{aja}}$
Tree Aut.
$\mathfrak{T}$
✓/✗

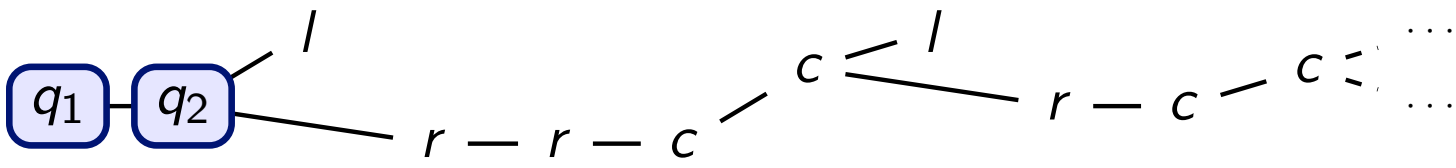
Tree Automata



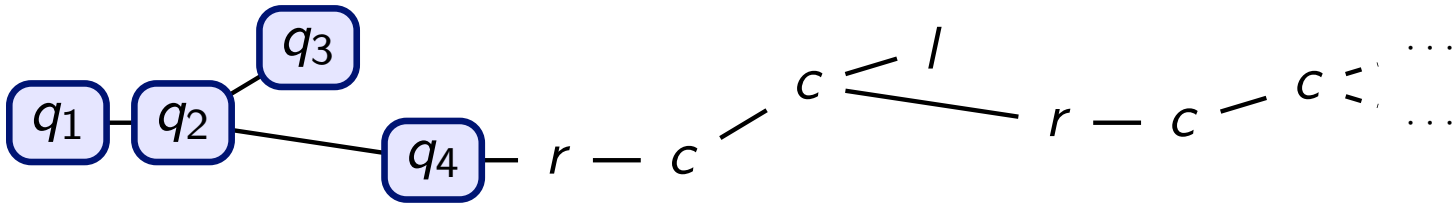
Tree Automata



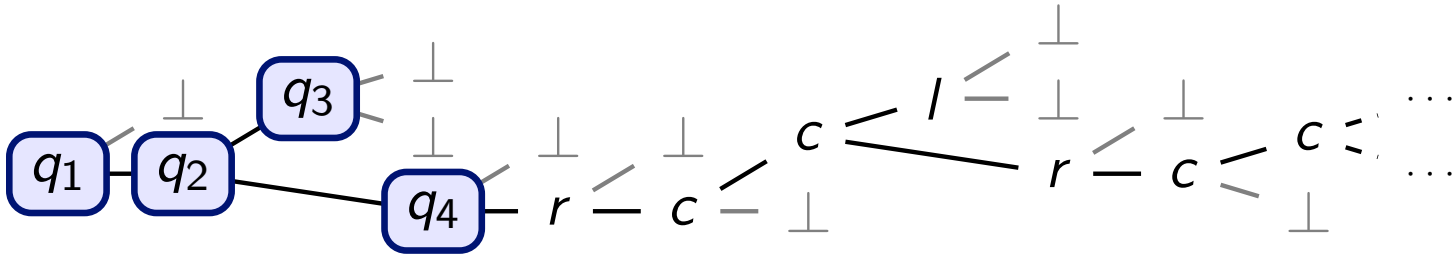
Tree Automata



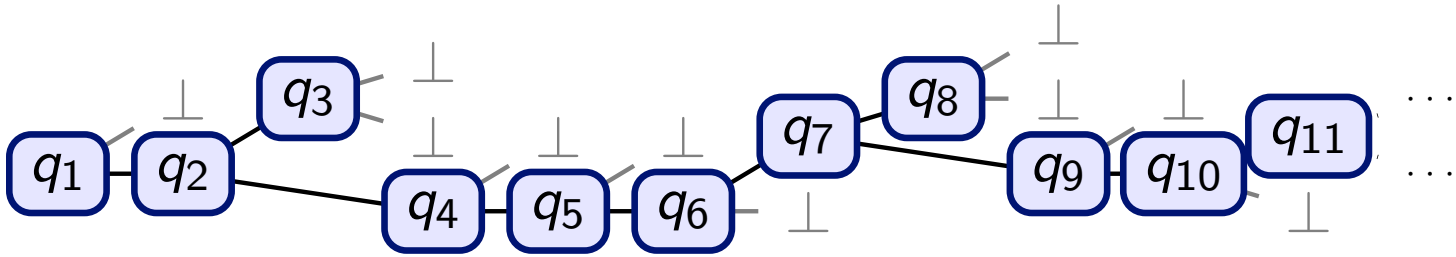
Tree Automata



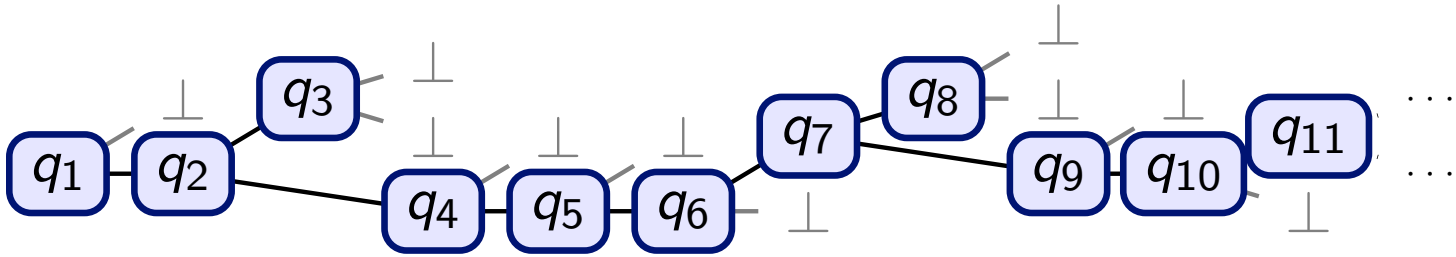
Tree Automata



Tree Automata

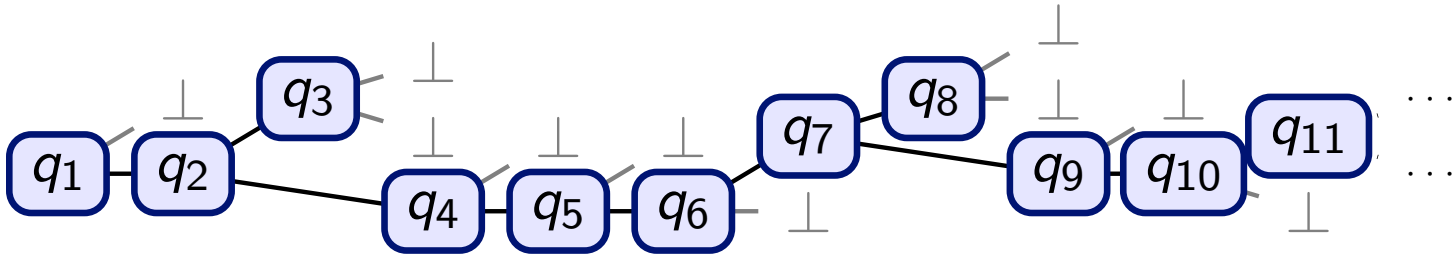


Tree Automata



Acceptance: Every branch has infinitely many accepting vertices

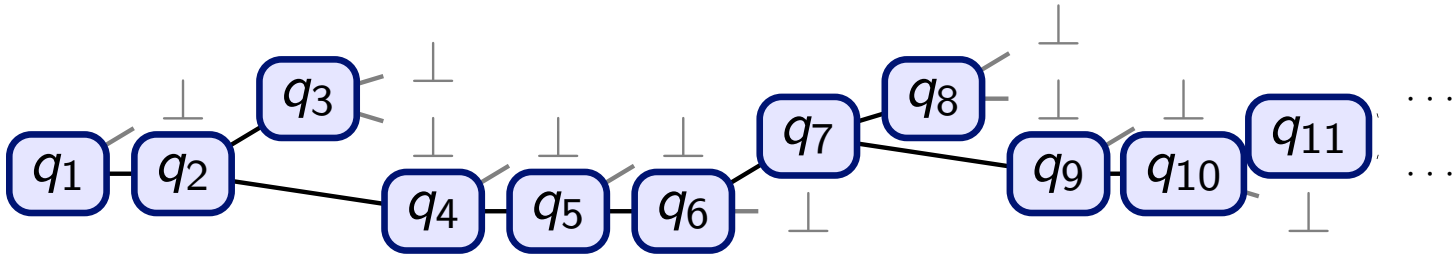
Tree Automata



Acceptance: Every branch has infinitely many accepting vertices

Component	Technique

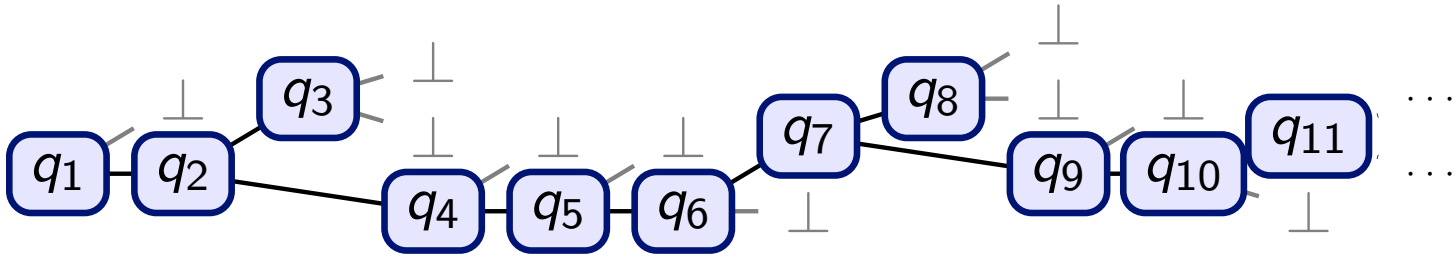
Tree Automata



Acceptance: Every branch has infinitely many accepting vertices

Component	Technique
States	

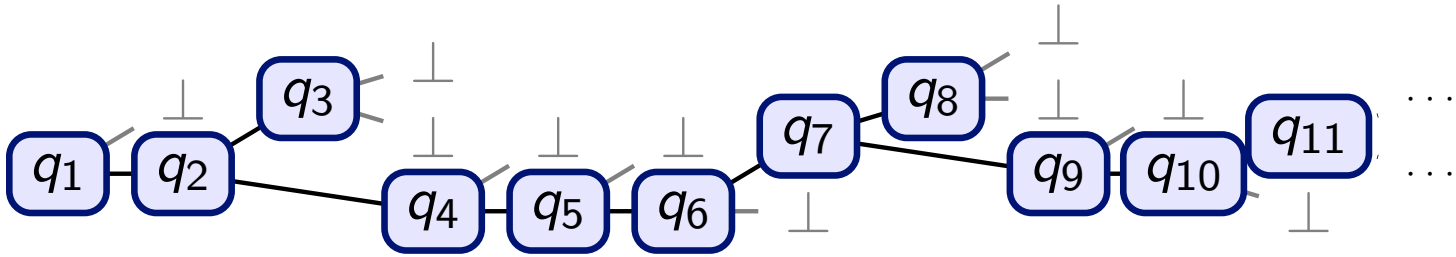
Tree Automata



Acceptance: Every branch has infinitely many accepting vertices

Component	Technique
States	
Transitions	

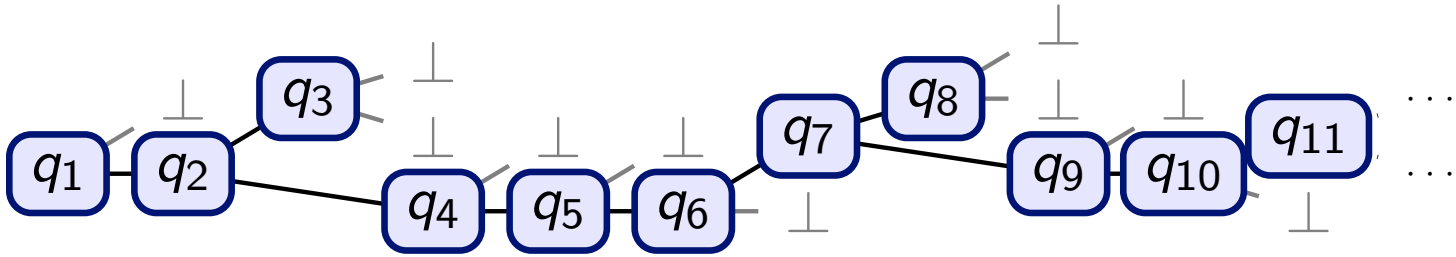
Tree Automata



Acceptance: Every branch has infinitely many accepting vertices

Component	Technique
States	
Transitions	
Accepting States	

Tree Automata



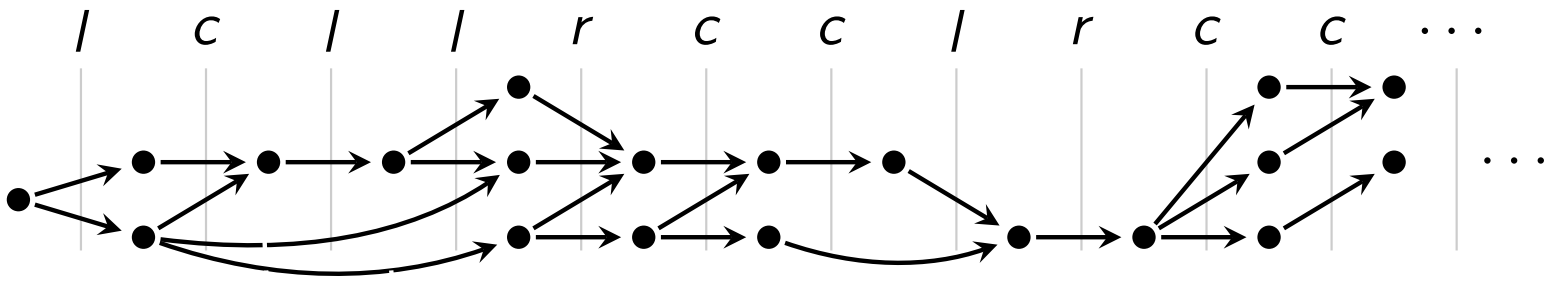
Acceptance: Every branch has infinitely many accepting vertices

Component	Technique
States	?
Transitions	?
Accepting States	?

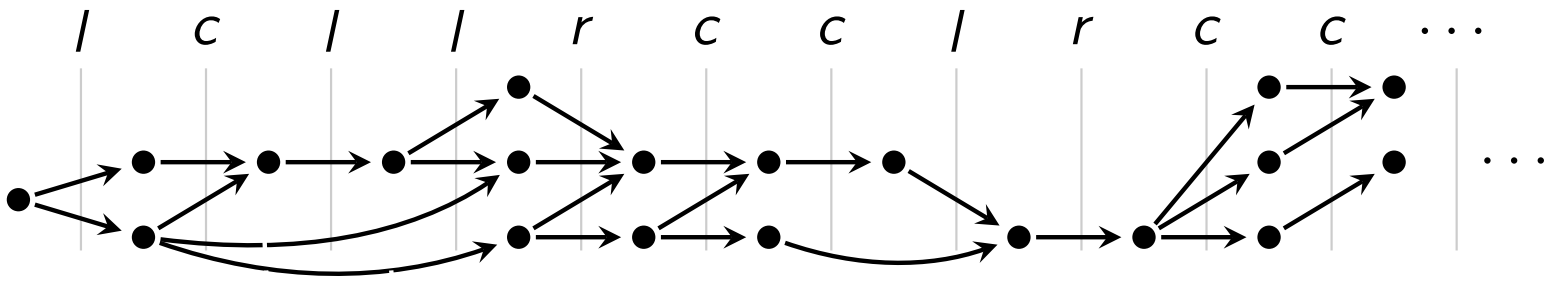
Translating 1 – AJAs into Tree Automata

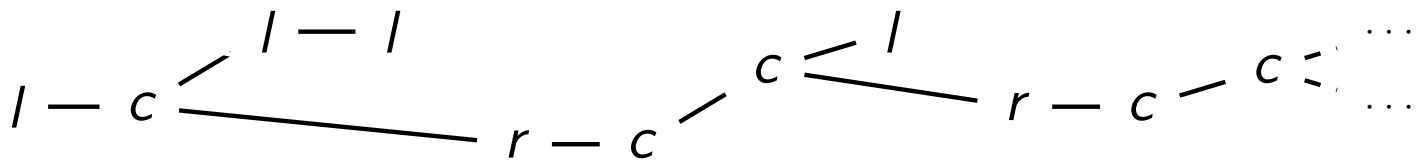
$$/ \quad c \quad / \quad / \quad r \quad c \quad c \quad / \quad r \quad c \quad c \quad \cdots$$

Translating 1 – AJAs into Tree Automata

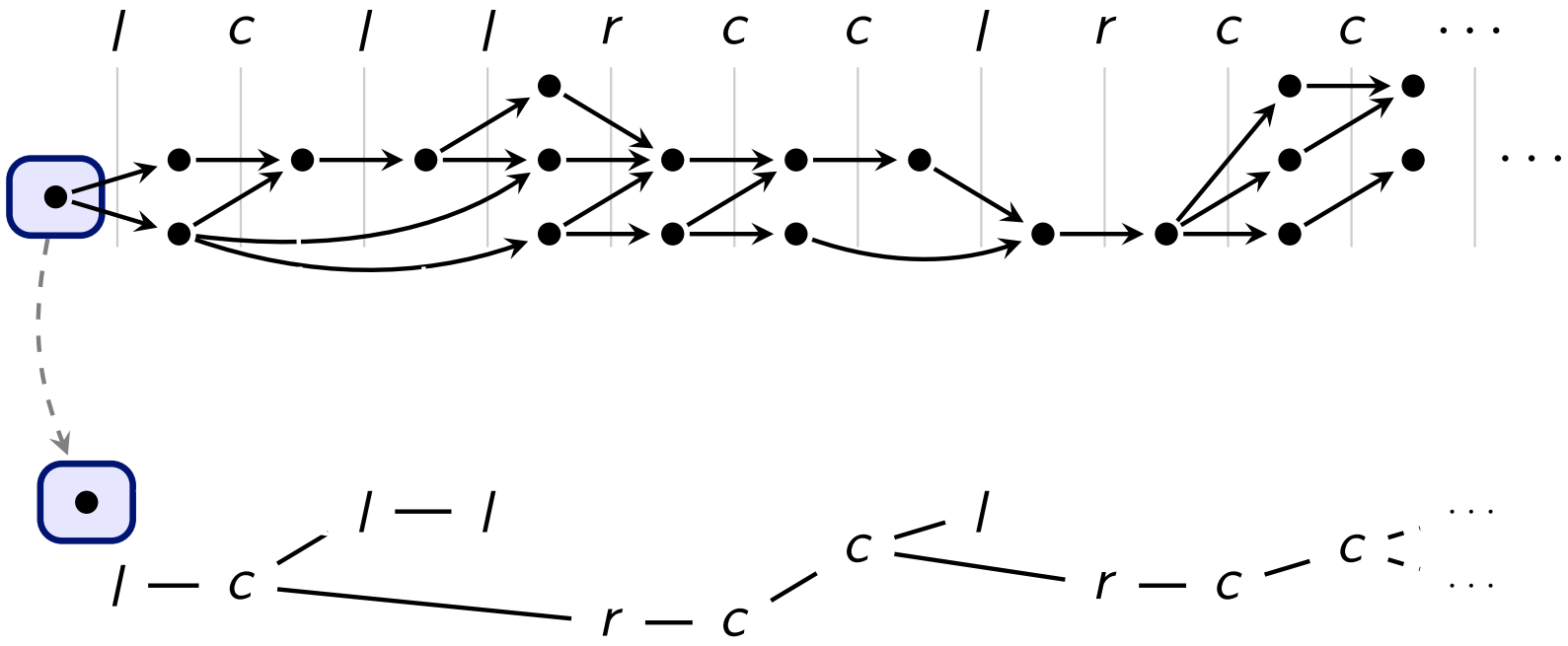


Translating 1 – AJAs into Tree Automata

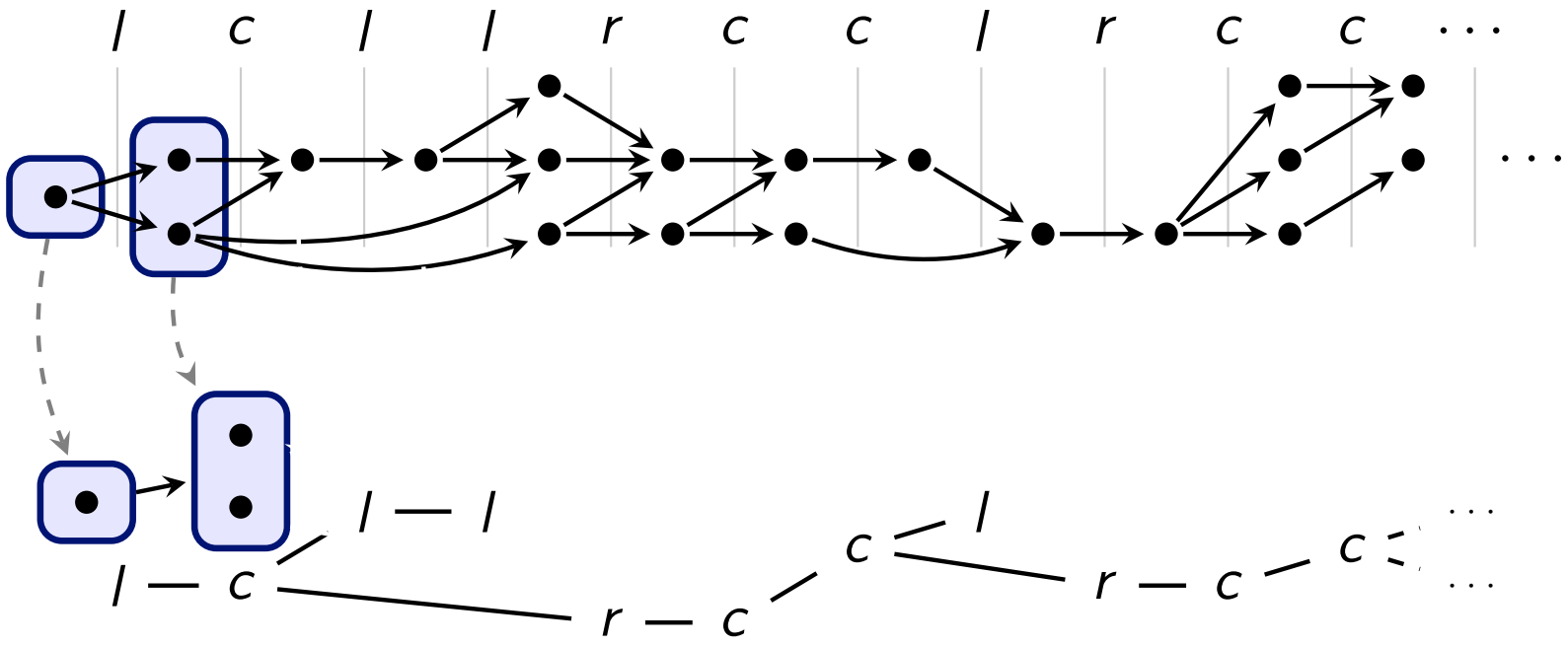




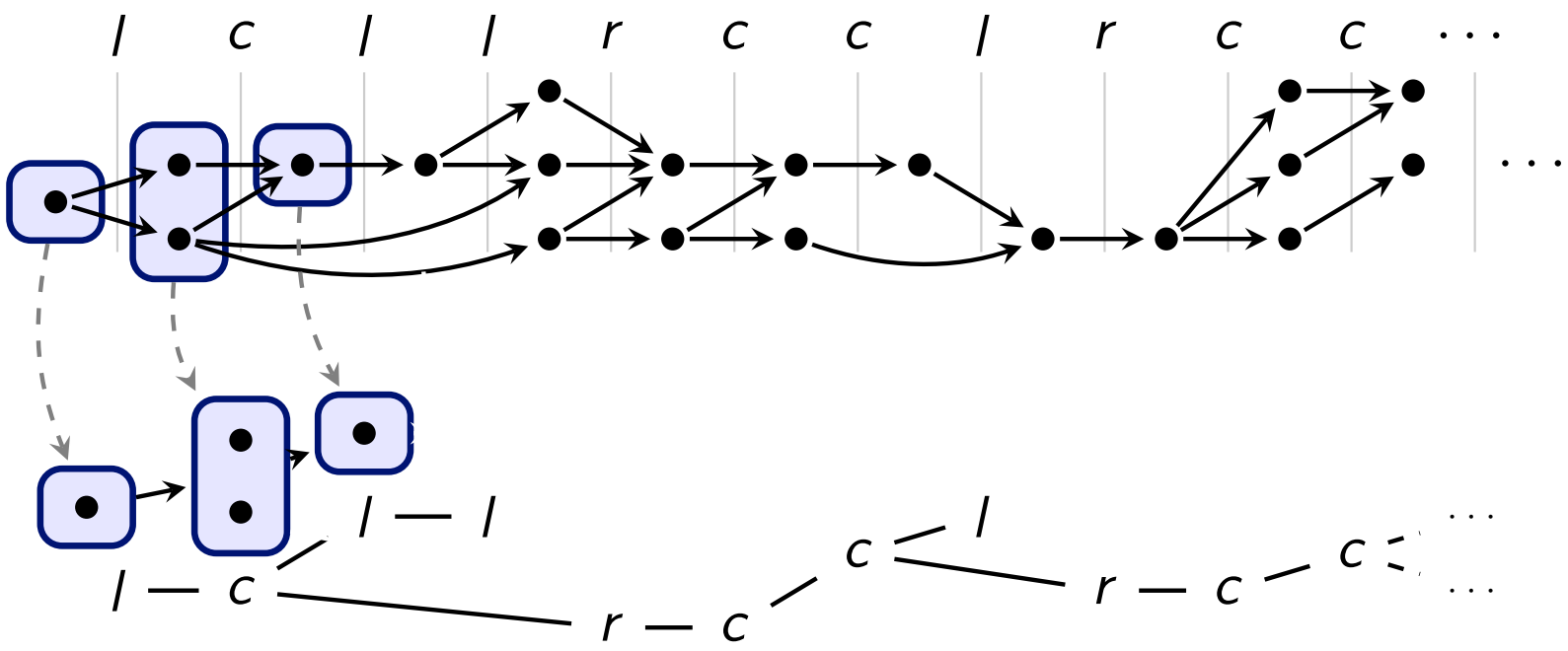
Translating 1 – AJAs into Tree Automata



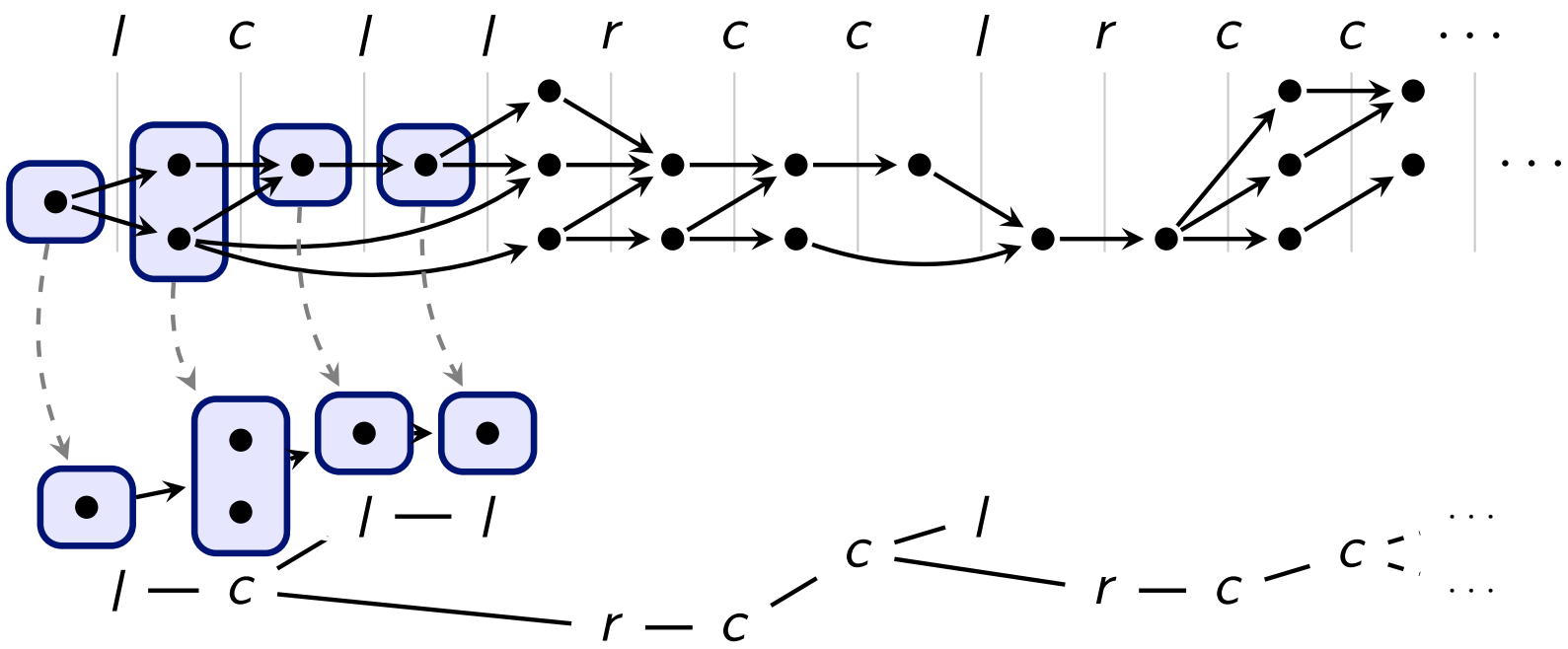
Translating 1 – AJAs into Tree Automata



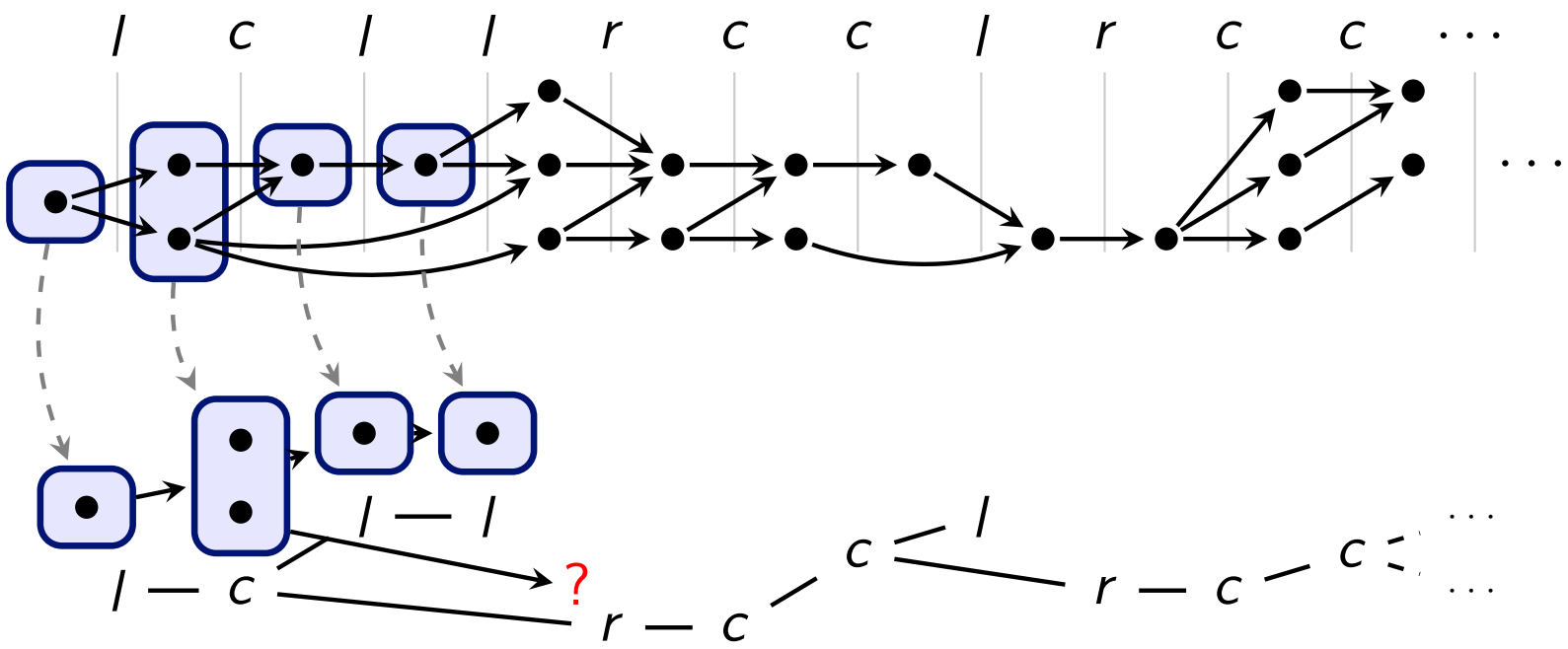
Translating 1 – AJAs into Tree Automata



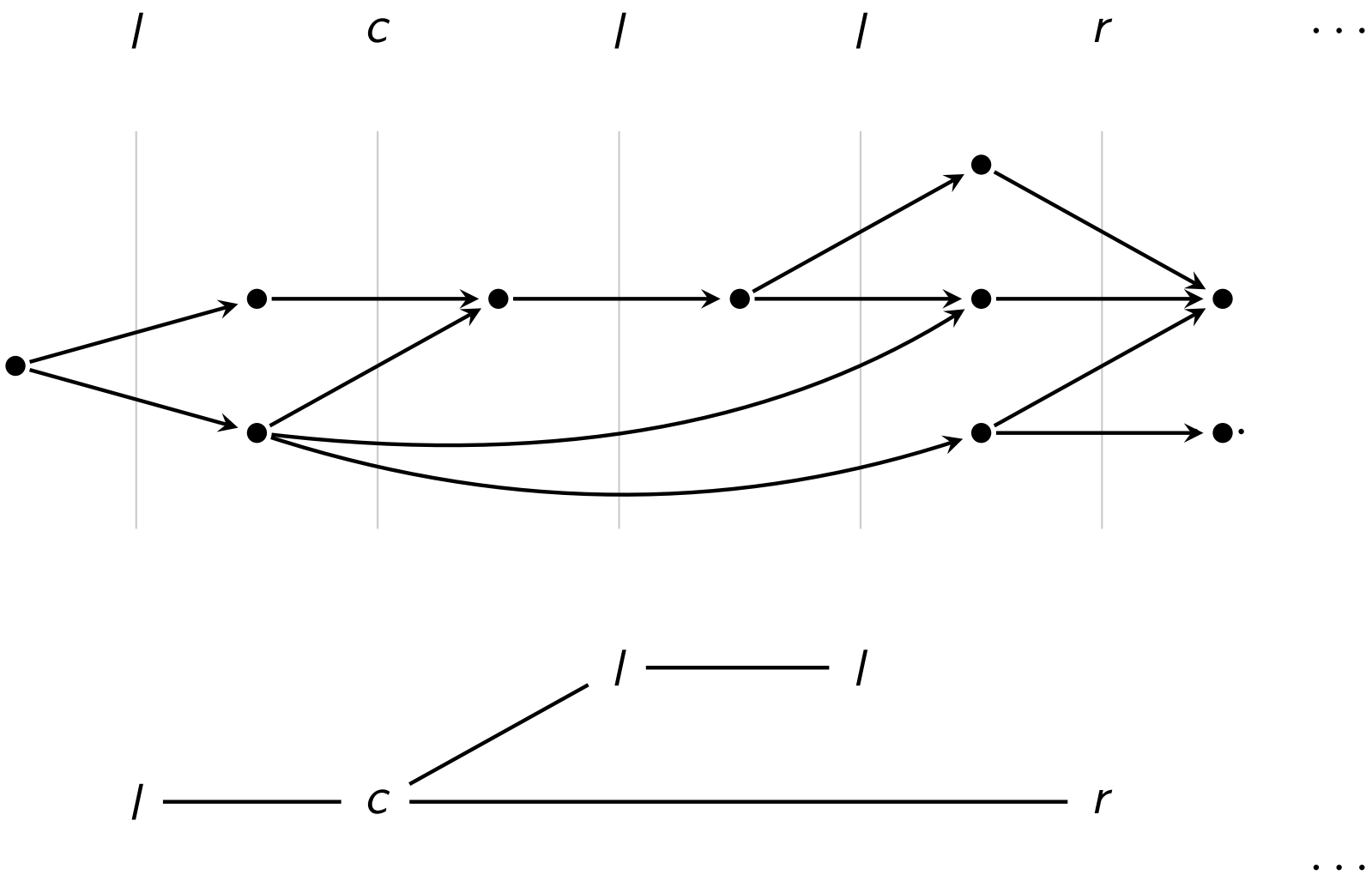
Translating 1 – AJAs into Tree Automata



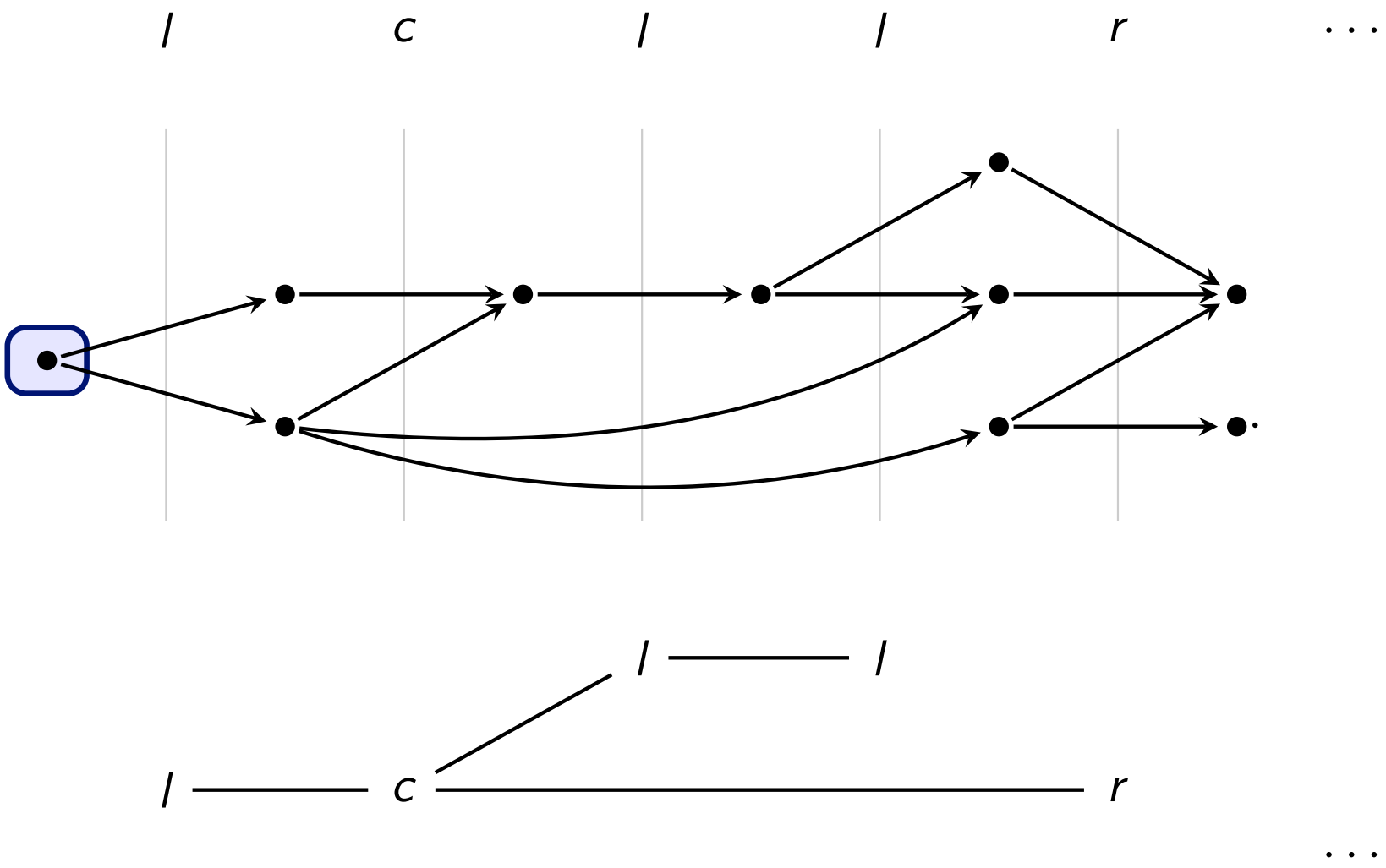
Translating 1 – AJAs into Tree Automata



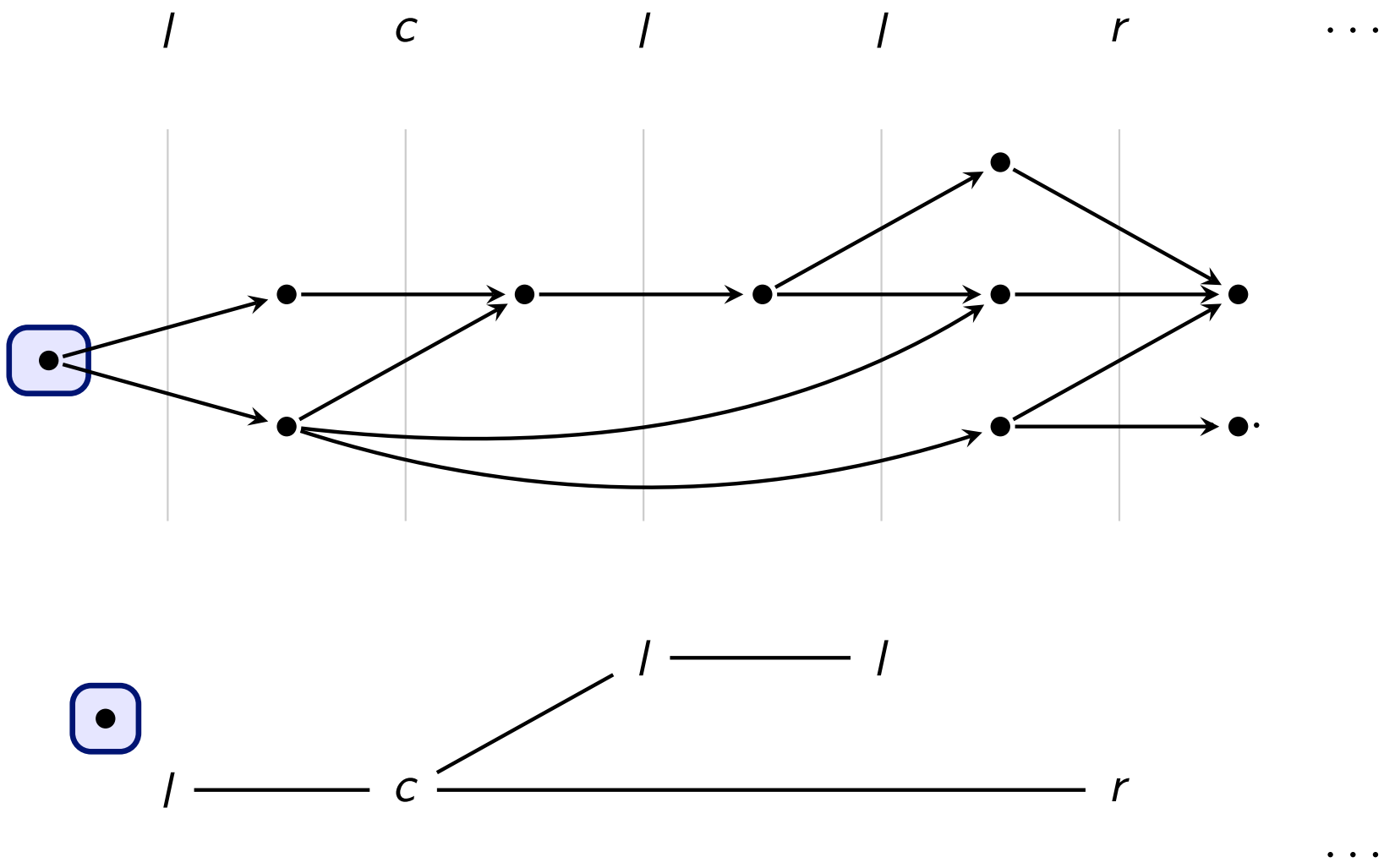
Construction in Detail



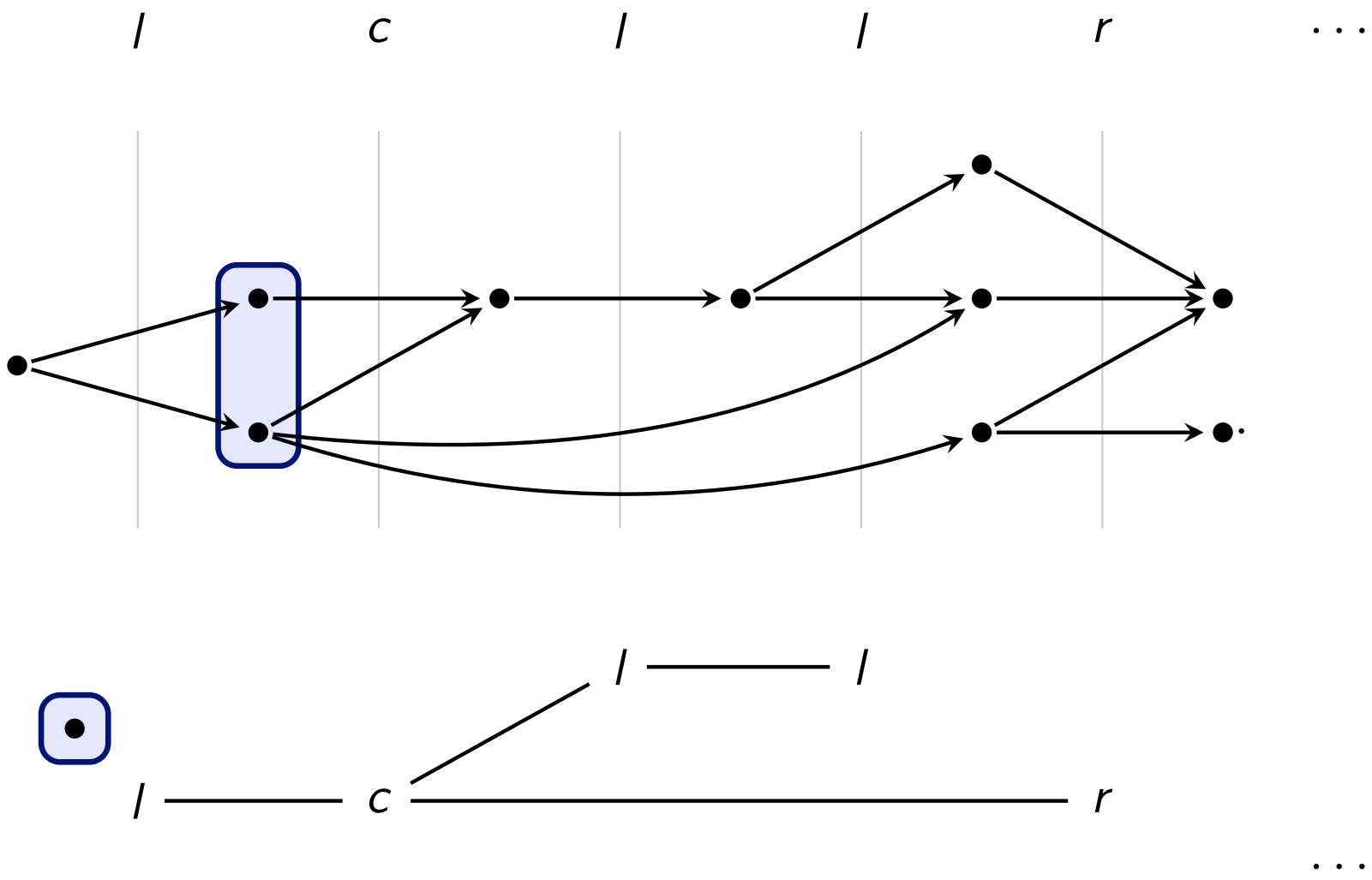
Construction in Detail



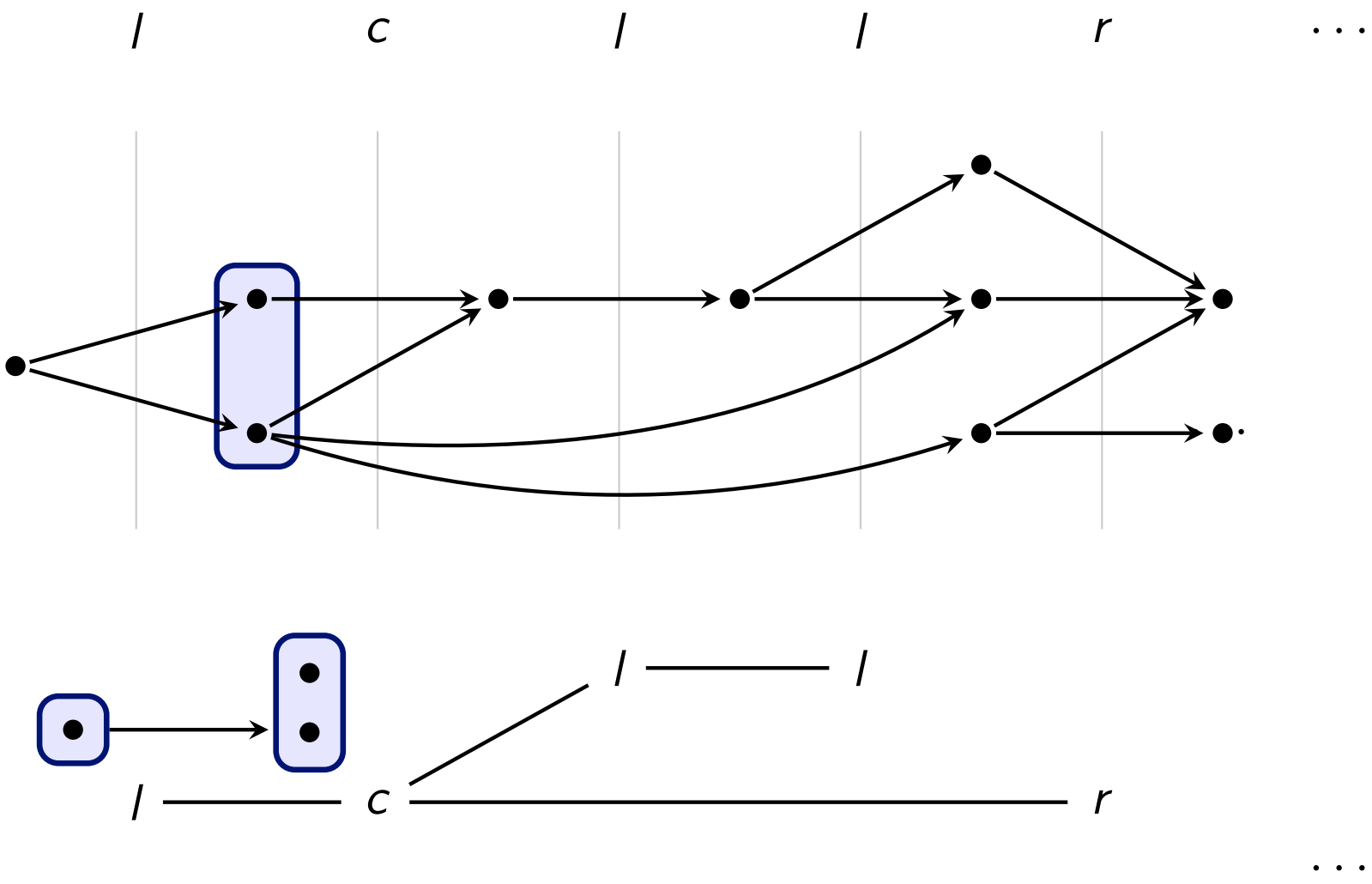
Construction in Detail



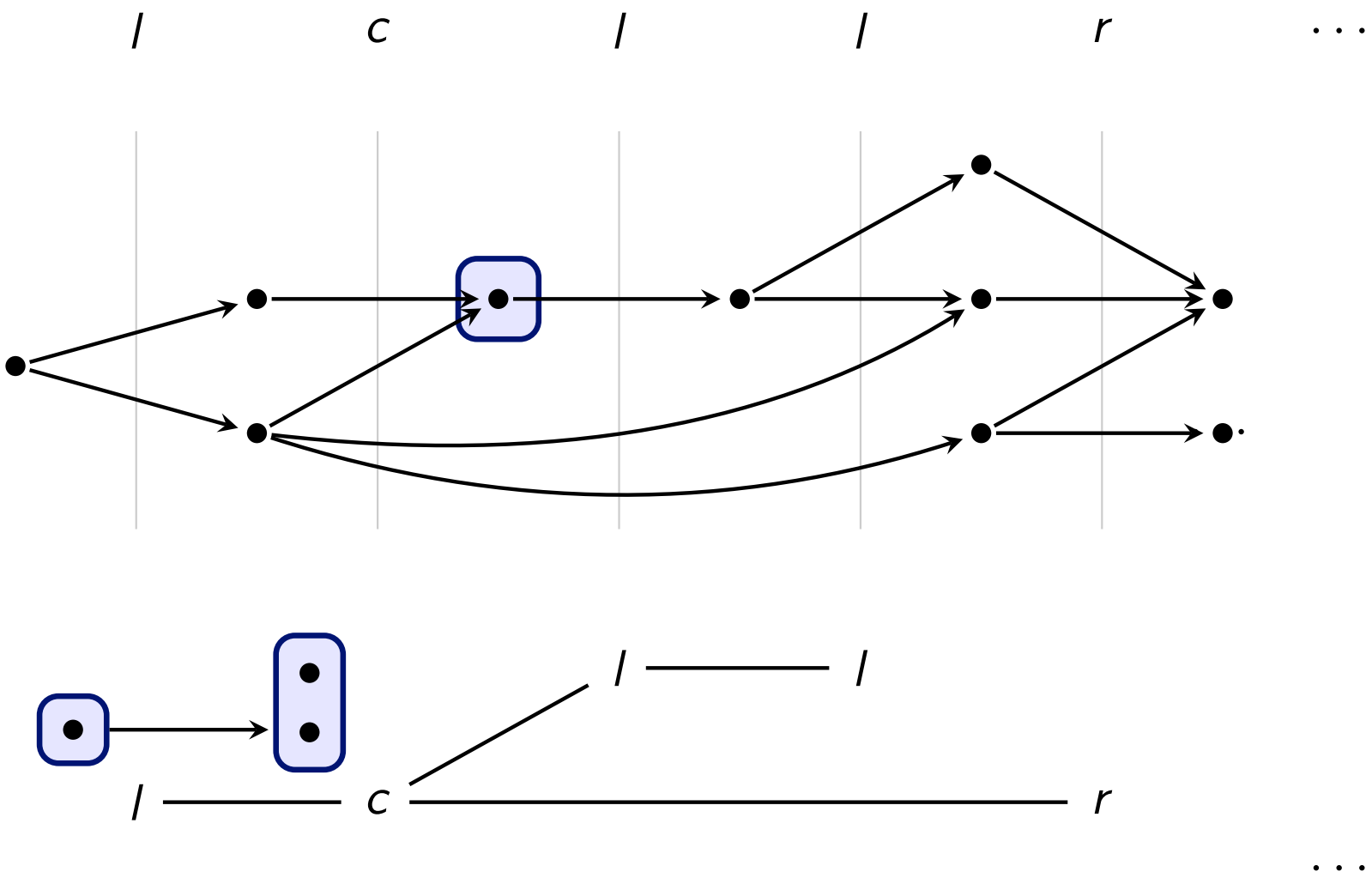
Construction in Detail



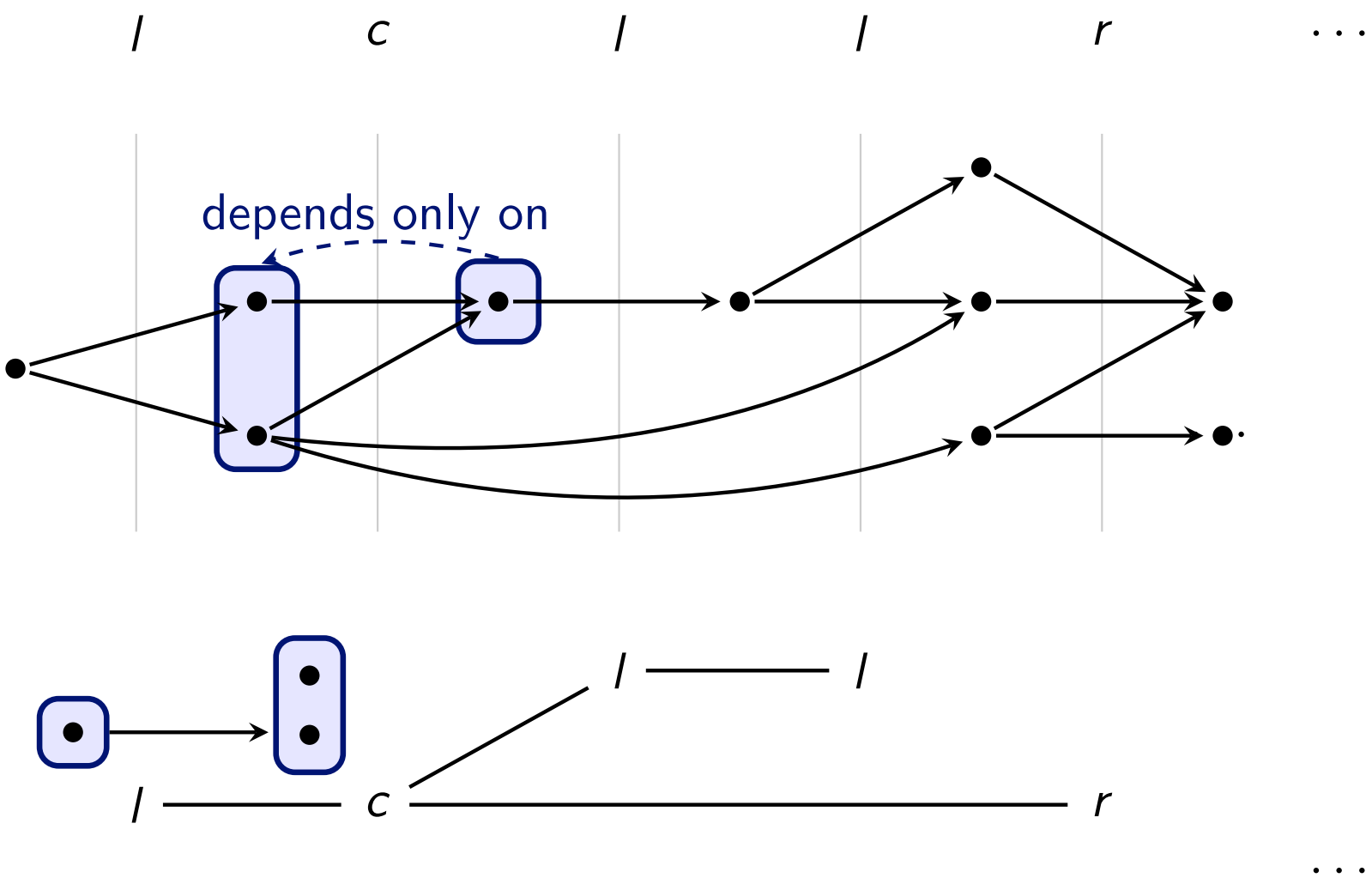
Construction in Detail



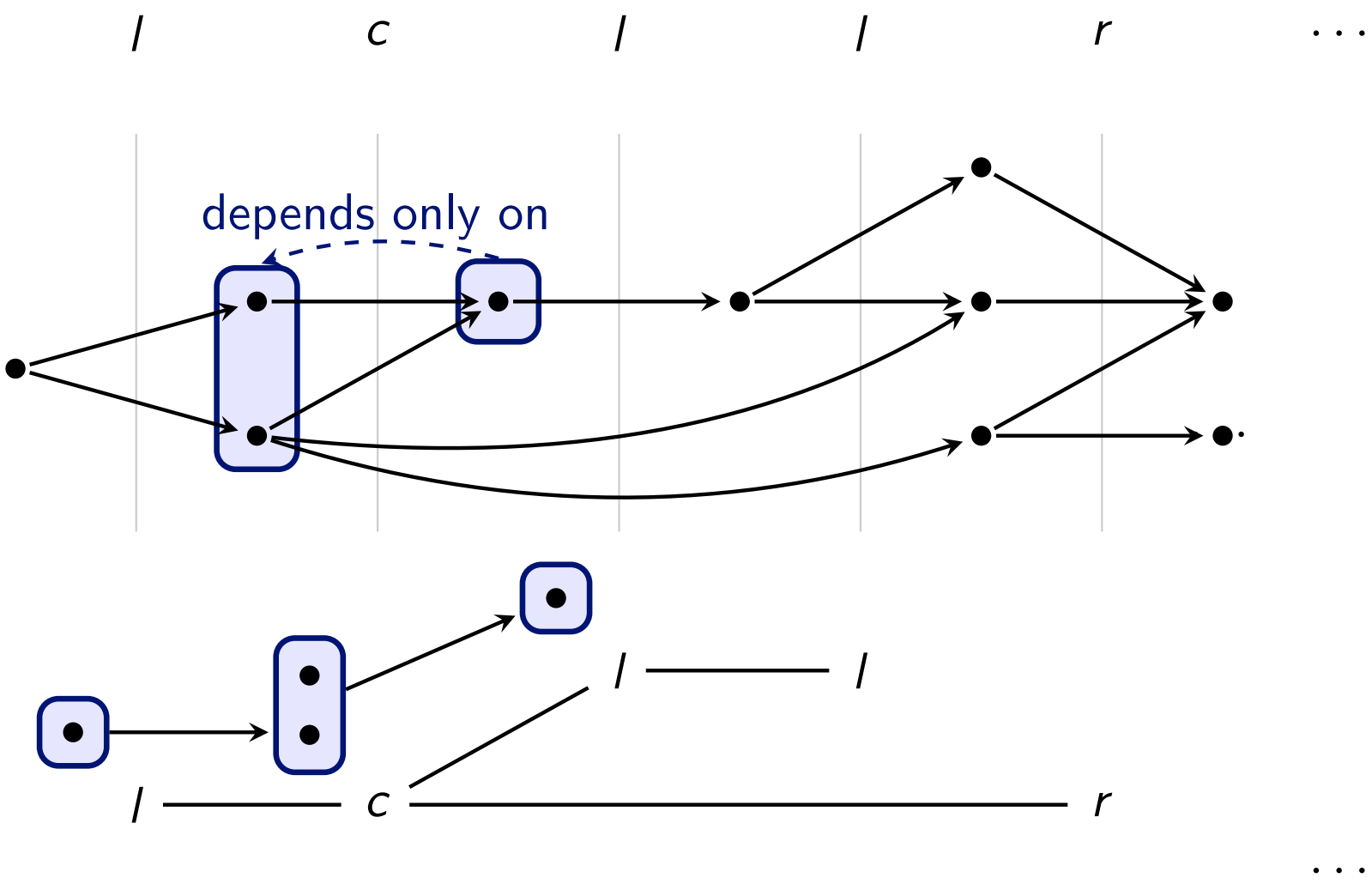
Construction in Detail



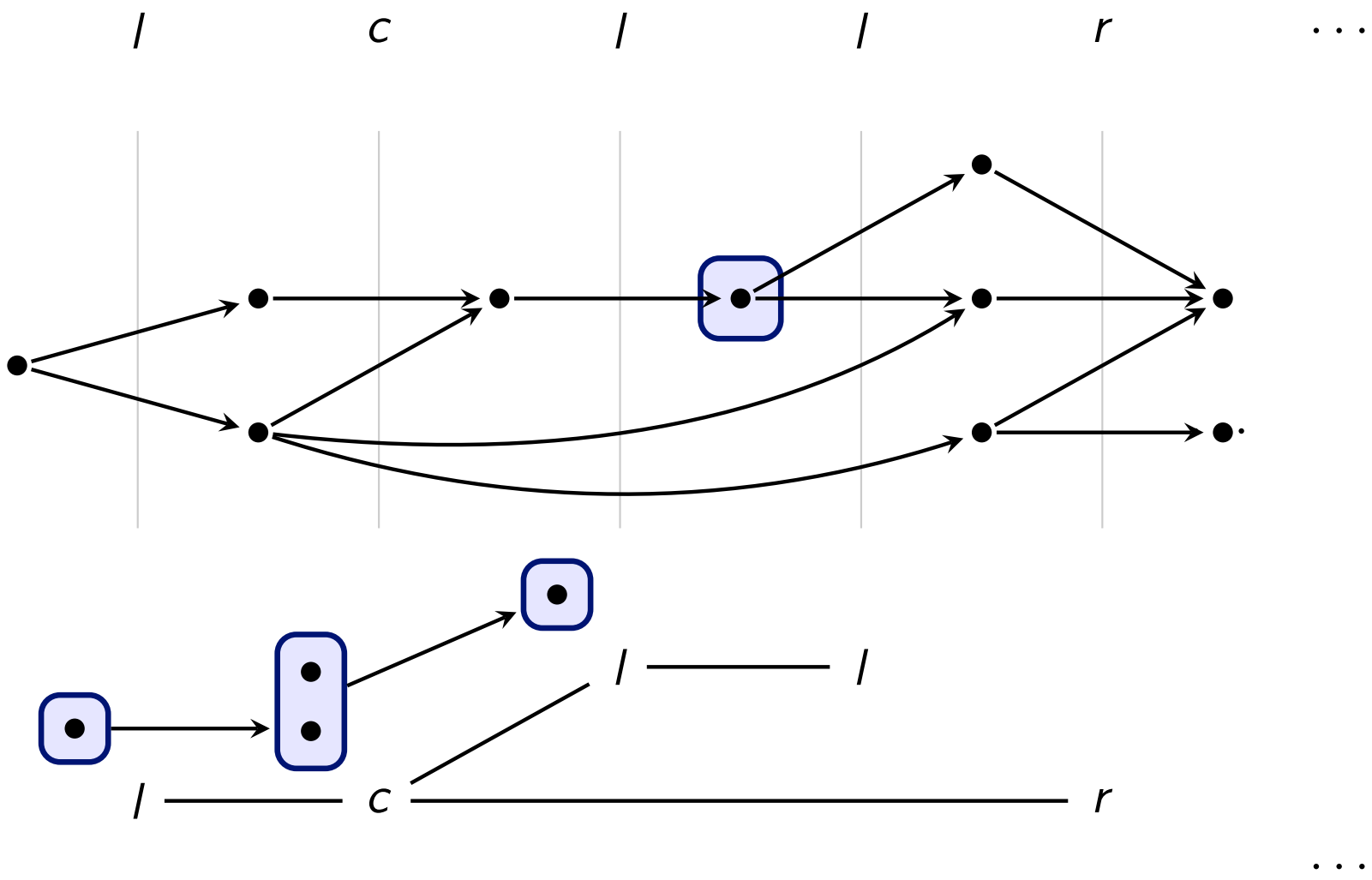
Construction in Detail



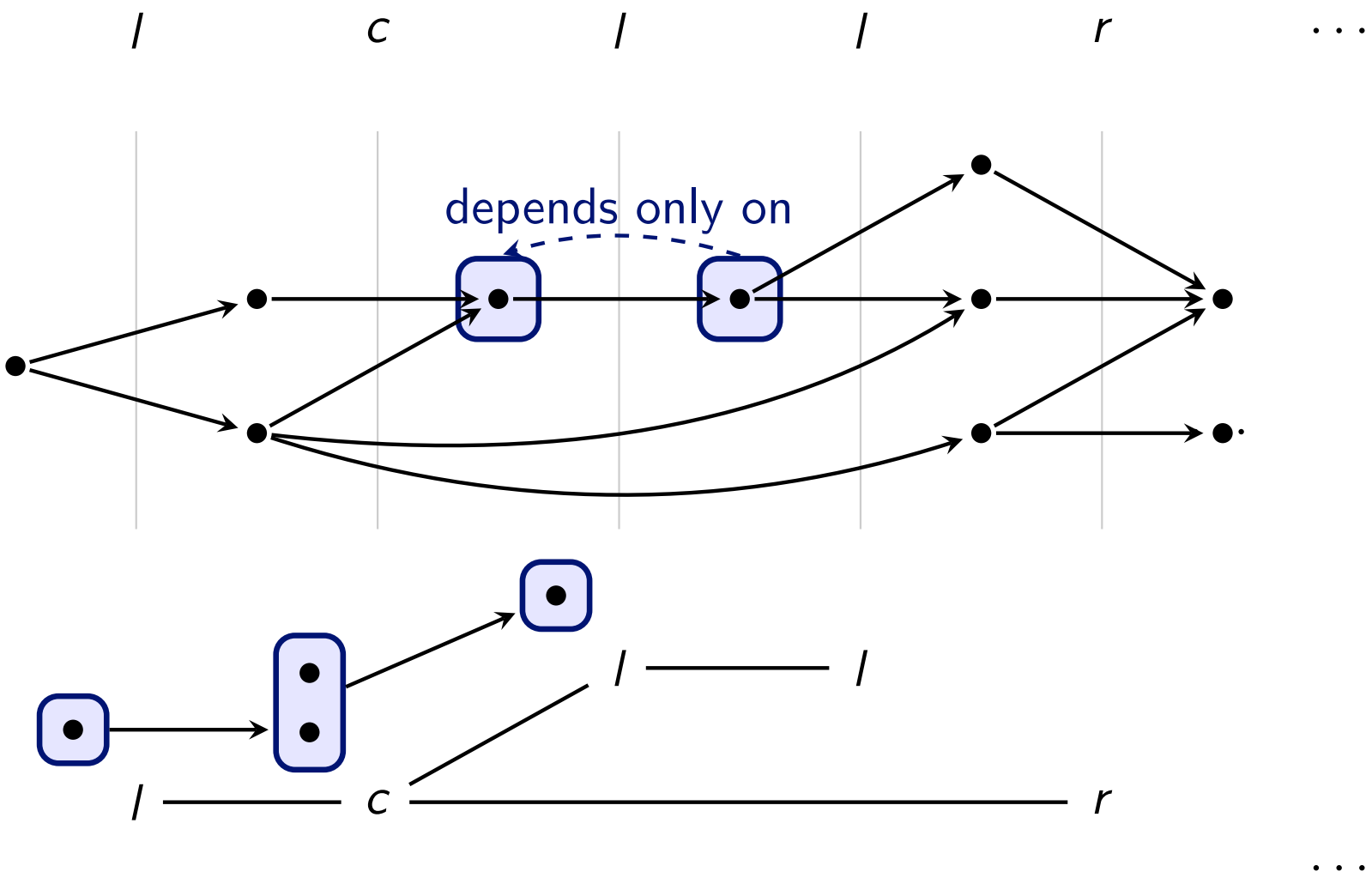
Construction in Detail



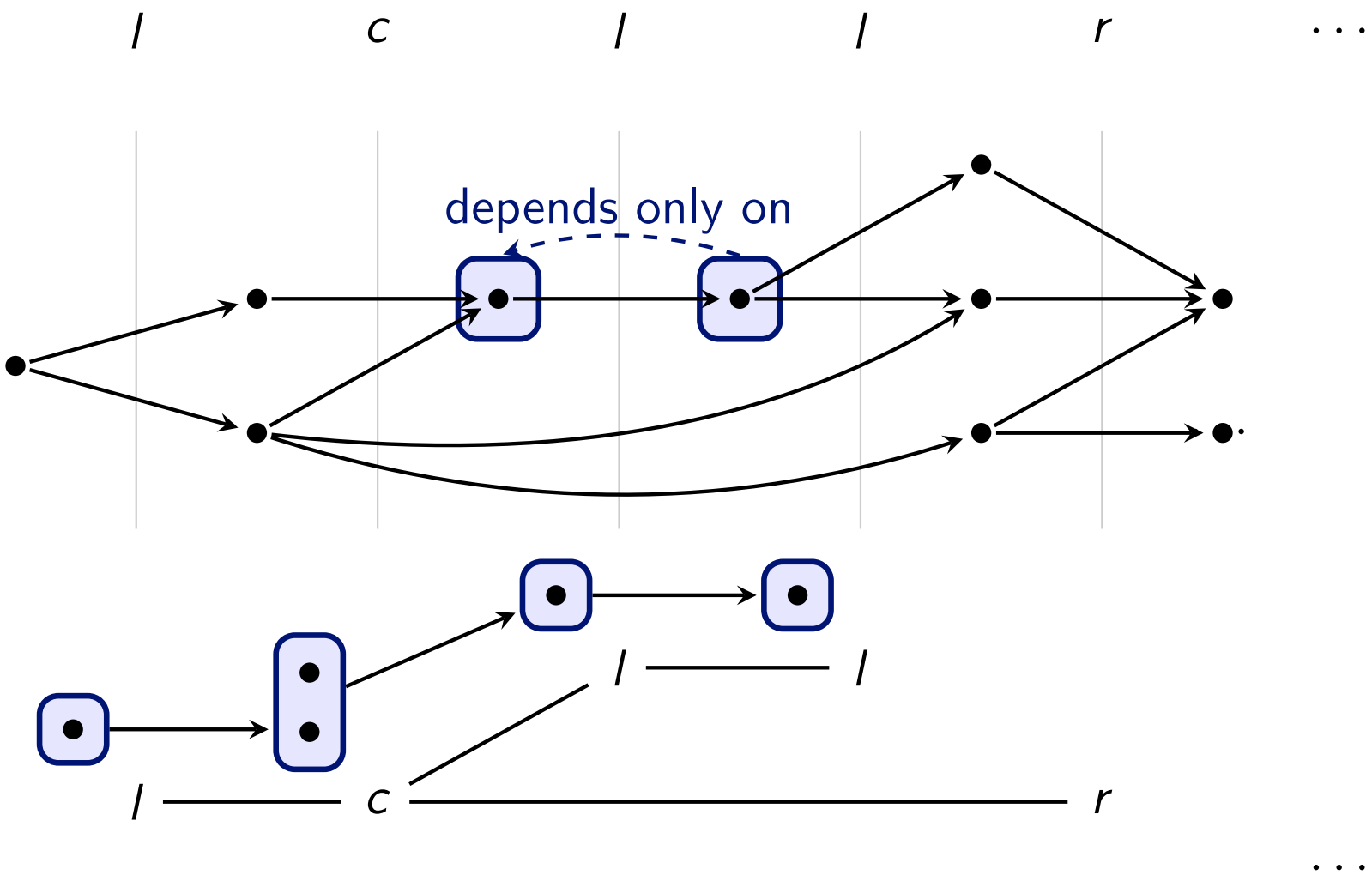
Construction in Detail



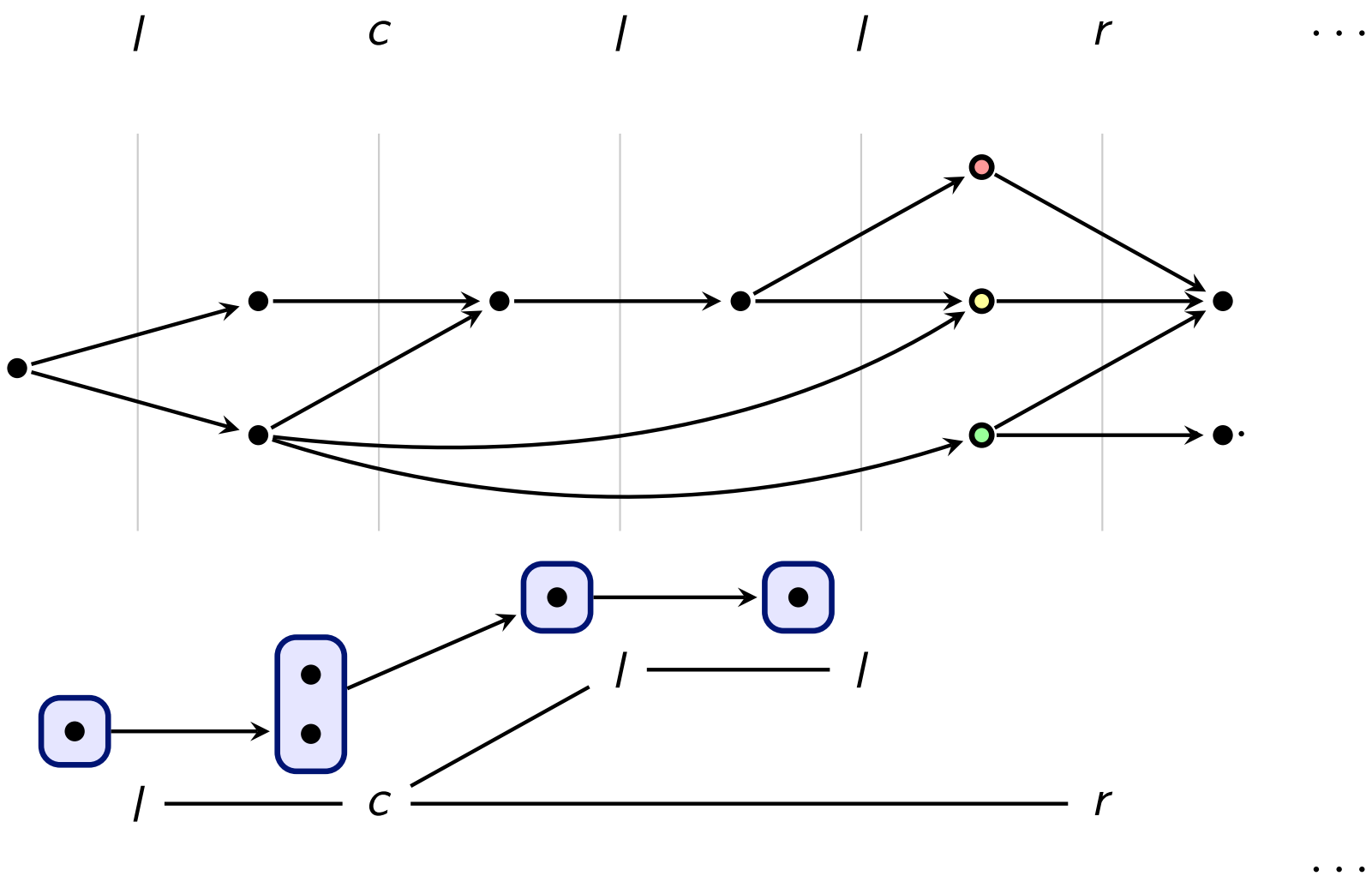
Construction in Detail



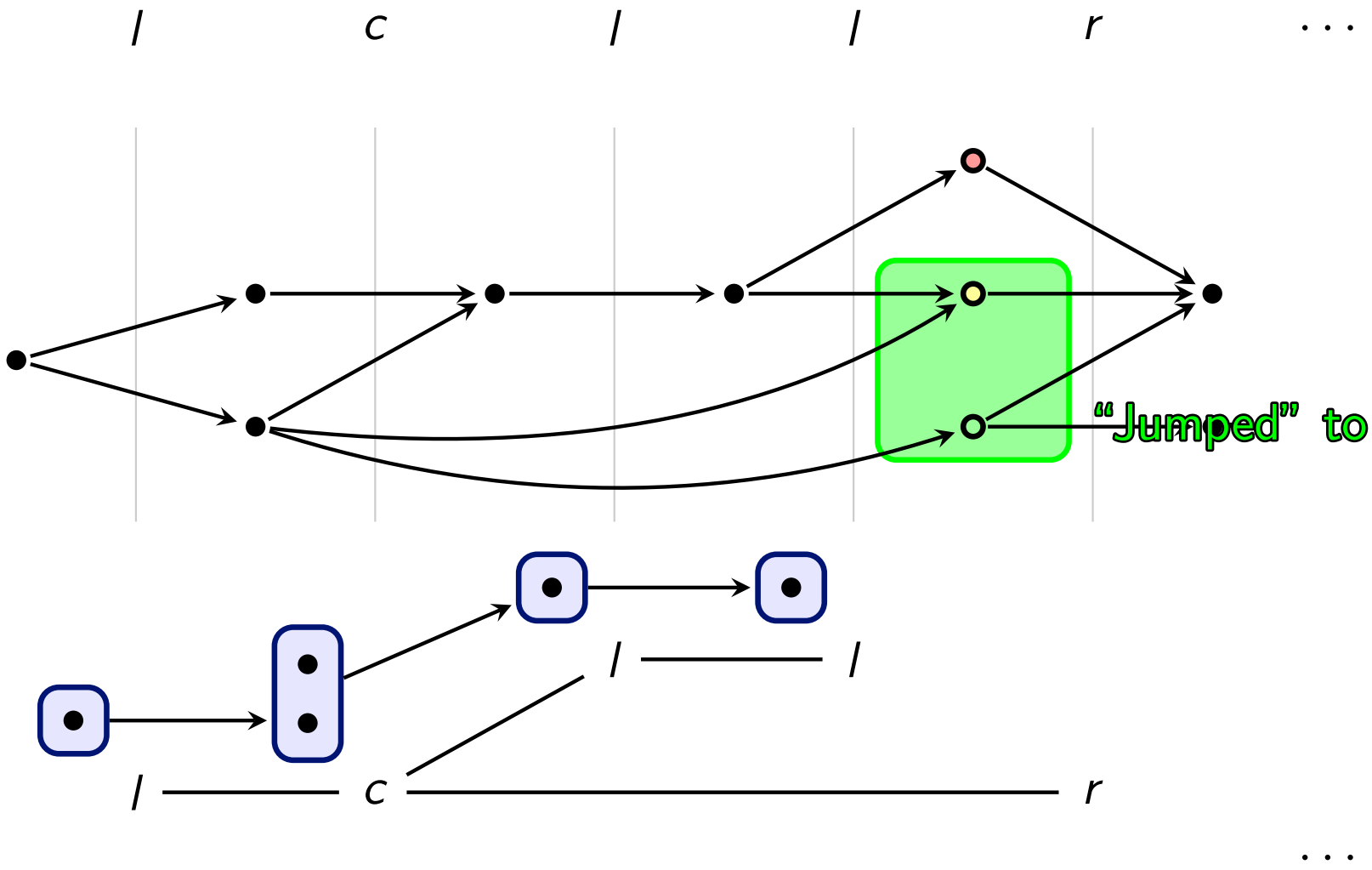
Construction in Detail



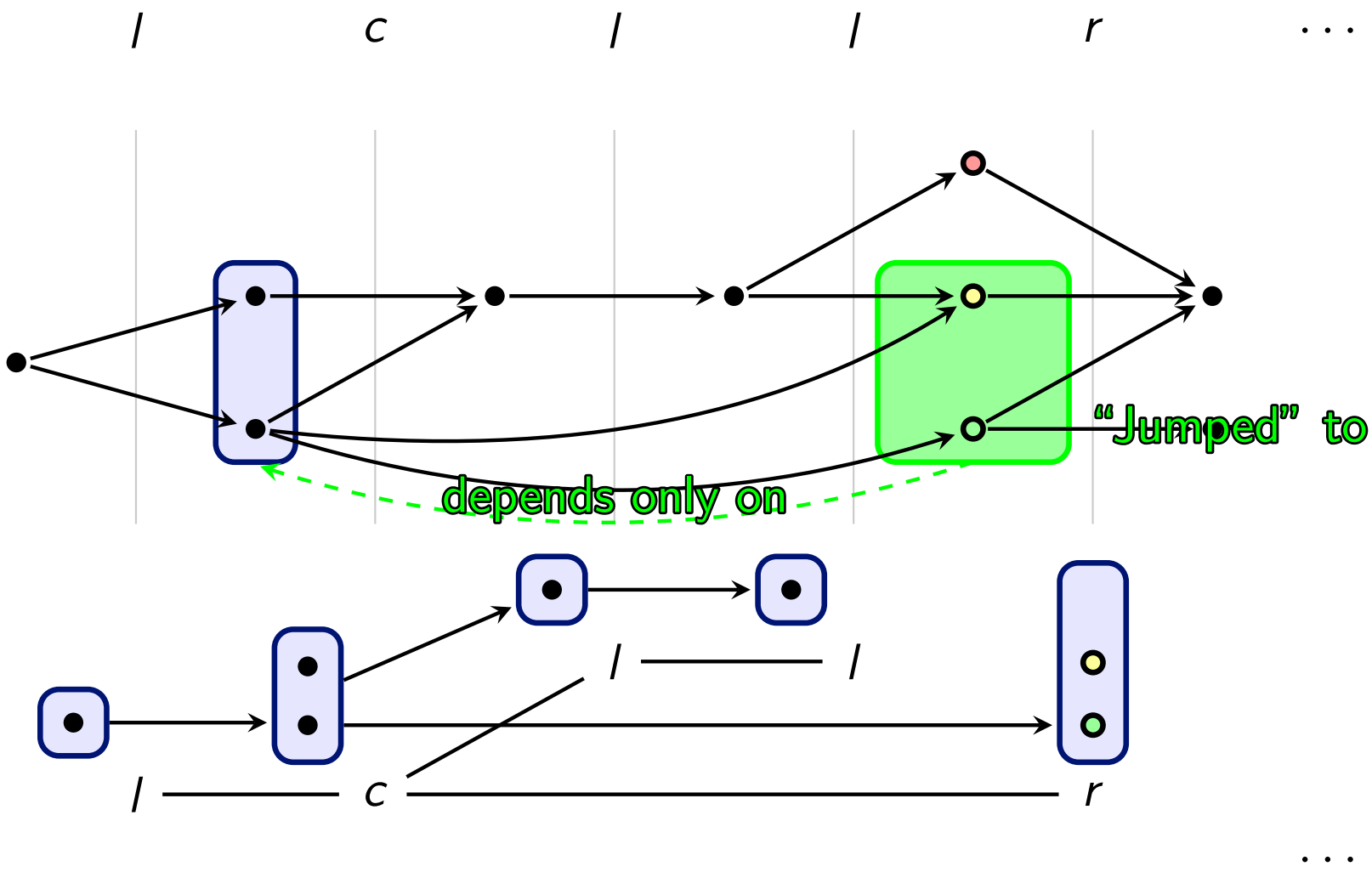
Construction in Detail



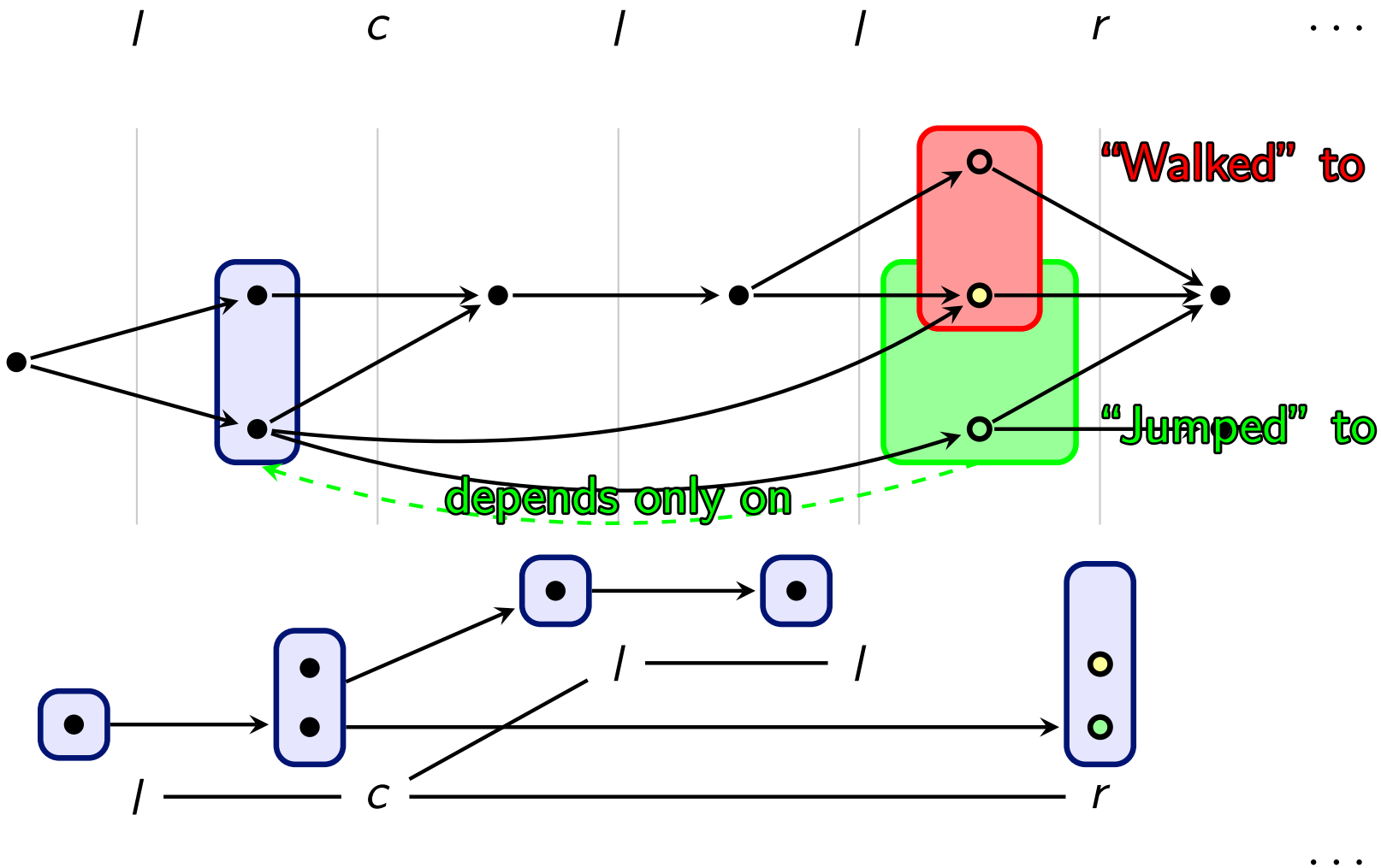
Construction in Detail



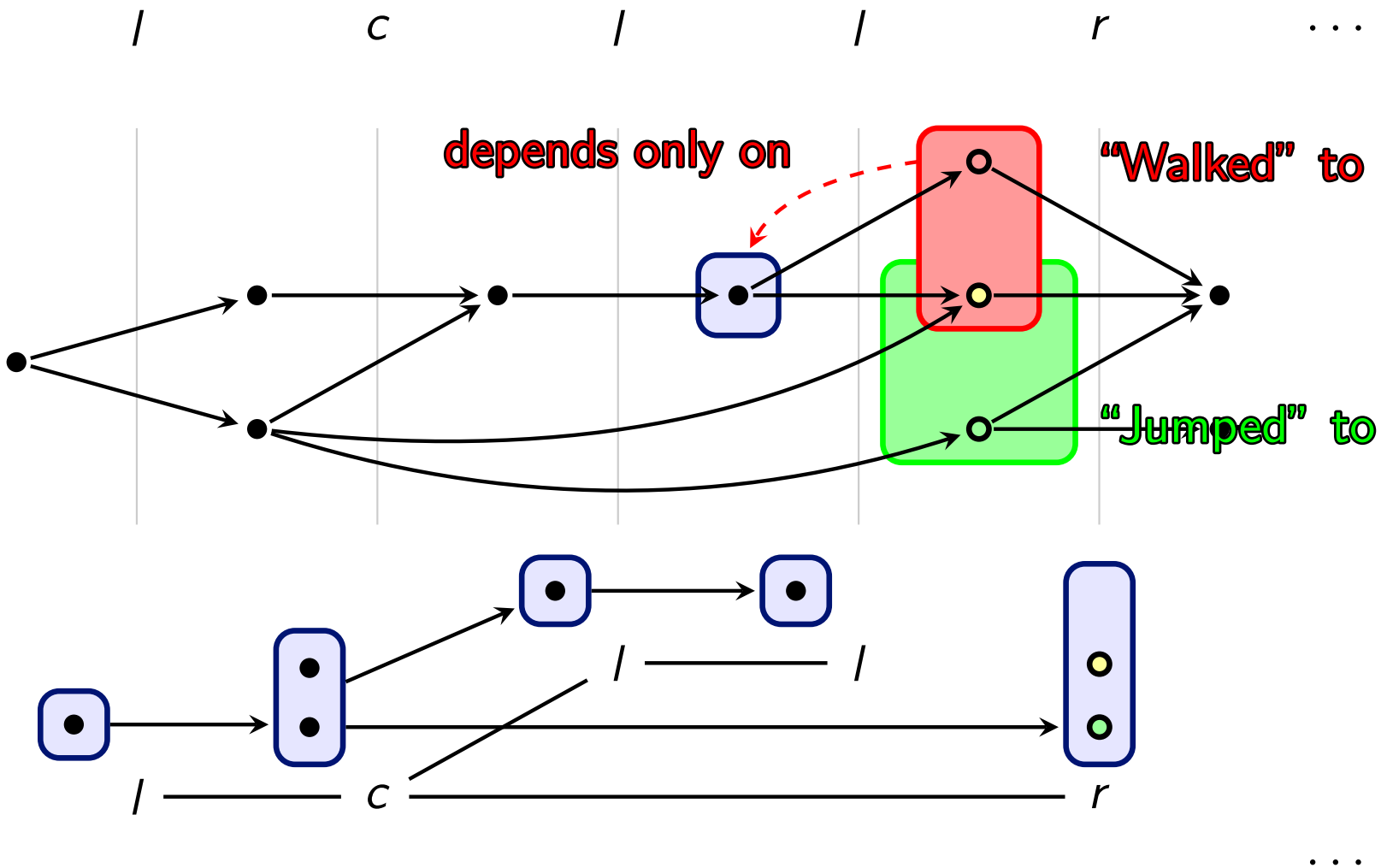
Construction in Detail



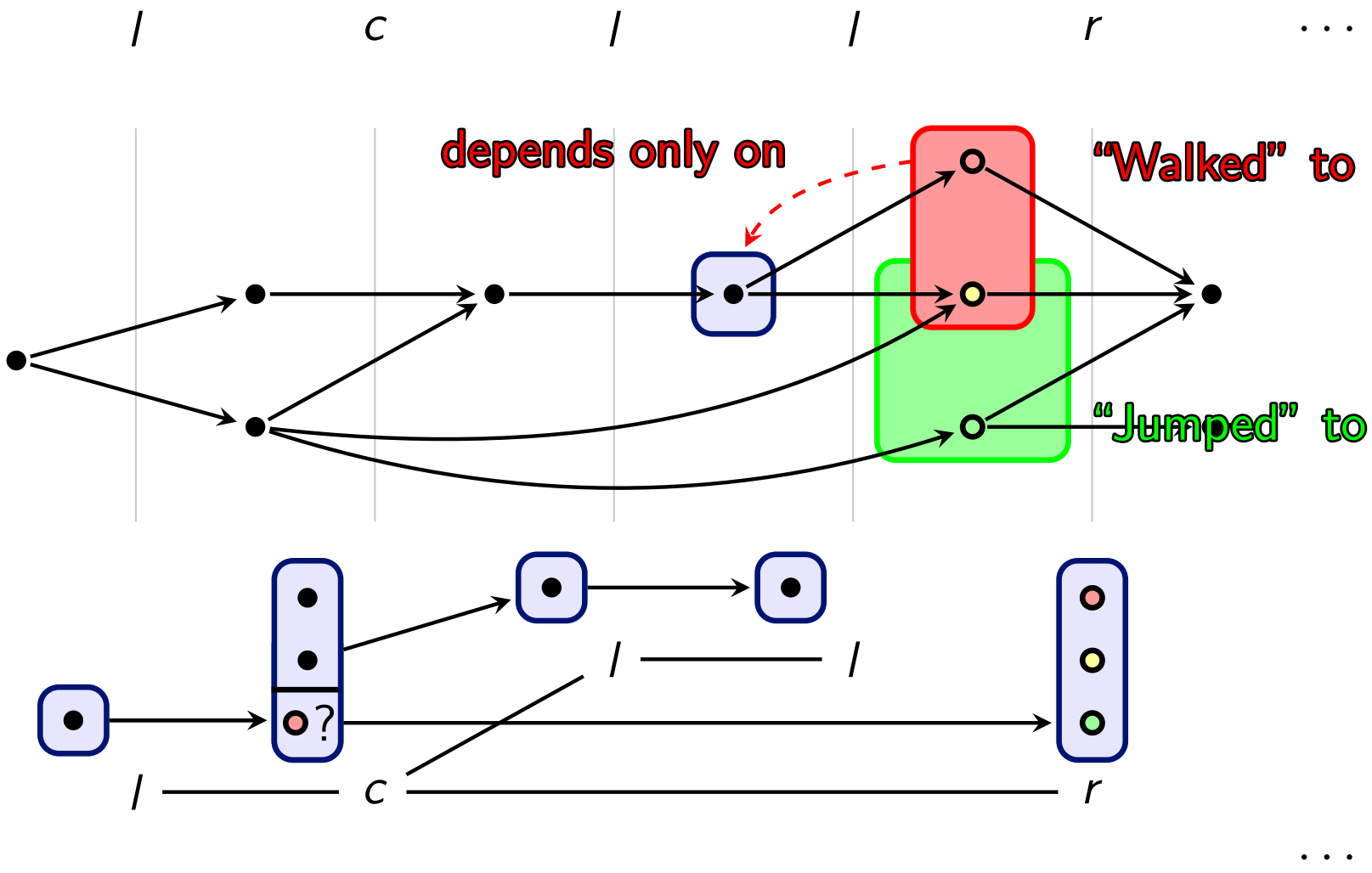
Construction in Detail



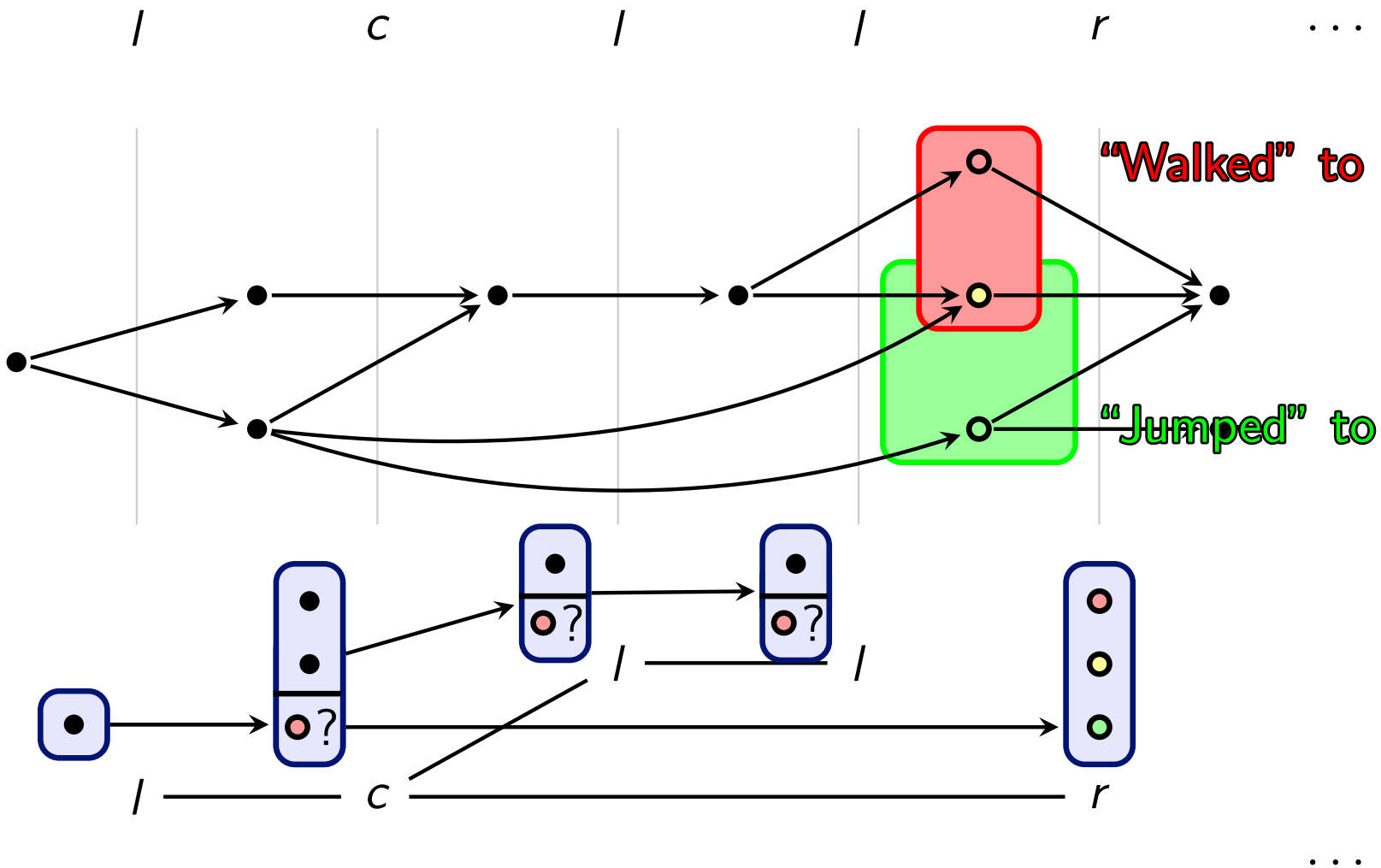
Construction in Detail



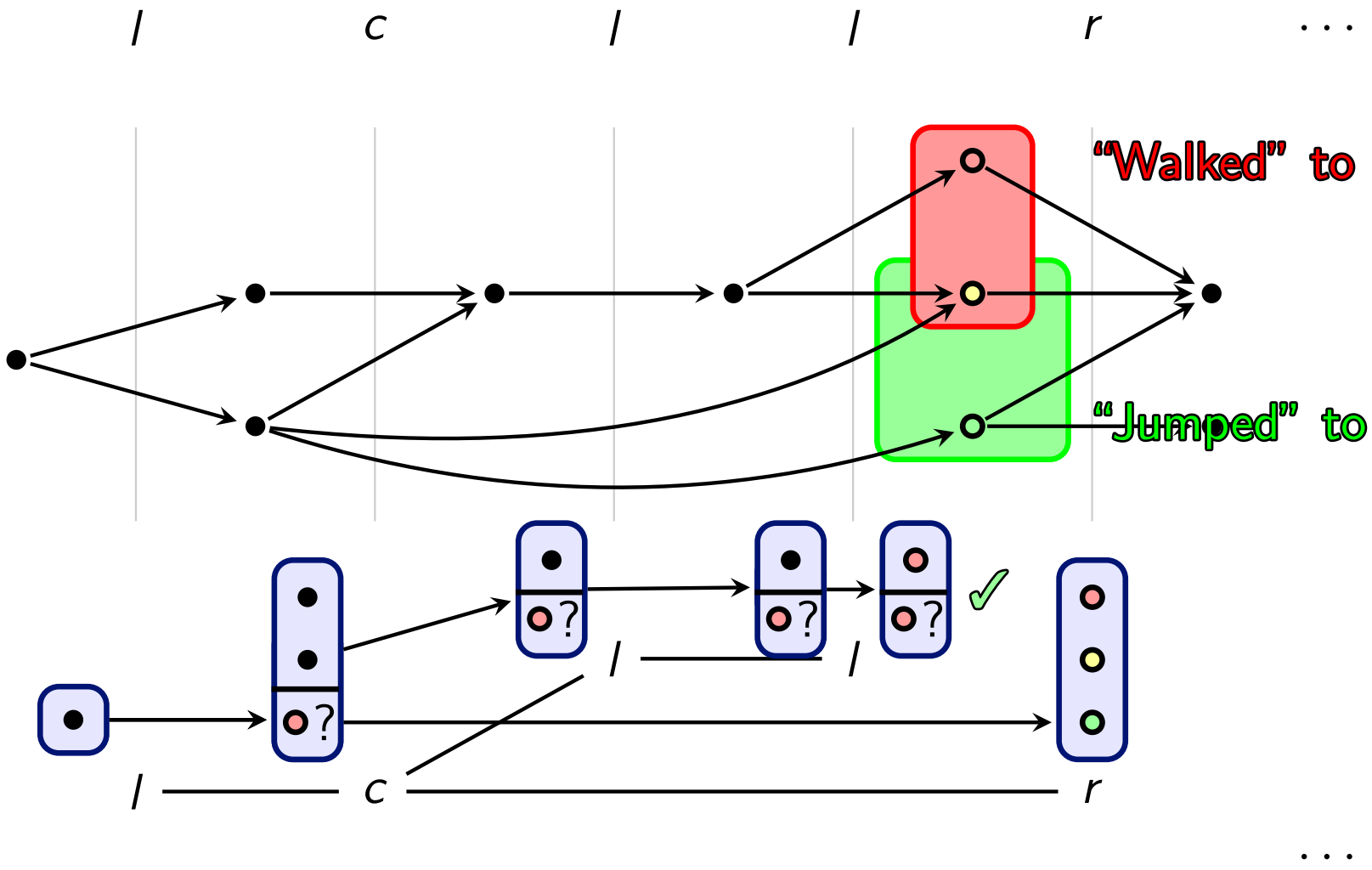
Construction in Detail



Construction in Detail

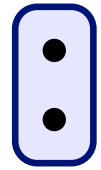


Construction in Detail



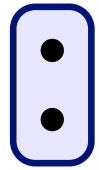
Summary of Construction

1. Subset construction

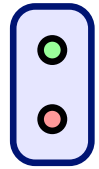


Summary of Construction

1. Subset construction

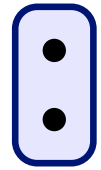


2. Split into “jump”- and “walk”-states

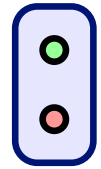


Summary of Construction

1. Subset construction



2. Split into “jump”- and “walk”-states



3. Guess and verify

Constructing a Tree Automaton

Component	Technique
States	
Transitions	
Accepting States	

Constructing a Tree Automaton

Component	Technique	
States	Subset Construction	✓
Transitions		
Accepting States		

Constructing a Tree Automaton

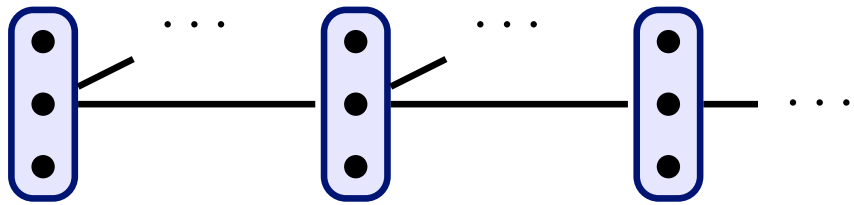
Component	Technique	
States	Subset Construction	✓
Transitions	Guess/Verify	✓
Accepting States		

Constructing a Tree Automaton

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

Accepting States

Run of the tree automaton:



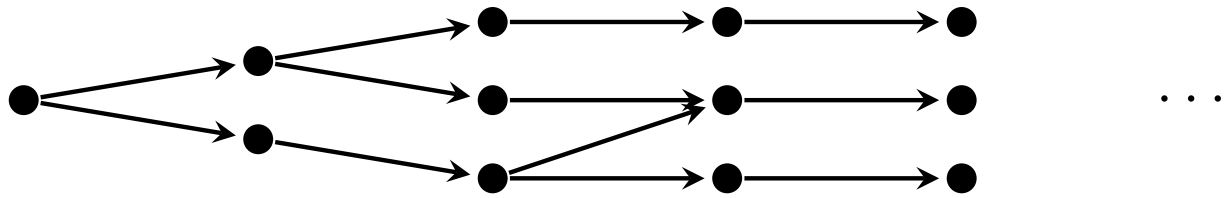
Recall:

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

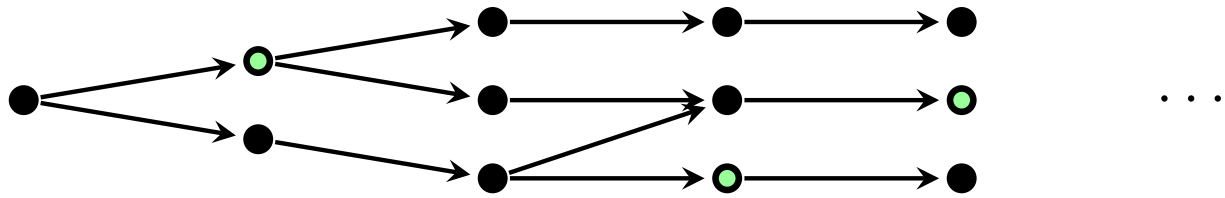
Solution: Lift acceptance condition from paths to states

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

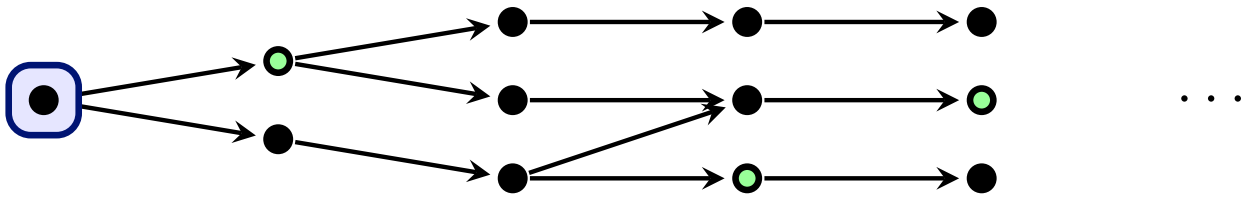
Breakpoint Construction



Breakpoint Construction

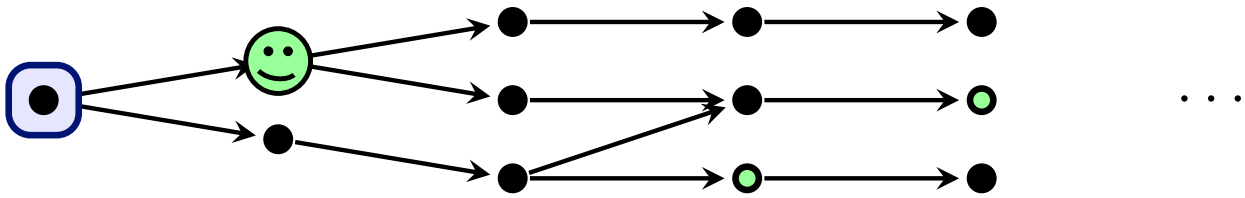


Breakpoint Construction

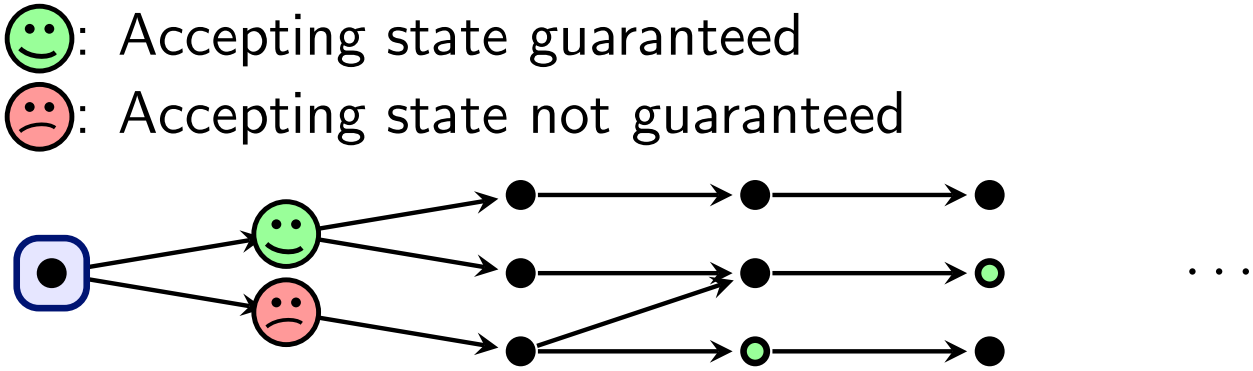


Breakpoint Construction

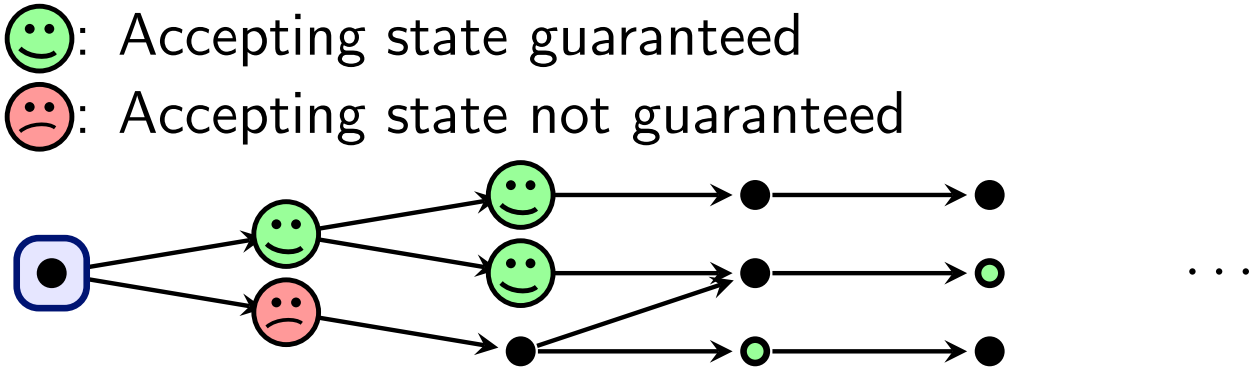
: Accepting state guaranteed



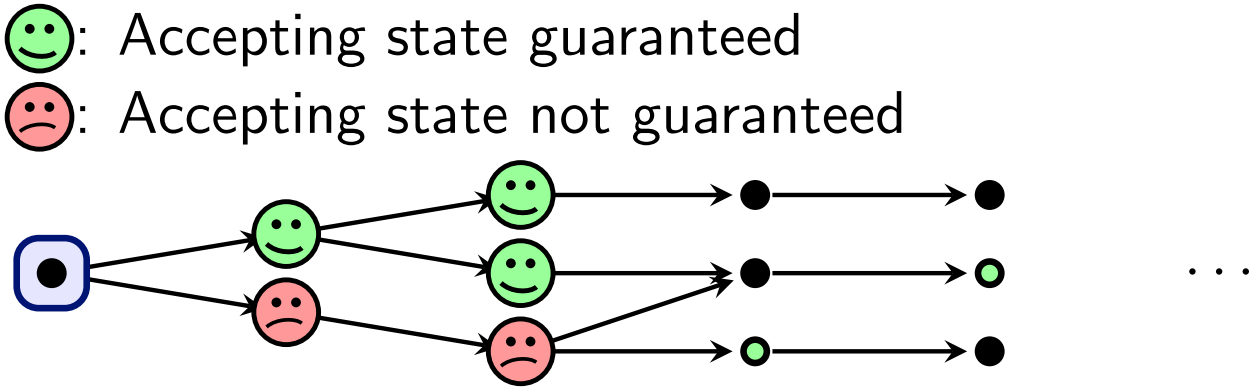
Breakpoint Construction



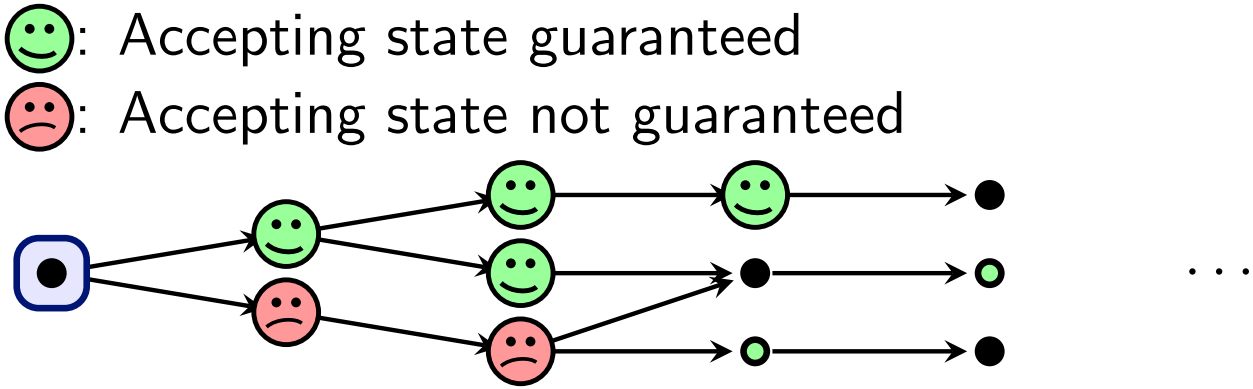
Breakpoint Construction



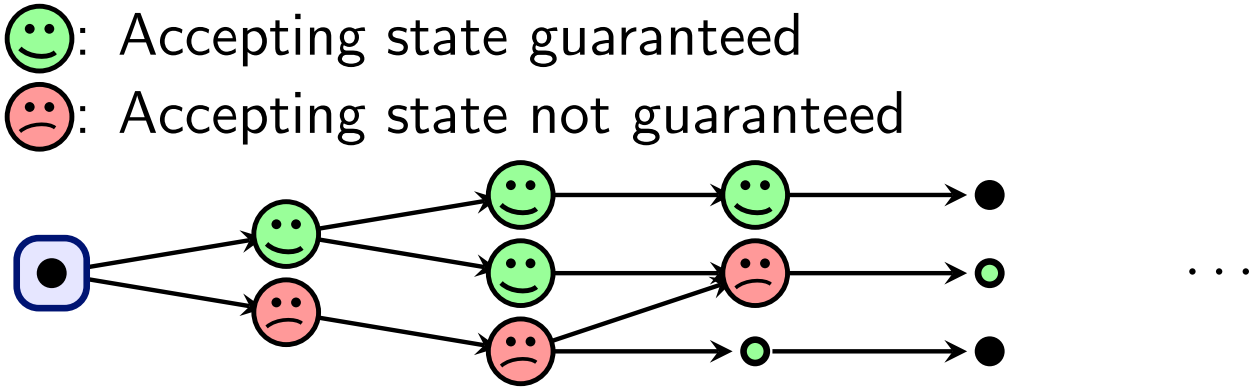
Breakpoint Construction



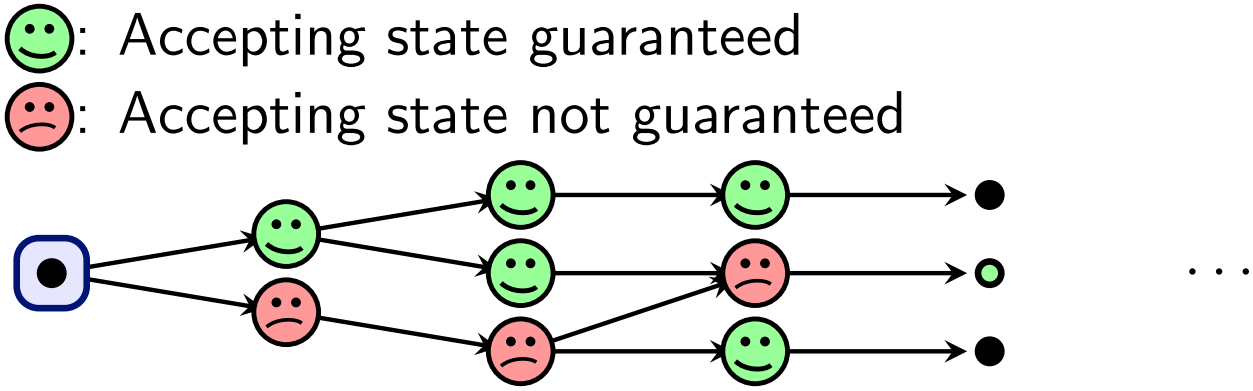
Breakpoint Construction



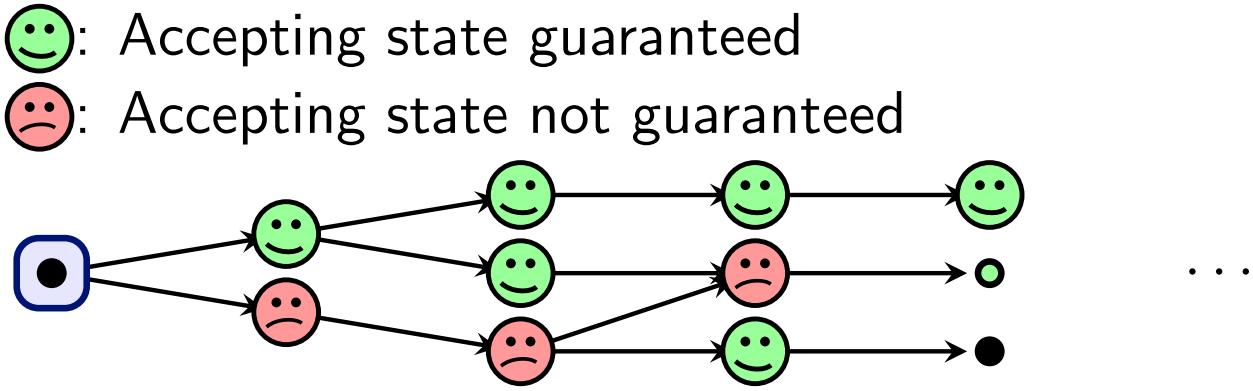
Breakpoint Construction



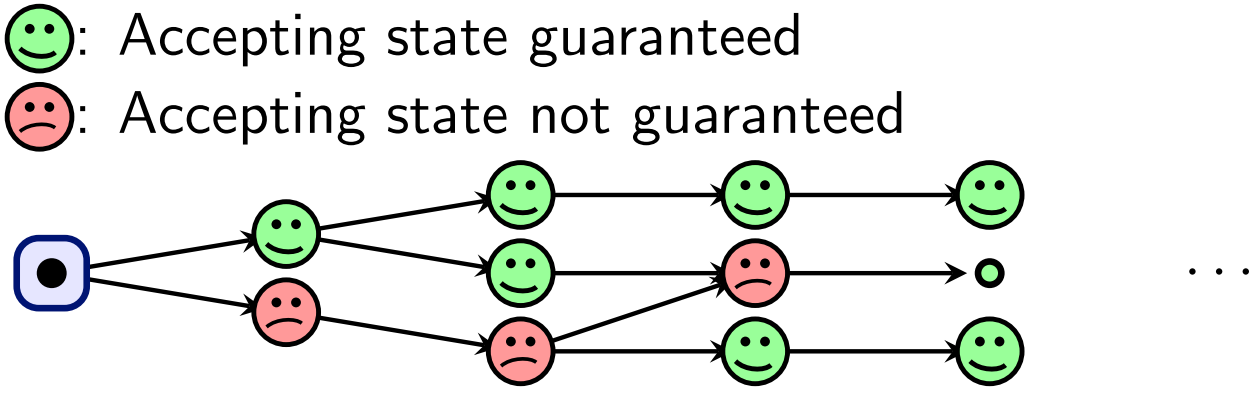
Breakpoint Construction



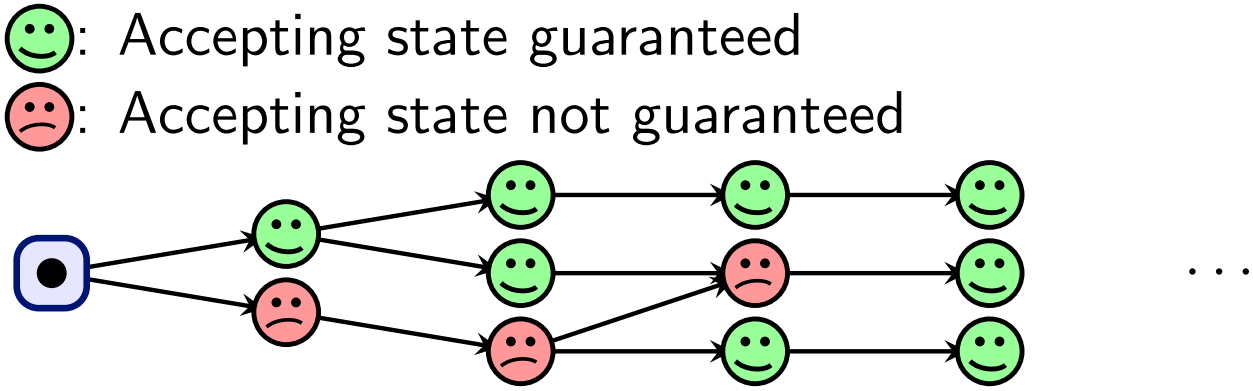
Breakpoint Construction



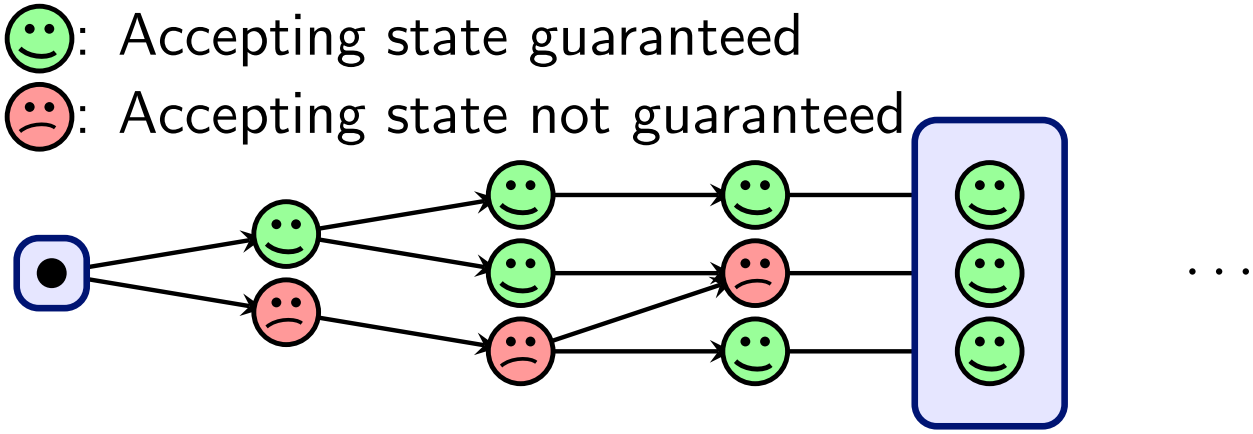
Breakpoint Construction



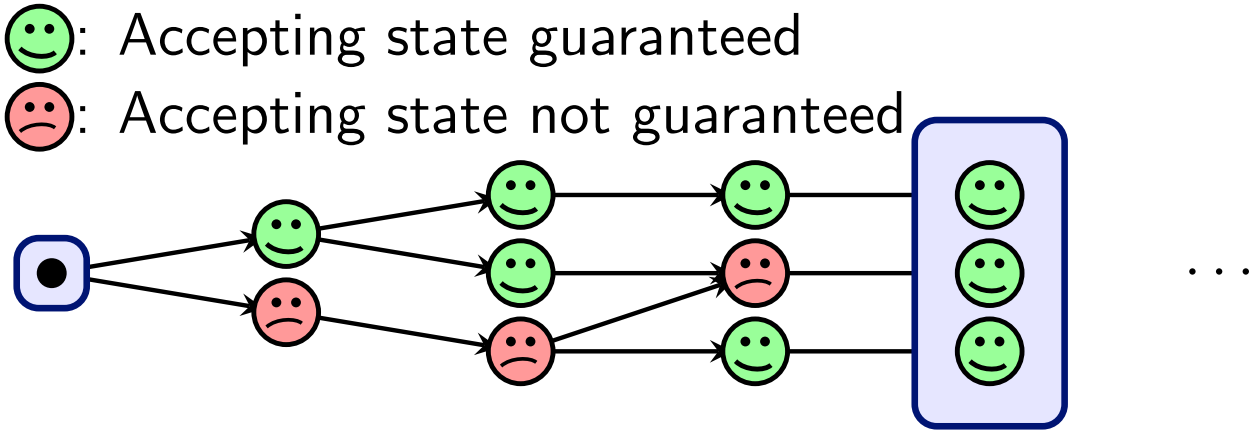
Breakpoint Construction



Breakpoint Construction

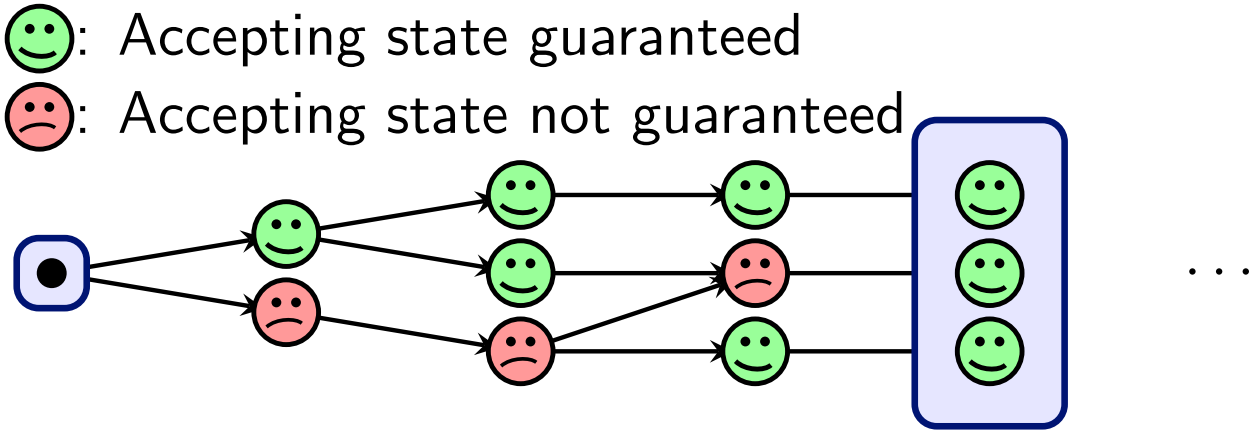


Breakpoint Construction



Breakpoint: All paths since last breakpoint visit accepting state

Breakpoint Construction

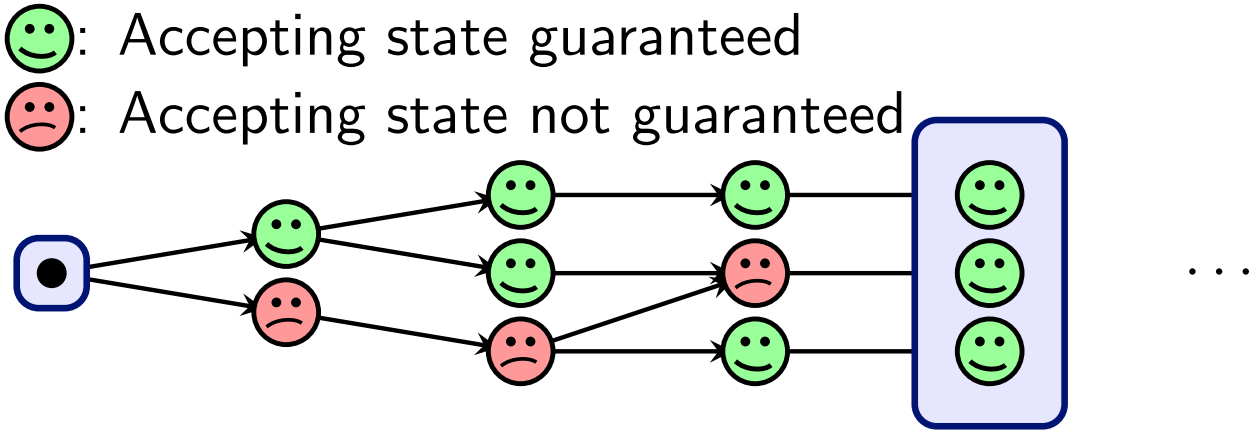


Breakpoint: All paths since last breakpoint visit accepting state

Lemma (Miyano and Hayashi, 1984)

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

Breakpoint Construction



Breakpoint: All paths since last breakpoint visit accepting state

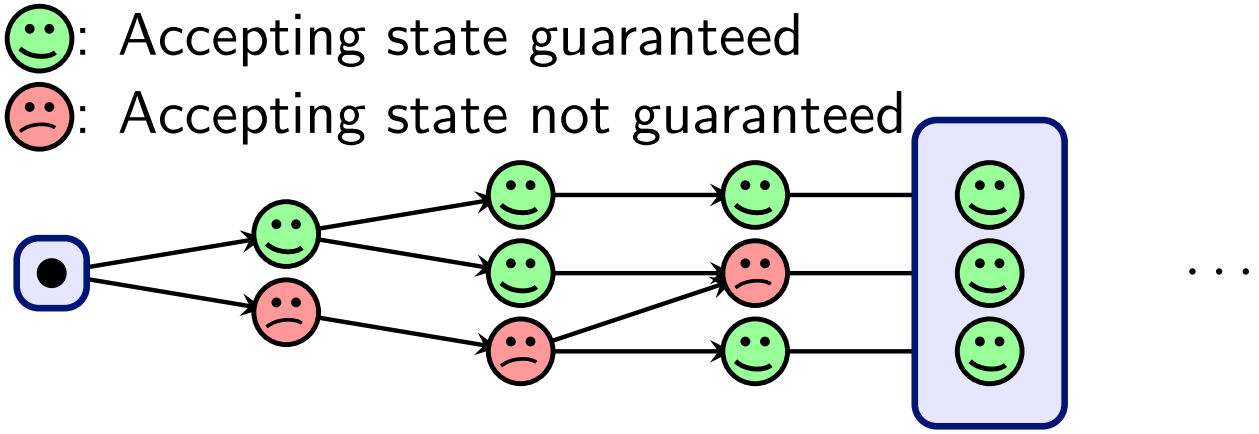
Lemma (Miyano and Hayashi, 1984)

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.*

Breakpoint Construction



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

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

Accepting States

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		

Constructing a Tree Automaton

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

Constructing a Tree Automaton

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

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.

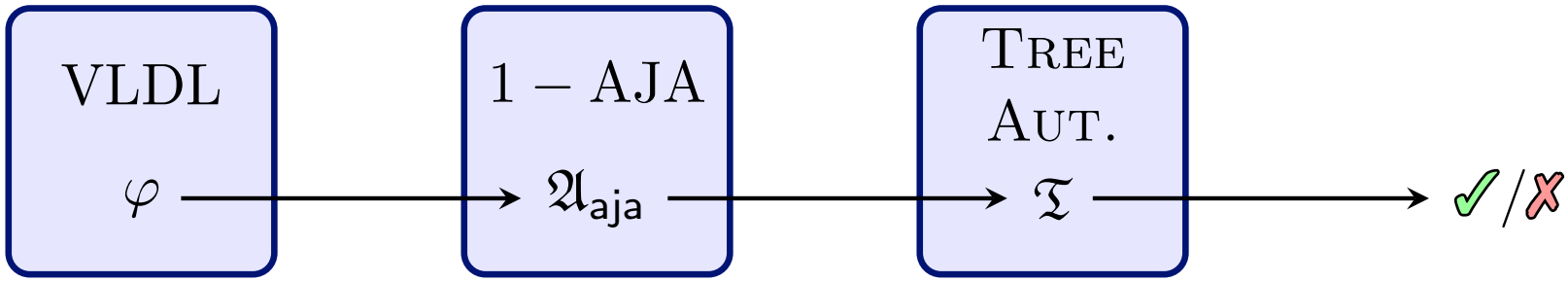
Guiding Questions

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

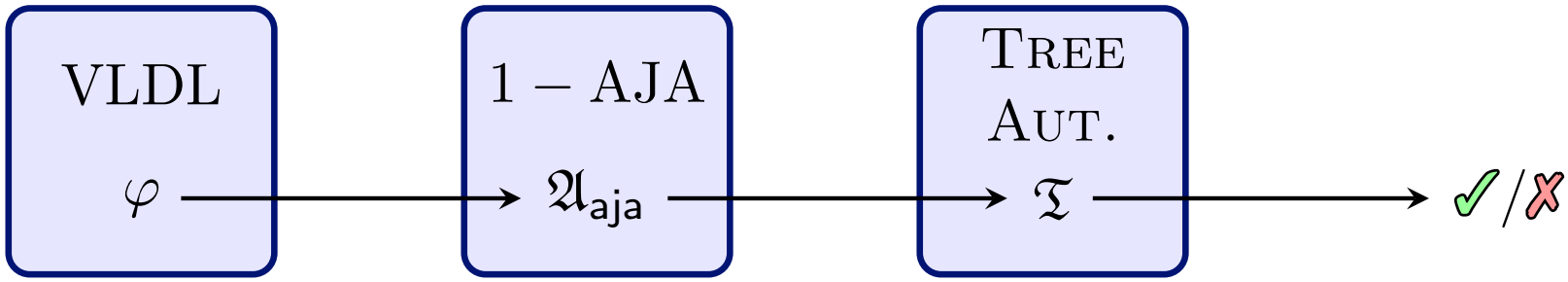
Guiding Questions

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

New Approach



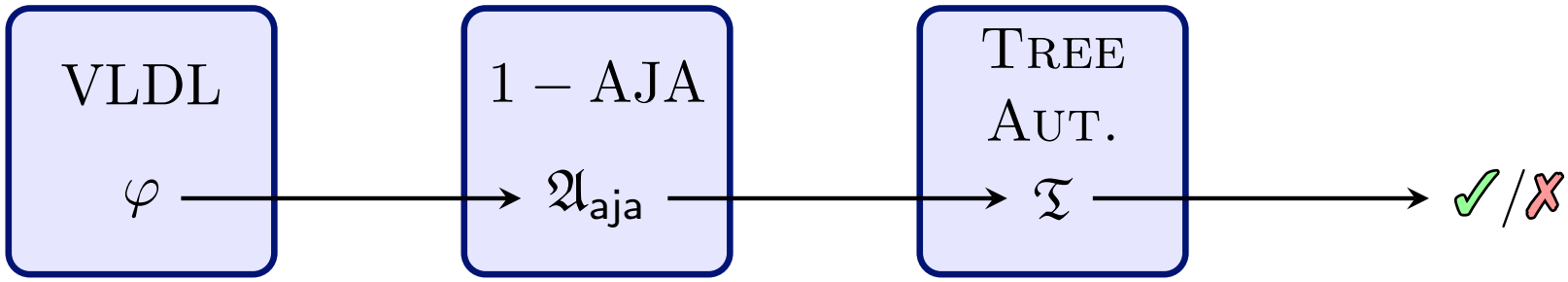
New Approach



Lemma

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

New Approach



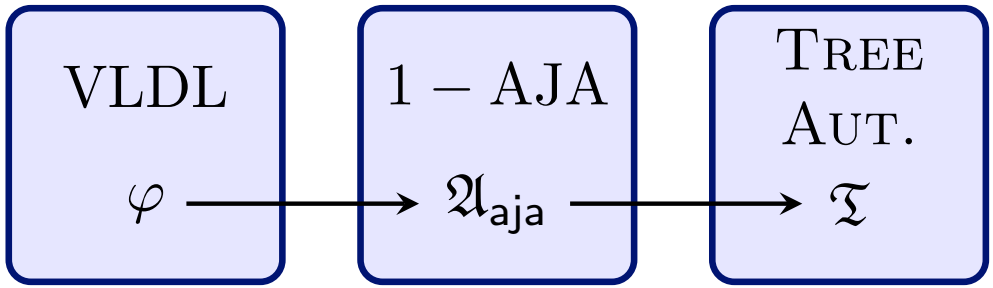
Lemma

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

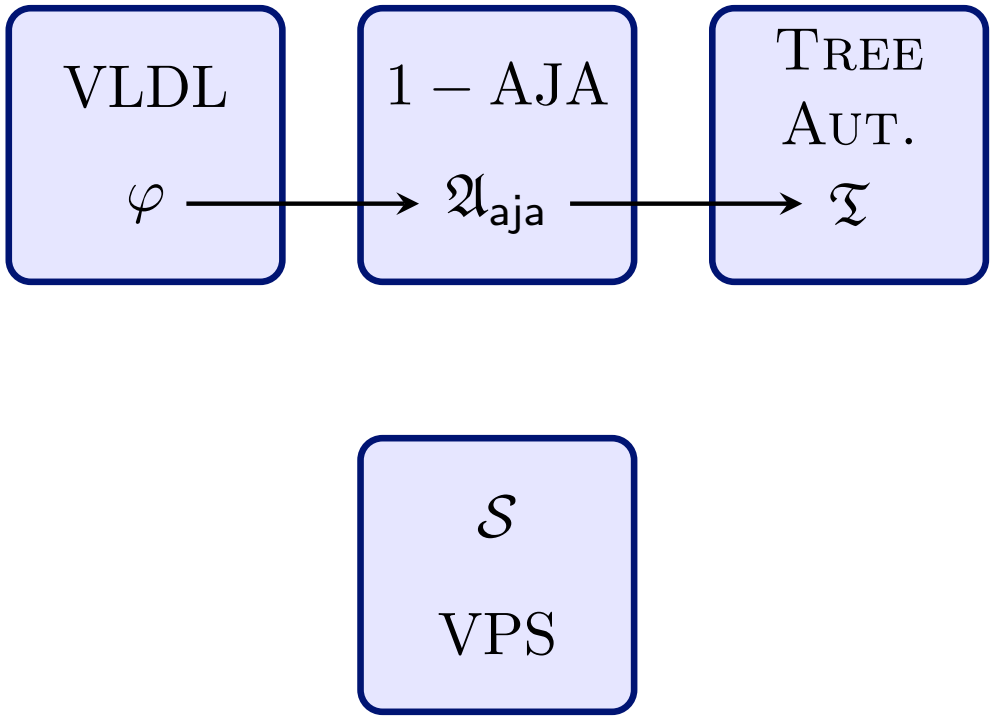
Theorem

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

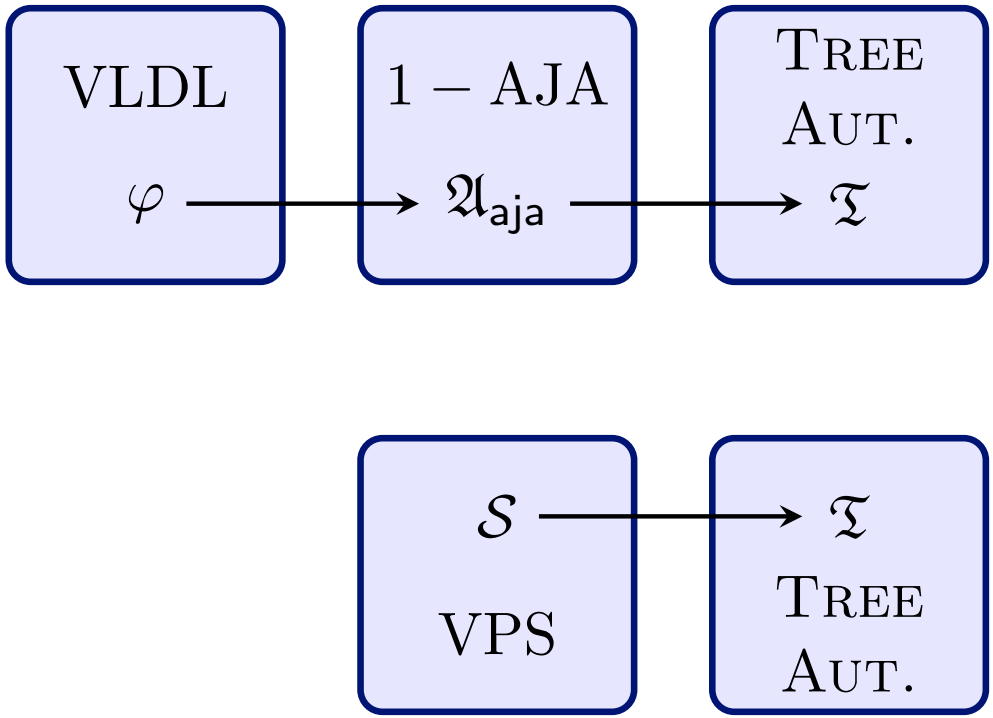
Model Checking



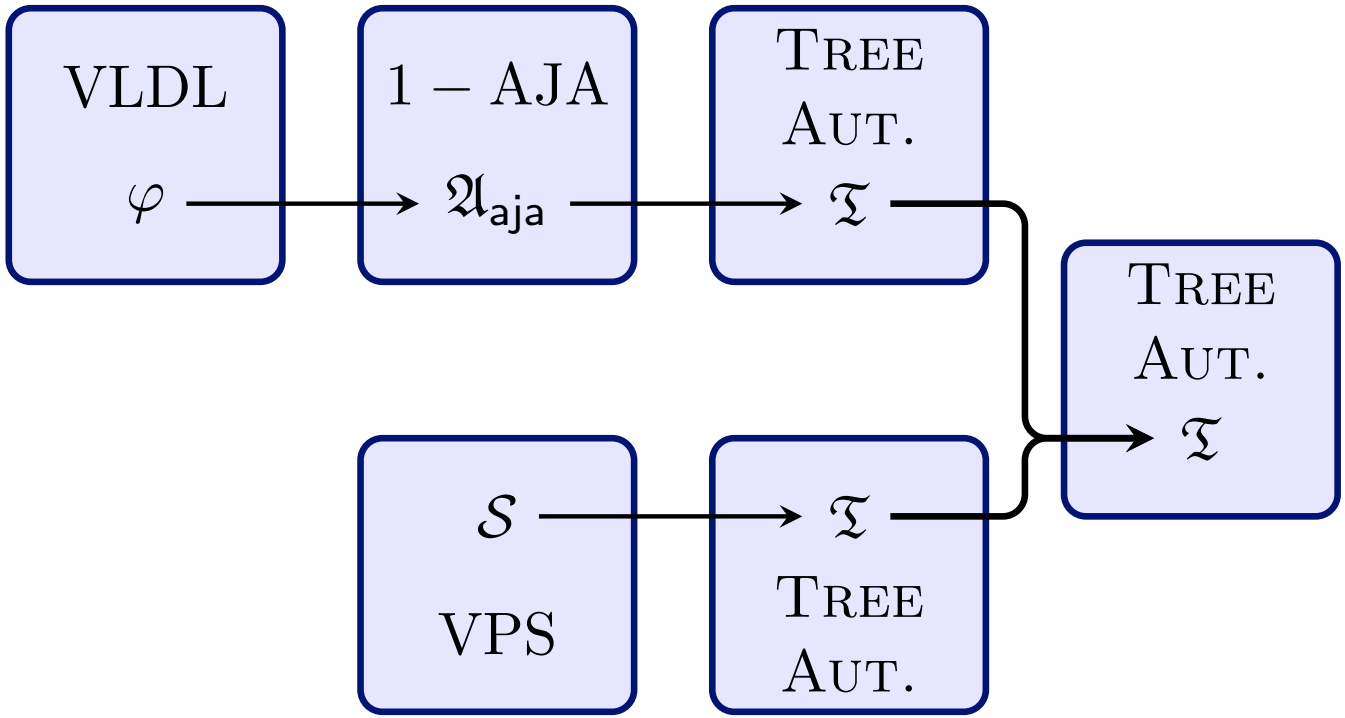
Model Checking



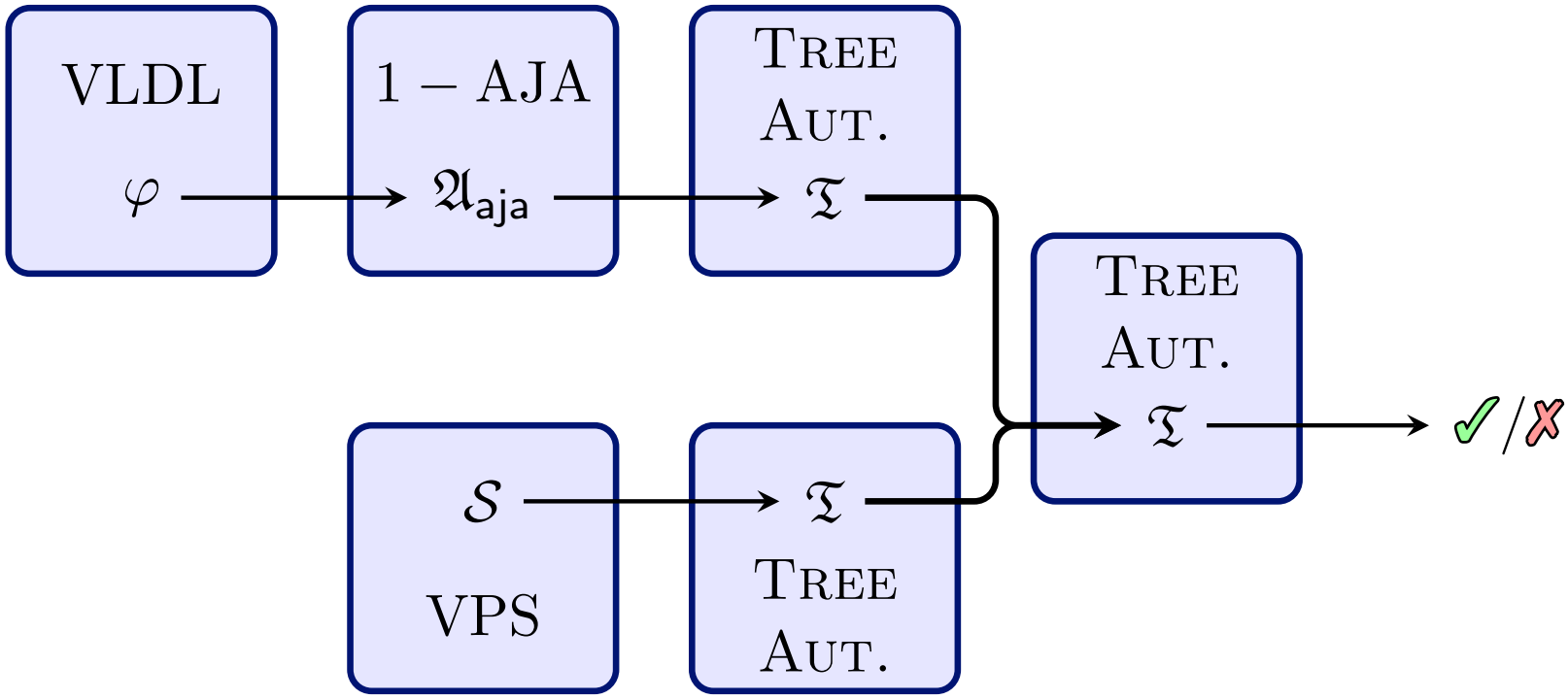
Model Checking



Model Checking



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

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

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