Input-Driven Pushdown Automata with Limited Nondeterminism

Alexander Okhotin Kai Salomaa

Turku, Finland Kingston, Canada

August 27, 2014

Outline of the talk

- Input-driven PDAs: historical background and definitions

Outline of the talk

- Input-driven PDAs: historical background and definitions
  - Other equivalent models

Outline of the talk

- Input-driven PDAs: historical background and definitions
  - Other equivalent models
- *Nondeterministic* IDPDA: Size blow-up of determinization

Outline of the talk

- Input-driven PDAs: historical background and definitions
  - Other equivalent models
- *Nondeterministic* IDPDA: Size blow-up of determinization
- Unambiguous nondeterminism
  - nondeterministic $\longrightarrow$ unambiguous $\longrightarrow$ deterministic

Outline of the talk

- Input-driven PDAs: historical background and definitions
  - Other equivalent models
- *Nondeterministic* IDPDA: Size blow-up of determinization
- Unambiguous nondeterminism
  - nondeterministic $\longrightarrow$ unambiguous $\longrightarrow$ deterministic
- *Limited nondeterminism*

Outline of the talk

- Input-driven PDAs: historical background and definitions
  - Other equivalent models
- *Nondeterministic* IDPDA: Size blow-up of determinization
- Unambiguous nondeterminism
  - nondeterministic $\longrightarrow$ unambiguous $\longrightarrow$ deterministic
- *Limited nondeterminism*
  - k-path NIDPDA and multiple entry DIDPDA

Outline of the talk

- Input-driven PDAs: historical background and definitions
  - Other equivalent models
- *Nondeterministic* IDPDA: Size blow-up of determinization
- Unambiguous nondeterminism
  - nondeterministic $\longrightarrow$ unambiguous $\longrightarrow$ deterministic
- *Limited nondeterminism*
  - k-path NIDPDA and multiple entry DIDPDA
  - Determinizing k-path NIDPDAs
    - ⋆ lower bounds for size blow-up
    - ⋆ nondeterministic $\longrightarrow$ k-path $\longrightarrow$ deterministic

Outline of the talk

- Input-driven PDAs: historical background and definitions
  - Other equivalent models
- *Nondeterministic* IDPDA: Size blow-up of determinization
- Unambiguous nondeterminism
  - nondeterministic $\longrightarrow$ unambiguous $\longrightarrow$ deterministic
- *Limited nondeterminism*
  - k-path NIDPDA and multiple entry DIDPDA
  - Determinizing k-path NIDPDAs
    - ★ lower bounds for size blow-up
    - ★ nondeterministic $\longrightarrow$ k-path $\longrightarrow$ deterministic
  - Decision problems
    - ★ Does a given NIDPDA have the k-path property?

Outline of the talk

- Input-driven PDAs: historical background and definitions
  - Other equivalent models
- *Nondeterministic* IDPDA: Size blow-up of determinization
- Unambiguous nondeterminism
  - nondeterministic $\longrightarrow$ unambiguous $\longrightarrow$ deterministic
- *Limited nondeterminism*
  - k-path NIDPDA and multiple entry DIDPDA
  - Determinizing k-path NIDPDAs
    - lower bounds for size blow-up
    - nondeterministic $\longrightarrow$ k-path $\longrightarrow$ deterministic
  - Decision problems
    - Does a given NIDPDA have the k-path property?
- Open problems and further topics

Preliminaries

- A *pushdown automaton* reads input left-to-right and has access to finite-state memory and a pushdown stack.
- Each operation either reads a symbol from the stack (pop), pushes a string to the top of the stack (push) or does not change the stack. (Additionally, a PDA may have ε-transitions.)
- *Input-driven computation:* the input symbol determines whether the machines pushes or pops the stack, or does not touch the stack.

Why IDPDA?

- every nondeterministic input-driven automaton (NIDPDA) can be determinized

Why IDPDA?

- every nondeterministic input-driven automaton (NIDPDA) can be determinized
  - a (general) nondeterministic PDA does not have an equivalent deterministic PDA

Why IDPDA?

- every nondeterministic input-driven automaton (NIDPDA) can be determinized
  - a (general) nondeterministic PDA does not have an equivalent deterministic PDA
- IDPDA retain many of the desirable closure and decision properties of finite automata
  - languages recognized by nondeterministic PDAs are not closed under intersection/complement

Why IDPDA?

- every nondeterministic input-driven automaton (NIDPDA) can be determinized
  - a (general) nondeterministic PDA does not have an equivalent deterministic PDA
- IDPDA retain many of the desirable closure and decision properties of finite automata
  - languages recognized by nondeterministic PDAs are not closed under intersection/complement
  - equivalence (inclusion) of nondeterministic (deterministic) PDAs is undecidable

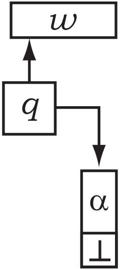
Why IDPDA?

- every nondeterministic input-driven automaton (NIDPDA) can be determinized
  - a (general) nondeterministic PDA does not have an equivalent deterministic PDA
- IDPDA retain many of the desirable closure and decision properties of finite automata
  - languages recognized by nondeterministic PDAs are not closed under intersection/complement
  - equivalence (inclusion) of nondeterministic (deterministic) PDAs is undecidable

- *Next we'll define IDPDA computations and after that will summarize basic IDPDA decision properties.*

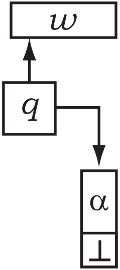
Definition: Input-driven pushdown automata

- $\Sigma = \Sigma_{+1} \cup \Sigma_0 \cup \Sigma_{-1}$: input alphabet;



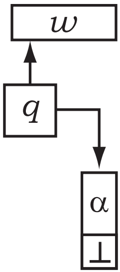
Definition: Input-driven pushdown automata

- $\Sigma = \Sigma_{+1} \cup \Sigma_0 \cup \Sigma_{-1}$: input alphabet;
- Q: finite set of states;



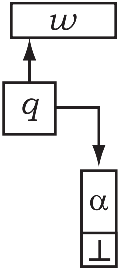
Definition: Input-driven pushdown automata

- $\Sigma = \Sigma_{+1} \cup \Sigma_0 \cup \Sigma_{-1}$: input alphabet;
- Q: finite set of states;
- $q_0 \in Q$: initial state;



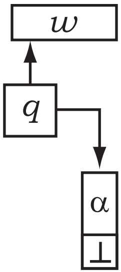
Definition: Input-driven pushdown automata

- $\Sigma = \Sigma_{+1} \cup \Sigma_0 \cup \Sigma_{-1}$: input alphabet;
- Q: finite set of states;
- $q_0 \in Q$: initial state;
- Γ: stack alphabet;



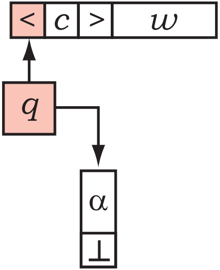
Definition: Input-driven pushdown automata

- $\Sigma = \Sigma_{+1} \cup \Sigma_0 \cup \Sigma_{-1}$: input alphabet;
- Q: finite set of states;
- $q_0 \in Q$: initial state;
- Γ: stack alphabet;
- $\perp \in \Gamma$: bottom stack symbol;



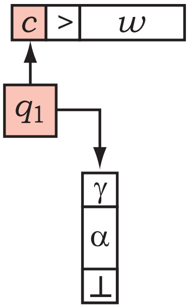
Definition: Input-driven pushdown automata

- $\Sigma = \Sigma_{+1} \cup \Sigma_0 \cup \Sigma_{-1}$: input alphabet;
- Q: finite set of states;
- $q_0 \in Q$: initial state;
- Γ: stack alphabet;
- $\perp \in \Gamma$: bottom stack symbol;
- $\delta_< : Q \to Q \times \Gamma$, for each $< \in \Sigma_{+1}$;



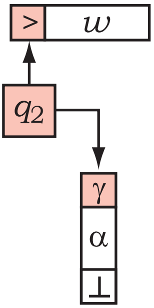
Definition: Input-driven pushdown automata

- $\Sigma = \Sigma_{+1} \cup \Sigma_0 \cup \Sigma_{-1}$: input alphabet;
- Q: finite set of states;
- $q_0 \in Q$: initial state;
- Γ: stack alphabet;
- $\perp \in \Gamma$: bottom stack symbol;
- $\delta_< : Q \to Q \times \Gamma$, for each $< \in \Sigma_{+1}$;
- $\delta_c : Q \to Q$, for each $c \in \Sigma_0$.



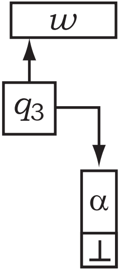
Definition: Input-driven pushdown automata

- $\Sigma = \Sigma_{+1} \cup \Sigma_0 \cup \Sigma_{-1}$: input alphabet;
- Q: finite set of states;
- $q_0 \in Q$: initial state;
- Γ: stack alphabet;
- $\perp \in \Gamma$: bottom stack symbol;
- $\delta_< : Q \to Q \times \Gamma$, for each $< \in \Sigma_{+1}$;
- $\delta_c : Q \to Q$, for each $c \in \Sigma_0$.
- $\delta_> : Q \times \Gamma \to Q$, for each $> \in \Sigma_{-1}$;
  - $\perp$ is never popped.

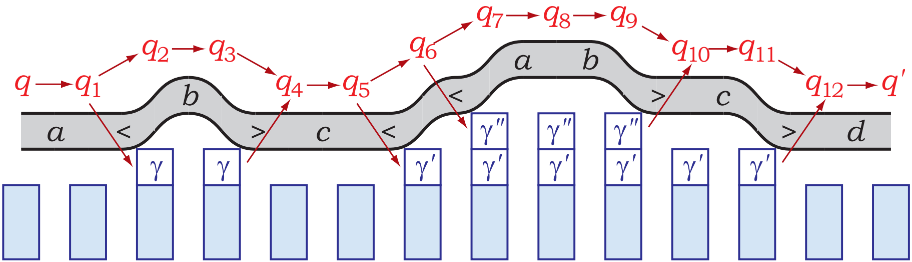


Definition: Input-driven pushdown automata

- $\Sigma = \Sigma_{+1} \cup \Sigma_0 \cup \Sigma_{-1}$: input alphabet;
- Q: finite set of states;
- $q_0 \in Q$: initial state;
- Γ: stack alphabet;
- $\perp \in \Gamma$: bottom stack symbol;
- $\delta_< : Q \to Q \times \Gamma$, for each $< \in \Sigma_{+1}$;
- $\delta_c : Q \to Q$, for each $c \in \Sigma_0$.
- $\delta_> : Q \times \Gamma \to Q$, for each $> \in \Sigma_{-1}$;
  - $\perp$ is never popped.
- $F \subseteq Q$: accepting states.



Computation of an IDPDA



Research on IDPDAs

- Languages recognized in space $\frac{\log^2 n}{\log \log n}$ and poly time on TM (Mehlhorn, ICALP 1980).

Research on IDPDAs

- Languages recognized in space $\frac{\log^2 n}{\log \log n}$ and poly time on TM (Mehlhorn, ICALP 1980).
- ...in space $\log n$ and time $n^2 \log n$ (von Braunmühl, Verbeek, 1983).

Research on IDPDAs

- Languages recognized in space $\frac{\log^2 n}{\log \log n}$ and poly time on TM (Mehlhorn, ICALP 1980).
- ...in space $\log n$ and time $n^2 \log n$ (von Braunmühl, Verbeek, 1983).
  - Alternative proof (Rytter, 1986).

Research on IDPDAs

- Languages recognized in space $\frac{\log^2 n}{\log \log n}$ and poly time on TM (Mehlhorn, ICALP 1980).
- ... in space $\log n$ and time $n^2 \log n$ (von Braunmühl, Verbeek, 1983).
  - Alternative proof (Rytter, 1986).
- ... in NC^1 (Dymond, 1988).

Research on IDPDAs

- Languages recognized in space $\frac{\log^2 n}{\log \log n}$ and poly time on TM (Mehlhorn, ICALP 1980).
- ...in space $\log n$ and time $n^2 \log n$ (von Braunmühl, Verbeek, 1983).
  - Alternative proof (Rytter, 1986).
- ...in NC^1 (Dymond, 1988).
- Language-theoretic study (Alur, Madhusudan, STOC 2004).
  - Rediscovered as "visibly pushdown automata".

Research on IDPDAs

- Languages recognized in space $\frac{\log^2 n}{\log \log n}$ and poly time on TM (Mehlhorn, ICALP 1980).
- ...in space $\log n$ and time $n^2 \log n$ (von Braunmühl, Verbeek, 1983).
  - Alternative proof (Rytter, 1986).
- ...in NC^1 (Dymond, 1988).
- Language-theoretic study (Alur, Madhusudan, STOC 2004).
  - Rediscovered as "visibly pushdown automata".
  - Reinterpreted as "nested word automata" (Alur, Madhusudan, DLT 2006).

Research on IDPDAs

- Languages recognized in space $\frac{\log^2 n}{\log \log n}$ and poly time on TM (Mehlhorn, ICALP 1980).
- ...in space $\log n$ and time $n^2 \log n$ (von Braunmühl, Verbeek, 1983).
  - Alternative proof (Rytter, 1986).
- ...in NC^1 (Dymond, 1988).
- Language-theoretic study (Alur, Madhusudan, STOC 2004).
  - Rediscovered as "visibly pushdown automata".
  - Reinterpreted as "nested word automata" (Alur, Madhusudan, DLT 2006).
  - *Descriptional complexity*

Research on IDPDAs

- Languages recognized in space $\frac{\log^2 n}{\log \log n}$ and poly time on TM (Mehlhorn, ICALP 1980).
- ... in space $\log n$ and time $n^2 \log n$ (von Braunmühl, Verbeek, 1983).
  - Alternative proof (Rytter, 1986).
- ... in NC^1 (Dymond, 1988).
- Language-theoretic study (Alur, Madhusudan, STOC 2004).
  - Rediscovered as "visibly pushdown automata".
  - Reinterpreted as "nested word automata" (Alur, Madhusudan, DLT 2006).
  - *Descriptional complexity*
    - ★ Lower bound on size blow-up of determinization

Research on IDPDAs

- Languages recognized in space $\frac{\log^2 n}{\log \log n}$ and poly time on TM (Mehlhorn, ICALP 1980).
- ... in space $\log n$ and time $n^2 \log n$ (von Braunmühl, Verbeek, 1983).
  - Alternative proof (Rytter, 1986).
- ... in NC^1 (Dymond, 1988).
- Language-theoretic study (Alur, Madhusudan, STOC 2004).
  - Rediscovered as "visibly pushdown automata".
  - Reinterpreted as "nested word automata" (Alur, Madhusudan, DLT 2006).
  - *Descriptional complexity*
    - ★ Lower bound on size blow-up of determinization
  - Closure under most standard language operations.

Research on IDPDAs

- Languages recognized in space $\frac{\log^2 n}{\log \log n}$ and poly time on TM (Mehlhorn, ICALP 1980).
- ...in space $\log n$ and time $n^2 \log n$ (von Braunmühl, Verbeek, 1983).
  - Alternative proof (Rytter, 1986).
- ...in NC^1 (Dymond, 1988).
- Language-theoretic study (Alur, Madhusudan, STOC 2004).
  - Rediscovered as "visibly pushdown automata".
  - Reinterpreted as "nested word automata" (Alur, Madhusudan, DLT 2006).
  - *Descriptional complexity*
    - ★ Lower bound on size blow-up of determinization
  - Closure under most standard language operations.
- Much ongoing research – motivated by new applications that use data with linear/hierarchical structure

Closure properties of central language families

	$\cup$	$\cap$	Complement	Concatenation	Kleene-*
Regular	Yes	Yes	Yes	Yes	Yes
CFL	Yes	No	No	Yes	Yes
DCFL	No	No	Yes	No	No
IDPDA	Yes	Yes	Yes	Yes	Yes

(D)CFL = (deterministic) context-free languages

Summary of decision properties

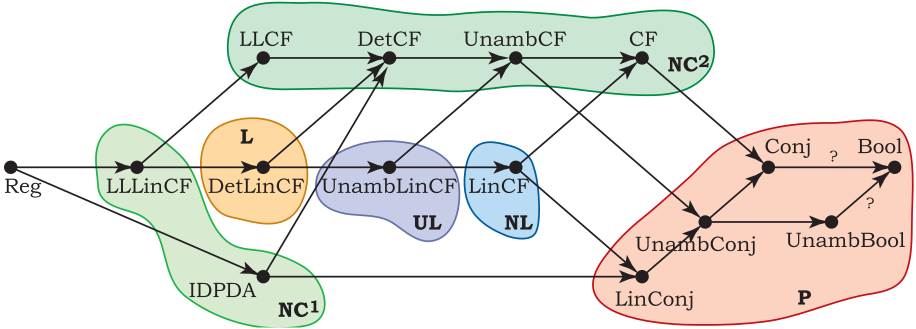
	Membership		Properties of a language		
	fixed	uniform	emptiness	equality	inclusion
DFA	regular	L	NL	NL	NL
NFA	regular	NL	NL	PSPACE	PSPACE
DIDPDA	in NC^1	in SC^2	P	P	P
NIDPDA	in NC^1	in P	P	EXPTIME	EXPTIME
DPDA	in $\mathrm{NC}^2 \cap \mathrm{SC}^2$	P	P	decidable	co-r.e.
CF	in NC^2	P	P	co-r.e.	co-r.e.

Summary of decision properties

	Membership		Properties of a language		
	fixed	uniform	emptiness	equality	inclusion
DFA	regular	L	NL	NL	NL
NFA	regular	NL	NL	PSPACE	PSPACE
DIDPDA	in NC^1	in SC^2	P	P	P
NIDPDA	in NC^1	in P	P	EXPTIME	EXPTIME
DPDA	in $\mathrm{NC}^2 \cap \mathrm{SC}^2$	P	P	decidable	co-r.e.
CF	in NC^2	P	P	co-r.e.	co-r.e.

- Recent comprehensive survey:
  A. Okhotin, K. Salomaa, *Complexity of Input-Driven Pushdown Automata*, SIGACT News Complexity Theory Column 82 (Lane A. Hemaspaandra, Ed.), vol. 45, no. 2, June 2014, pp. 46–67

The big picture: IDPDAs among formal grammars



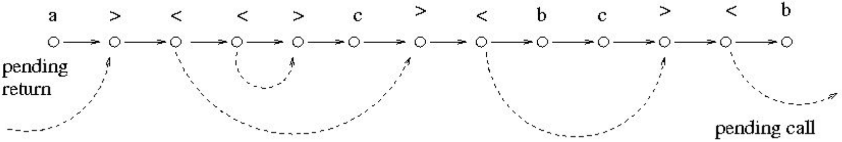
Equivalent models: pushdown forest automata

- *Pushdown forest automaton* (Neumann & Seidl 1998)
  - traverses input tree in depth-first left-to-right order
  - machine pushes onto the stack when going down to the leftmost child
  - pops from the stack when returning from the rightmost child
- Equivalent to IDPDA (Gauwin, Niehren & Roos, 2008)
- Recognize only the class of regular tree languages. Are *exponentially more succinct* than ordinary bottom-up tree automata
- Earlier related work:
  - Engelfriet, Rozenberg & Slutzki (1980): tree-walking transducers with synchronized pushdown
  - Kamimura & Slutzki (1981): nondeterministic and deterministic variants of such graph walking automata with a synchronized pushdown are equivalent

Equivalent models: Nested word automata

(Alur and Madhusudan, DLT 2006)

- A *nested word* is a tagged word with a hierarchical structure that connects call symbol occurrences to return symbol occurrences



- A *nested word automaton* “sends” finite state information both along the linear and the hierarchical edges
  - Equivalent to an IDPDA

Why nested words?

- Nested word automata used e.g. in XML document processing and model checking
  - Retains many desirable properties of the classical regular languages
- Advantages over trees in applications like document processing:
  - Word operations like prefix, suffix and concatenation do not have clear analogoues as tree operations
  - Trees do not have an *explicit* linear ordering of all nodes
    - Descriptional complexity: for tree automata queries that refer to the global linear order can be more expensive

In the following we use the terminology associated with IDPDAs. The model is equivalent to a finite automaton operating on nested words.

Determinizing nondeterministic IDPDAs (NIDPDA)

Theorem (von Braunmühl & Verbeek (1983), Alur & Madhusudan (2006))

Every NIDPDA of size n has a deterministic IDPDA of size $2^{O(n^2)}$.

Determinizing nondeterministic IDPDAs (NIDPDA)

Theorem (von Braunmühl & Verbeek (1983), Alur & Madhusudan (2006))

Every NIDPDA of size n has a deterministic IDPDA of size $2^{O(n^2)}$.

Theorem (Alur, Madhusudan, 2006)

A DIDPDA equivalent to an NIDPDA of size n needs, in the worst case, $2^{\Omega(n^2)}$ *states.*

Determinizing nondeterministic IDPDAs (NIDPDA)

Theorem (von Braunmühl & Verbeek (1983), Alur & Madhusudan (2006))

Every NIDPDA of size n has a deterministic IDPDA of size $2^{O(n^2)}$.

Theorem (Alur, Madhusudan, 2006)

A DIDPDA equivalent to an NIDPDA of size n needs, in the worst case, $2^{\Omega(n^2)}$ *states.*

- The more precise constants in the lower bound remain open.
  - With a *constant alphabet* can reach lower bound $2^{\frac{1}{9}n^2}$

Determinizing nondeterministic IDPDAs (NIDPDA)

Theorem (von Braunmühl & Verbeek (1983), Alur & Madhusudan (2006))

Every NIDPDA of size n has a deterministic IDPDA of size $2^{O(n^2)}$.

Theorem (Alur, Madhusudan, 2006)

A DIDPDA equivalent to an NIDPDA of size n needs, in the worst case, $2^{\Omega(n^2)}$ *states.*

- The more precise constants in the lower bound remain open.
  - With a *constant alphabet* can reach lower bound $2^{\frac{1}{9}n^2}$
  - *Linear size alphabet* can reach $2^{\frac{1}{4}n^2}$

Determinizing nondeterministic IDPDAs (NIDPDA)

Theorem (von Braunmühl & Verbeek (1983), Alur & Madhusudan (2006))

Every NIDPDA of size n has a deterministic IDPDA of size $2^{O(n^2)}$.

Theorem (Alur, Madhusudan, 2006)

A DIDPDA equivalent to an NIDPDA of size n needs, in the worst case, $2^{\Omega(n^2)}$ *states.*

- The more precise constants in the lower bound remain open.
  - With a *constant alphabet* can reach lower bound $2^{\frac{1}{9}n^2}$
  - *Linear size alphabet* can reach $2^{\frac{1}{4}n^2}$
  - *Exponential size alphabet* can reach 2^{n^2}

Determinizing NIDPDAs: the number of stack symbols

- The determinization construction does not depend on the number of stack symbols
  - NIDPDA with n states $\rightarrow$ DIDPDA with 2^{n^2} states and $O(2^{n^2})$ stack symbols

Determinizing NIDPDAs: the number of stack symbols

- The determinization construction does not depend on the number of stack symbols
  - NIDPDA with n states $\rightarrow$ DIDPDA with 2^{n^2} states and $O(2^{n^2})$ stack symbols
- In the lower bound construction, the equivalent DIDPDA does not use the stack, and could be replaced by a DFA of size $2^{\Omega(n^2)}$

Determinizing NIDPDAs: the number of stack symbols

- The determinization construction does not depend on the number of stack symbols
  - NIDPDA with n states $\rightarrow$ DIDPDA with 2^{n^2} states and $O(2^{n^2})$ stack symbols
- In the lower bound construction, the equivalent DIDPDA does not use the stack, and could be replaced by a DFA of size $2^{\Omega(n^2)}$

✓ More refined lower bound for determinizing an NIDPDA.

Determinizing NIDPDAs: the number of stack symbols

- The determinization construction does not depend on the number of stack symbols
  - NIDPDA with n states $\rightarrow$ DIDPDA with 2^{n^2} states and $O(2^{n^2})$ stack symbols
- In the lower bound construction, the equivalent DIDPDA does not use the stack, and could be replaced by a DFA of size $2^{\Omega(n^2)}$

✓ More refined lower bound for determinizing an NIDPDA.

Theorem (Okhotin, Piao & Salomaa, 2012)

For all $k, h \in \mathbb{N}$, $k \leq h$, there exists a language $L_{k,h}$ recognized by an NIDPDA with $O(h)$ states and $O(k)$ stack symbols such that any DIDPDA for $L_{k,h}$ needs $\Omega(2^{k \cdot h})$ states and $\Omega(2^{k^2})$ stack symbols.

Determinizing NIDPDAs: the number of stack symbols

- The determinization construction does not depend on the number of stack symbols
  - NIDPDA with n states $\to$ DIDPDA with 2^{n^2} states and $O(2^{n^2})$ stack symbols
- In the lower bound construction, the equivalent DIDPDA does not use the stack, and could be replaced by a DFA of size $2^{\Omega(n^2)}$

✓ More refined lower bound for determinizing an NIDPDA.

Theorem (Okhotin, Piao & Salomaa, 2012)

For all $k, h \in \mathbb{N}$, $k \leq h$, there exists a language $L_{k,h}$ recognized by an NIDPDA with $O(h)$ states and $O(k)$ stack symbols such that any DIDPDA for $L_{k,h}$ needs $\Omega(2^{k \cdot h})$ states and $\Omega(2^{k^2})$ stack symbols.

- Tight bound with respect to both the number of states and the number of stack symbols (within a constant factor)

The lower bound due to Alur and Madhusudan

Lower bound: Let $\Sigma_{+1} = \{<\}$, $\Sigma_0 = \{0, 1, \#\}$, $\Sigma_{-1} = \{>\}$, consider all

$$< \ldots uv \ldots v > u$$

with $u, v \in \{0, 1\}^{\log n}$. (markers $\#$ omitted here and later)

The lower bound due to Alur and Madhusudan

Lower bound: Let $\Sigma_{+1} = \{<\}$, $\Sigma_0 = \{0, 1, \#\}$, $\Sigma_{-1} = \{>\}$, consider all

$$< \ldots uv \ldots v > u$$

with $u, v \in \{0, 1\}^{\log n}$. (markers # omitted here and later)

- $O(n)$-state NIDPDA guesses u and pushes it.

The lower bound due to Alur and Madhusudan

Lower bound: Let $\Sigma_{+1} = \{<\}$, $\Sigma_0 = \{0, 1, \#\}$, $\Sigma_{-1} = \{>\}$, consider all

$$< \dots uv \dots v > u$$

with $u, v \in \{0, 1\}^{\log n}$. (markers # omitted here and later)

- $O(n)$-state NIDPDA guesses u and pushes it.
- IDPDA has to remember all pairs (u, v).

The lower bound due to Alur and Madhusudan

Lower bound: Let $\Sigma_{+1} = \{<\}$, $\Sigma_0 = \{0, 1, \#\}$, $\Sigma_{-1} = \{>\}$, consider all

$$< \ldots uv \ldots v > u$$

with $u, v \in \{0, 1\}^{\log n}$. (markers # omitted here and later)

- $O(n)$-state NIDPDA guesses u and pushes it.
- IDPDA has to remember all pairs (u, v).

Upper bound: Remember sets of pairs (q, q') on each level of brackets:

$$\ldots < \underbrace{\ldots}_{q \to q'} > \ldots$$

Unambiguous Nondeterminism

Commonly used form of limited nondeterminism

- "If a string is accepted, it has a unique accepting computation".

Unambiguous Nondeterminism

Commonly used form of limited nondeterminism

- "If a string is accepted, it has a unique accepting computation".
- In complexity theory:

$$L \subseteq UL \subseteq NL \qquad P \subseteq UP \subseteq NP$$

Unambiguous Nondeterminism

Commonly used form of limited nondeterminism

- "If a string is accepted, it has a unique accepting computation".
- In complexity theory:

$$L \subseteq UL \subseteq NL \qquad P \subseteq UP \subseteq NP$$

- In finite automata: DFA $\rightarrow$ UFA $\rightarrow$ NFA.

Unambiguous Nondeterminism

Commonly used form of limited nondeterminism

- "If a string is accepted, it has a unique accepting computation".
- In complexity theory:

$$L \subseteq UL \subseteq NL \qquad P \subseteq UP \subseteq NP$$

- In finite automata: DFA $\rightarrow$ UFA $\rightarrow$ NFA.
  - NFA–DFA tradeoff: 2^n (Lupanov, 1963).

Unambiguous Nondeterminism

Commonly used form of limited nondeterminism

- "If a string is accepted, it has a unique accepting computation".
- In complexity theory:

$$L \subseteq UL \subseteq NL \qquad P \subseteq UP \subseteq NP$$

- In finite automata: DFA $\rightarrow$ UFA $\rightarrow$ NFA.
  - NFA–DFA tradeoff: 2^n (Lupanov, 1963).
  - UFA–DFA and NFA–UFA tradeoffs: 2^n and $2^n - 1$ (Hing Leung, 2005).

Unambiguous Nondeterminism

Commonly used form of limited nondeterminism

- "If a string is accepted, it has a unique accepting computation".
- In complexity theory:

$$L \subseteq UL \subseteq NL \qquad P \subseteq UP \subseteq NP$$

- In finite automata: DFA $\to$ UFA $\to$ NFA.
  - NFA–DFA tradeoff: 2^n (Lupanov, 1963).
  - UFA–DFA and NFA–UFA tradeoffs: 2^n and $2^n - 1$ (Hing Leung, 2005).
  - UFA–DFA and NFA–UFA tradeoffs for unary alphabet: $e^{\Theta(\sqrt[3]{n \ln^2 n})}$ and $e^{(1+o(1))\sqrt{n \ln n}}$ (Okhotin, 2010).

Unambiguous Nondeterminism

Commonly used form of limited nondeterminism

- "If a string is accepted, it has a unique accepting computation".
- In complexity theory:

$$L \subseteq UL \subseteq NL \qquad P \subseteq UP \subseteq NP$$

- In finite automata: DFA $\rightarrow$ UFA $\rightarrow$ NFA.
  - NFA–DFA tradeoff: 2^n (Lupanov, 1963).
  - UFA–DFA and NFA–UFA tradeoffs: 2^n and $2^n - 1$ (Hing Leung, 2005).
  - UFA–DFA and NFA–UFA tradeoffs for unary alphabet: $e^{\Theta(\sqrt[3]{n \ln^2 n})}$ and $e^{(1+o(1))\sqrt{n \ln n}}$ (Okhotin, 2010).
- What about unambiguous IDPDAs (UIDPDA)?

Unambiguous Nondeterminism

Commonly used form of limited nondeterminism

- "If a string is accepted, it has a unique accepting computation".
- In complexity theory:

$$L \subseteq UL \subseteq NL \qquad P \subseteq UP \subseteq NP$$

- In finite automata: DFA → UFA → NFA.
  - NFA–DFA tradeoff: 2^n (Lupanov, 1963).
  - UFA–DFA and NFA–UFA tradeoffs: 2^n and $2^n - 1$ (Hing Leung, 2005).
  - UFA–DFA and NFA–UFA tradeoffs for unary alphabet: $e^{\Theta(\sqrt[3]{n \ln^2 n})}$ and $e^{(1+o(1))\sqrt{n \ln n}}$ (Okhotin, 2010).
- What about unambiguous IDPDAs (UIDPDA)?

Theorem (Okhotin, Salomaa 2011)

The worst-case UIDPDA–DIDPDA and NIDPDA–UIDPDA trade-offs are both $2^{\Theta(n^2)}$.

UIDPDA to DIDPDA: $2^{\Theta(n^2)}$

- Upper bound: known for NIDPDA $\to$ DIDPDA.

UIDPDA to DIDPDA: $2^{\Theta(n^2)}$

- Upper bound: known for NIDPDA $\to$ DIDPDA.
- Let $\Sigma_{+1} = \{<\}$, $\Sigma_0 = \{0, 1, \#, \$\}$, $\Sigma_{-1} = \{>\}$, consider all

$$<x_0 \ldots x_\ell \$ v > u$$

with $u, v \in \{0, 1\}^{\log n}$, where the $(v)_2$-th bit in $x_{(u)_2}$ is 1.

UIDPDA to DIDPDA: $2^{\Theta(n^2)}$

- Upper bound: known for NIDPDA $\to$ DIDPDA.
- Let $\Sigma_{+1} = \{<\}$, $\Sigma_0 = \{0, 1, \#, \$\}$, $\Sigma_{-1} = \{>\}$, consider all

$$<x_0 \ldots x_\ell \$ v > u$$

  with $u, v \in \{0, 1\}^{\log n}$, where the $(v)_2$-th bit in $x_{(u)_2}$ is 1.
- $O(n)$-state UIDPDA guesses u.

UIDPDA to DIDPDA: $2^{\Theta(n^2)}$

- Upper bound: known for NIDPDA $\to$ DIDPDA.
- Let $\Sigma_{+1} = \{<\}$, $\Sigma_0 = \{0, 1, \#, \$\}$, $\Sigma_{-1} = \{>\}$, consider all

$$<x_0 \dots x_\ell \$ v > u$$

  with $u, v \in \{0,1\}^{\log n}$, where the $(v)_2$-th bit in $x_{(u)_2}$ is 1.
- $O(n)$-state UIDPDA guesses u.
- IDPDA has to remember $x_0, \dots, x_{n-1}$.

NIDPDA to an unambiguous IDPDA

Lower bound method for unambiguous IDPDAs

Lemma (Schmidt, 1978)

Let $L \subseteq \Sigma^*$ *and* $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ *with* $x_i, y_i \in \Sigma^*$.
Define $M \in \mathbb{Z}^{n \times n}$ *by* $M_{i,j} = 1$ *if* $x_i y_j \in L$, *and* $M_{i,j} = 0$ *otherwise.*
Then every UFA for L *has*

$$|Q| \geqslant \operatorname{rank} M.$$

NIDPDA to an unambiguous IDPDA

Lower bound method for unambiguous IDPDAs

Lemma (Schmidt, 1978)

Let $L \subseteq \Sigma^*$ *and* $\{(x_1, y_1), \dots, (x_n, y_n)\}$ *with* $x_i, y_i \in \Sigma^*$.
Define $M \in \mathbb{Z}^{n \times n}$ *by* $M_{i,j} = 1$ *if* $x_i y_j \in L$, *and* $M_{i,j} = 0$ *otherwise.*
Then every UFA for L *has*

$$|Q| \geqslant \operatorname{rank} M.$$

Lemma

Further assume that $|x_1|_{\Sigma_{+1}} = \dots = |x_n|_{\Sigma_{+1}} = k$.
Then every UIDPDA for L *has*

$$|Q| \cdot |\Gamma|^k \geqslant \operatorname{rank} M.$$

NIDPDA to an unambiguous IDPDA

Lower bound method for unambiguous IDPDAs

Lemma (Schmidt, 1978)

Let $L \subseteq \Sigma^*$ *and* $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ *with* $x_i, y_i \in \Sigma^*$.
Define $M \in \mathbb{Z}^{n \times n}$ *by* $M_{i,j} = 1$ *if* $x_i y_j \in L$, *and* $M_{i,j} = 0$ *otherwise.*
Then every UFA for L *has*

$$|Q| \geqslant \operatorname{rank} M.$$

Lemma

Further assume that $|x_1|_{\Sigma_{+1}} = \ldots = |x_n|_{\Sigma_{+1}} = k$.
Then every UIDPDA for L *has*

$$|Q| \cdot |\Gamma|^k \geqslant \operatorname{rank} M.$$

- To compute the rank, need a very simple matrix.

NIDPDA to UIDPDA: $2^{\Theta(n^2)}$

- Upper bound: known for NIDPDA $\to$ IDPDA.

NIDPDA to UIDPDA: $2^{\Theta(n^2)}$

- Upper bound: known for NIDPDA $\to$ IDPDA.
- Let $\Sigma_{+1} = \{<\}$, $\Sigma_0 = \{0, 1, \#, \$\}$, $\Sigma_{-1} = \{>\}$, consider

$$< \ldots uv \ldots \$ \ldots vu \ldots >$$

(n different us and vs)

NIDPDA to UIDPDA: $2^{\Theta(n^2)}$

- Upper bound: known for NIDPDA $\to$ IDPDA.
- Let $\Sigma_{+1} = \{<\}$, $\Sigma_0 = \{0, 1, \#, \$\}$, $\Sigma_{-1} = \{>\}$, consider

$$< \ldots uv \ldots \$ \ldots vu \ldots >$$

  (n different us and vs)
- $O(n)$-state NIDPDA guesses u.

NIDPDA to UIDPDA: $2^{\Theta(n^2)}$

- Upper bound: known for NIDPDA $\to$ IDPDA.
- Let $\Sigma_{+1} = \{<\}$, $\Sigma_0 = \{0, 1, \#, \$\}$, $\Sigma_{-1} = \{>\}$, consider

$$< \ldots uv \ldots \$ \ldots vu \ldots >$$

  (n different us and vs)
- $O(n)$-state NIDPDA guesses u.
- Every UIDPDA requires $2^{\frac{n^2}{2}}$ states.

NIDPDA to UIDPDA: $2^{\Theta(n^2)}$

- Upper bound: known for NIDPDA $\to$ IDPDA.
- Let $\Sigma_{+1} = \{<\}$, $\Sigma_0 = \{0, 1, \#, \$\}$, $\Sigma_{-1} = \{>\}$, consider

$$< \ldots uv \ldots \$ \ldots vu \ldots >$$

  (n different us and vs)
- $O(n)$-state NIDPDA guesses u.
- Every UIDPDA requires $2^{\frac{n^2}{2}}$ states.
  - Arrange n^2 pairs (u, v) into $\frac{n^2}{2}$ pairs.

$$\begin{array}{ccc} (u_1, v_1) & \longleftrightarrow & (u'_1, v'_1) \\ & \vdots & \\ (u_{\frac{n^2}{2}}, v_{\frac{n^2}{2}}) & \longleftrightarrow & (u'_{\frac{n^2}{2}}, v'_{\frac{n^2}{2}}) \end{array}$$

NIDPDA to UIDPDA: $2^{\Theta(n^2)}$

- Upper bound: known for NIDPDA $\to$ IDPDA.
- Let $\Sigma_{+1} = \{<\}$, $\Sigma_0 = \{0, 1, \#, \$\}$, $\Sigma_{-1} = \{>\}$, consider

$$< \ldots uv \ldots \$ \ldots vu \ldots >$$

  (n different us and vs)
- $O(n)$-state NIDPDA guesses u.
- Every UIDPDA requires $2^{\frac{n^2}{2}}$ states.
  - Arrange n^2 pairs (u, v) into $\frac{n^2}{2}$ pairs.

$$\begin{array}{ccc} (u_1, v_1) & \longleftrightarrow & (u'_1, v'_1) \\ & \vdots & \\ (u_{\frac{n^2}{2}}, v_{\frac{n^2}{2}}) & \longleftrightarrow & (u'_{\frac{n^2}{2}}, v'_{\frac{n^2}{2}}) \end{array}$$

  - Choose one from each line:

$$x_i = < \boxed{\text{all } (u, v) \text{ chosen}} \, \$ \qquad y_i = \boxed{\text{all } (u, v) \text{ not chosen}} >$$

NIDPDA to UIDPDA: $2^{\Theta(n^2)}$

- Upper bound: known for NIDPDA $\to$ IDPDA.
- Let $\Sigma_{+1} = \{<\}$, $\Sigma_0 = \{0, 1, \#, \$\}$, $\Sigma_{-1} = \{>\}$, consider

$$< \ldots uv \ldots \$ \ldots vu \ldots >$$

  (n different us and vs)
- $O(n)$-state NIDPDA guesses u.
- Every UIDPDA requires $2^{\frac{n^2}{2}}$ states.
  - Arrange n^2 pairs (u, v) into $\frac{n^2}{2}$ pairs.

$$\begin{array}{ccc} (u_1, v_1) & \longleftrightarrow & (u'_1, v'_1) \\ & \vdots & \\ (u_{\frac{n^2}{2}}, v_{\frac{n^2}{2}}) & \longleftrightarrow & (u'_{\frac{n^2}{2}}, v'_{\frac{n^2}{2}}) \end{array}$$

  - Choose one from each line:

$$x_i = < \boxed{\text{all } (u, v) \text{ chosen}} \ \$ \qquad y_i = \boxed{\text{all } (u, v) \text{ not chosen}} >$$

  - The matrix has full rank.

Limited Nondeterminism

Finite path IDPDA

- UIDPDAs: unique accepting computation, otherwise *unlimited* nondeterminism.

Limited Nondeterminism

Finite path IDPDA

- UIDPDAs: unique accepting computation, otherwise *unlimited* nondeterminism.
- Limit total amount of nondeterminism in NIDPDAs?
  - Analogous measures earlier considered for NFAs

Limited Nondeterminism

Finite path IDPDA

- UIDPDAs: unique accepting computation, otherwise *unlimited* nondeterminism.
- Limit total amount of nondeterminism in NIDPDAs?
  - Analogous measures earlier considered for NFAs
- a *k-path* NIDPDA has at most k-branches in any computation tree

Limited Nondeterminism

Finite path IDPDA

- UIDPDAs: unique accepting computation, otherwise *unlimited* nondeterminism.
- Limit total amount of nondeterminism in NIDPDAs?
  - Analogous measures earlier considered for NFAs
- a *k-path* NIDPDA has at most k-branches in any computation tree
  - an NIDPDA has *finite path property* if it is k-path for some $k \geq 1$

Limited Nondeterminism

Finite path IDPDA

- UIDPDAs: unique accepting computation, otherwise *unlimited* nondeterminism.
- Limit total amount of nondeterminism in NIDPDAs?
  - Analogous measures earlier considered for NFAs
- a *k-path* NIDPDA has at most k-branches in any computation tree
  - an NIDPDA has *finite path property* if it is k-path for some $k \geq 1$
- *multiple-entry* IDPDA: a DIDPDA with $k \geq 1$ initial states

Limited Nondeterminism

Finite path IDPDA

- UIDPDAs: unique accepting computation, otherwise *unlimited* nondeterminism.
- Limit total amount of nondeterminism in NIDPDAs?
  - Analogous measures earlier considered for NFAs
- a *k-path* NIDPDA has at most k-branches in any computation tree
  - an NIDPDA has *finite path property* if it is k-path for some $k \geq 1$
- *multiple-entry* IDPDA: a DIDPDA with $k \geq 1$ initial states
  - k-entry DIDPDA is a (very restricted) k-path NIDPDA

Limited Nondeterminism

Finite path IDPDA

- UIDPDAs: unique accepting computation, otherwise *unlimited* nondeterminism.
- Limit total amount of nondeterminism in NIDPDAs?
  - Analogous measures earlier considered for NFAs
- a *k-path* NIDPDA has at most k-branches in any computation tree
  - an NIDPDA has *finite path property* if it is k-path for some $k \geq 1$
- *multiple-entry* IDPDA: a DIDPDA with $k \geq 1$ initial states
  - k-entry DIDPDA is a (very restricted) k-path NIDPDA
- *Formal definitions in the proceedings.*

Determinizing k-entry DIDPDAs

Lemma

A k-entry DIDPDA A with n states and m stack symbols simulated by a DIDPDA B with $(n+1)^k - 1$ states and m^k stack symbols.

Determinizing k-entry DIDPDAs

Lemma

A k-entry DIDPDA A with n states and m stack symbols simulated by a DIDPDA B with $(n+1)^k - 1$ states and m^k stack symbols.

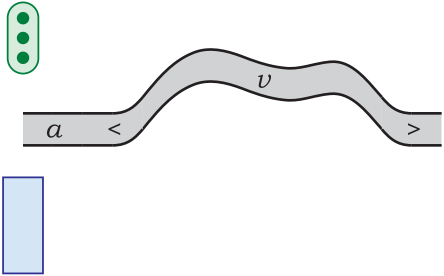
- B simulates k computations.

Determinizing k-entry DIDPDAs

Lemma

A k-entry DIDPDA A with n states and m stack symbols simulated by a DIDPDA B with $(n+1)^k - 1$ states and m^k stack symbols.

- B simulates k computations.
- B uses k-tuples of states.

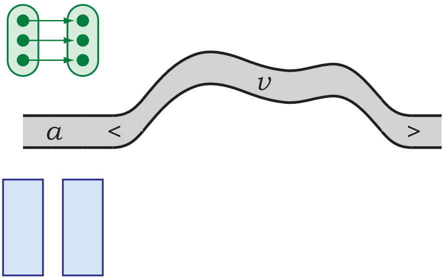


Determinizing k-entry DIDPDAs

Lemma

A k-entry DIDPDA A with n states and m stack symbols simulated by a DIDPDA B with $(n+1)^k - 1$ states and m^k stack symbols.

- B simulates k computations.
- B uses k-tuples of states.

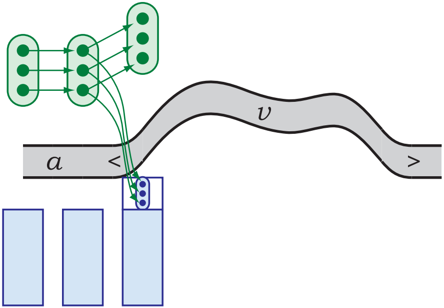


Determinizing k-entry DIDPDAs

Lemma

A k-entry DIDPDA A with n states and m stack symbols simulated by a DIDPDA B with $(n+1)^k - 1$ states and m^k stack symbols.

- B simulates k computations.
- B uses k-tuples of states.
- B pushes k-tuples of stack symbols.

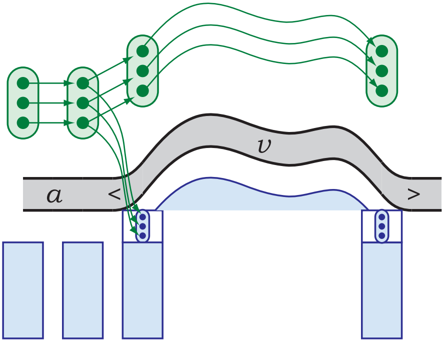


Determinizing k-entry DIDPDAs

Lemma

A k-entry DIDPDA A with n states and m stack symbols simulated by a DIDPDA B with $(n+1)^k - 1$ states and m^k stack symbols.

- B simulates k computations.
- B uses k-tuples of states.
- B pushes k-tuples of stack symbols.

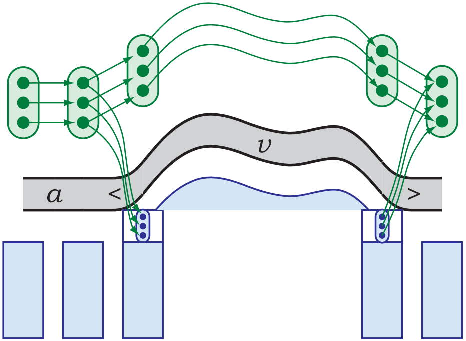


Determinizing k-entry DIDPDAs

Lemma

A k-entry DIDPDA A with n states and m stack symbols simulated by a DIDPDA B with $(n+1)^k - 1$ states and m^k stack symbols.

- B simulates k computations.
- B uses k-tuples of states.
- B pushes k-tuples of stack symbols.
- each state matched to a corresponding stack symbol.



Determinizing k-path NIDPDAs

Lemma

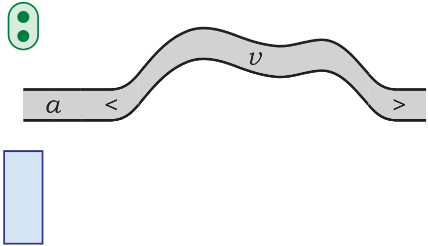
A k-path NIDPDA A with n states and m stack symbols simulated by a DIDPDA B with $\sum_{i=1}^{k}(n+1)^i \cdot i^i$ states and $\sum_{i=1}^{k} m^i$ stack symbols.

Determinizing k-path NIDPDAs

Lemma

A k-path NIDPDA A with n states and m stack symbols simulated by a DIDPDA B with $\sum_{i=1}^{k}(n+1)^i \cdot i^i$ states and $\sum_{i=1}^{k} m^i$ stack symbols.

- Begins with $k_0 \leqslant k$ computations.

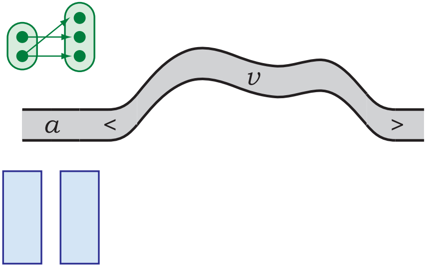


Determinizing k-path NIDPDAs

Lemma

A k-path NIDPDA A with n states and m stack symbols simulated by a DIDPDA B with $\sum_{i=1}^{k}(n+1)^i \cdot i^i$ states and $\sum_{i=1}^{k} m^i$ stack symbols.

- Begins with $k_0 \leqslant k$ computations.
- New computations may emerge at each step.

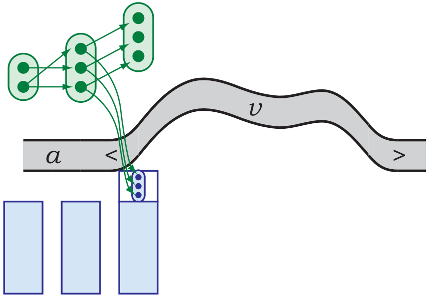


Determinizing k-path NIDPDAs

Lemma

A k-path NIDPDA A with n states and m stack symbols simulated by a DIDPDA B with $\sum_{i=1}^{k}(n+1)^i \cdot i^i$ states and $\sum_{i=1}^{k} m^i$ stack symbols.

- Begins with $k_0 \leqslant k$ computations.
- New computations may emerge at each step.
- Pushes ℓ-tuples.

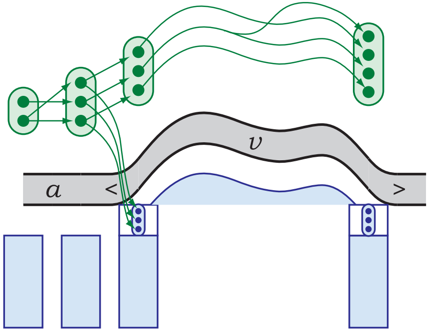


Determinizing k-path NIDPDAs

Lemma

A k-path NIDPDA A with n states and m stack symbols simulated by a DIDPDA B with $\sum_{i=1}^{k}(n+1)^i \cdot i^i$ states and $\sum_{i=1}^{k} m^i$ stack symbols.

- Begins with $k_0 \leqslant k$ computations.
- New computations may emerge at each step.
- Pushes ℓ-tuples.
- May branch inside the brackets.

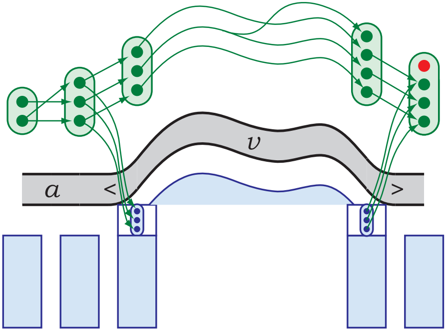


Determinizing k-path NIDPDAs

Lemma

A k-path NIDPDA A with n states and m stack symbols simulated by a DIDPDA B with $\sum_{i=1}^{k}(n+1)^i \cdot i^i$ states and $\sum_{i=1}^{k} m^i$ stack symbols.

- Begins with $k_0 \leqslant k$ computations.
- New computations may emerge at each step.
- Pushes ℓ-tuples.
- May branch inside the brackets.
- How to match $\ell' > \ell$ states to ℓ symbols?

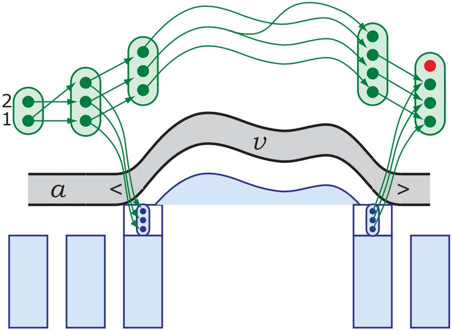


Determinizing k-path NIDPDAs

Lemma

A k-path NIDPDA A with n states and m stack symbols simulated by a DIDPDA B with $\sum_{i=1}^{k}(n+1)^i \cdot i^i$ states and $\sum_{i=1}^{k} m^i$ stack symbols.

- Begins with $k_0 \leqslant k$ computations.
- New computations may emerge at each step.
- Pushes ℓ-tuples.
- May branch inside the brackets.
- How to match $\ell' > \ell$ states to ℓ symbols?



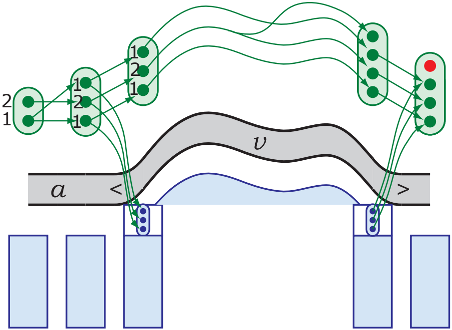
✓ Mark each component with the number of parent component.

Determinizing k-path NIDPDAs

Lemma

A k-path NIDPDA A with n states and m stack symbols simulated by a DIDPDA B with $\sum_{i=1}^{k}(n+1)^i \cdot i^i$ states and $\sum_{i=1}^{k} m^i$ stack symbols.

- Begins with $k_0 \leqslant k$ computations.
- New computations may emerge at each step.
- Pushes ℓ-tuples.
- May branch inside the brackets.
- How to match $\ell' > \ell$ states to ℓ symbols?



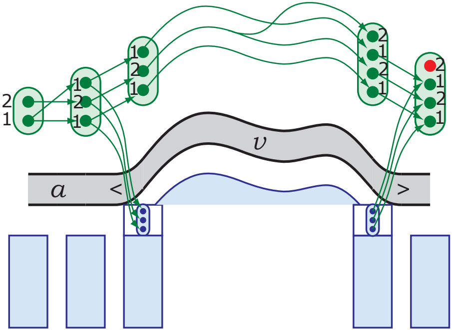
✓ Mark each component with the number of parent component.

Determinizing k-path NIDPDAs

Lemma

A k-path NIDPDA A with n states and m stack symbols simulated by a DIDPDA B with $\sum_{i=1}^{k}(n+1)^i \cdot i^i$ states and $\sum_{i=1}^{k} m^i$ stack symbols.

- Begins with $k_0 \leqslant k$ computations.
- New computations may emerge at each step.
- Pushes ℓ-tuples.
- May branch inside the brackets.
- How to match $\ell' > \ell$ states to ℓ symbols?



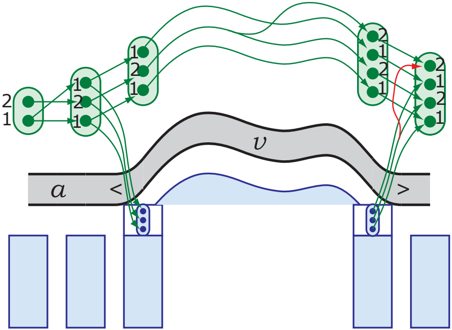
✓ Mark each component with the number of parent component.

Determinizing k-path NIDPDAs

Lemma

A k-path NIDPDA A with n states and m stack symbols simulated by a DIDPDA B with $\sum_{i=1}^{k}(n+1)^i \cdot i^i$ states and $\sum_{i=1}^{k} m^i$ stack symbols.

- Begins with $k_0 \leqslant k$ computations.
- New computations may emerge at each step.
- Pushes ℓ-tuples.
- May branch inside the brackets.
- How to match $\ell' > \ell$ states to ℓ symbols?



✓ Mark each component with the number of parent component.

- Use these data to match ℓ symbols to ℓ' states.

Lower bound on size blow-up of determinization

- For k-entry DIDPDAs, tight lower bound.
- The alphabet depends on n and k.

Lemma

For every $k \geqslant 1$ and $n \geqslant k$, there exists an alphabet $\Sigma^{k,n}$ and a language $L_{k,n}$ over $\Sigma^{k,n}$ recognized by a k-entry DIDPDA with n states and k stack symbols, such that any DIDPDA for $L_{k,n}$ needs $(n+1)^k - 1$ states.

Lower bound on size blow-up of determinization

- For k-entry DIDPDAs, tight lower bound.
- The alphabet depends on n and k.

Lemma

For every $k \geqslant 1$ and $n \geqslant k$, there exists an alphabet $\Sigma^{k,n}$ and a language $L_{k,n}$ over $\Sigma^{k,n}$ recognized by a k-entry DIDPDA with n states and k stack symbols, such that any DIDPDA for $L_{k,n}$ needs $(n+1)^k - 1$ states.

- the alphabet has one left bracket $<$, one right bracket $>$, and a large number of neutral symbols $\Sigma_0^{k,n} = X_{\text{func}} \cup Y_{\text{func}}$, where
  - $X_{\text{func}} = \{ \ a_f \mid f \colon \{1, \ldots, k\} \to \{1, \ldots, n, \text{undefined}\} \ \}$
  - $Y_{\text{func}} = \{ \ b_g \mid g \colon \{1, \ldots, n\} \to \{1, \ldots, k, \text{undefined}\} \ \}$

Lower bound language

(proof continued)

$$\widehat{L}_{k,n} = \{<a_f b_g> \mid a_f, b_g \in \Sigma_0^{k,n},\ \exists s \in \{1,\ldots,k\} :\ g(f(s)) = s\}.$$

Lower bound language

(proof continued)

$$\widehat{L}_{k,n} = \{<a_f b_g> \mid a_f, b_g \in \Sigma_0^{k,n},\ \exists s \in \{1, \ldots, k\} :\ g(f(s)) = s\}.$$

- strings of $\widehat{L}_{k,n}$ consist of two symbols indexed by functions whose composition has a fixed point

Lower bound language

(proof continued)

$$\widehat{L}_{k,n} = \{<a_f b_g> \mid a_f, b_g \in \Sigma_0^{k,n},\ \exists s \in \{1, \ldots, k\} :\ g(f(s)) = s\}.$$

- strings of $\widehat{L}_{k,n}$ consist of two symbols indexed by functions whose composition has a fixed point
- a string of the form $<a_f b_g>$ ($f \in X_{\mathrm{func}}$, $g \in Y_{\mathrm{func}}$) is said to be *well-formed*

Lower bound language

(proof continued)

$$\widehat{L}_{k,n} = \{<a_f b_g> \mid a_f, b_g \in \Sigma_0^{k,n},\ \exists s \in \{1,\ldots,k\} :\ g(f(s)) = s\}.$$

- strings of $\widehat{L}_{k,n}$ consist of two symbols indexed by functions whose composition has a fixed point
- a string of the form $<a_f b_g>$ ($f \in X_{\mathrm{func}}$, $g \in Y_{\mathrm{func}}$) is said to be *well-formed*
- a k-entry DIDPDA A with n states and k stack symbols that accepts well-formed strings from $\widehat{L}_{k,n}$ (as well as some ill-formed strings)
  - a k-entry DIDPDA for $\widehat{L}_{k,n}$ would need more states

Separator sets

(somewhat simplified definition)

A set $S = \{ \lhd x_1, \ldots \lhd x_m \}$, $x_1, \ldots, x_m \in \Sigma_0^*$, is a 1-separator set for language L if

- each $\lhd x_i$ is a prefix of some string in L,
- for all $i \neq j$. there exists $w_{i,j} \in \Sigma^*$ such that exactly one of $x_i w_{i,j}$ and $x_j w_{i,j}$ is in L.

Separator sets

(somewhat simplified definition)

A set $S = \{ <x_1, \ldots <x_m \}$, $x_1, \ldots, x_m \in \Sigma_0^*$, is a 1-separator set for language L if

- each $<x_i$ is a prefix of some string in L,
- for all $i \neq j$. there exists $w_{i,j} \in \Sigma^*$ such that exactly one of $x_i w_{i,j}$ and $x_j w_{i,j}$ is in L.

- Separator sets are analogous to the notion of fooling sets used to prove lower bounds for NFAs.

Separator sets

(somewhat simplified definition)

A set $S = \{ <x_1, \ldots <x_m \}$, $x_1, \ldots, x_m \in \Sigma_0^*$, is a 1-separator set for language L if

- each $<x_i$ is a prefix of some string in L,
- for all $i \neq j$. there exists $w_{i,j} \in \Sigma^*$ such that exactly one of $x_i w_{i,j}$ and $x_j w_{i,j}$ is in L.

- Separator sets are analogous to the notion of fooling sets used to prove lower bounds for NFAs.
- The definition used here is more specialized.

Separator sets

(somewhat simplified definition)

A set $S = \{ <x_1, \ldots <x_m \}$, $x_1, \ldots, x_m \in \Sigma_0^*$, is a 1-separator set for language L if

- each $<x_i$ is a prefix of some string in L,
- for all $i \neq j$. there exists $w_{i,j} \in \Sigma^*$ such that exactly one of $x_i w_{i,j}$ and $x_j w_{i,j}$ is in L.

- Separator sets are analogous to the notion of fooling sets used to prove lower bounds for NFAs.
- The definition used here is more specialized.

Lemma

If L has a separator set of cardinality m, then any DIDPDA of L must have at least m states.

Separator sets

(somewhat simplified definition)

A set $S = \{ <x_1, \ldots <x_m \}$, $x_1, \ldots, x_m \in \Sigma_0^*$, is a 1-separator set for language L if

- each $<x_i$ is a prefix of some string in L,
- for all $i \neq j$. there exists $w_{i,j} \in \Sigma^*$ such that exactly one of $x_i w_{i,j}$ and $x_j w_{i,j}$ is in L.

- Separator sets are analogous to the notion of fooling sets used to prove lower bounds for NFAs.
- The definition used here is more specialized.

Lemma

If L has a separator set of cardinality m, then any DIDPDA of L must have at least m states.

- Proof is straightforward.

Separator set for $\widehat{L}_{k,n}$
(lower bound proof continued)

$$\widehat{L}_{k,n} = \{<a_f b_g> \mid a_f, b_g \in \Sigma_0^{k,n},\ \exists s \in \{1, \ldots, k\} :\ g(f(s)) = s\}.$$

Separator set:

$$S = \{<a_f \mid a_f \in X_{\mathrm{func}},\ \exists i \in \{1, \ldots, k\} :\ f(i) \text{ is defined}\}$$

Separator set for $\widehat{L}_{k,n}$

(lower bound proof continued)

$$\widehat{L}_{k,n} = \{<a_f b_g> \mid a_f, b_g \in \Sigma_0^{k,n},\ \exists s \in \{1, \ldots, k\} :\ g(f(s)) = s\}.$$

Separator set:

$$S = \{<a_f \mid a_f \in X_{\mathrm{func}},\ \exists i \in \{1, \ldots, k\} :\ f(i) \text{ is defined}\}$$

- consider two distinct strings $<a_{f_1}, <a_{f_2} \in S$, with $f_1 \neq f_2$

Separator set for $\widehat{L}_{k,n}$
(lower bound proof continued)

$$\widehat{L}_{k,n} = \{<a_f b_g> \mid a_f, b_g \in \Sigma_0^{k,n},\ \exists s \in \{1, \dots, k\} :\ g(f(s)) = s\}.$$

Separator set:

$$S = \{<a_f \mid a_f \in X_{\text{func}},\ \exists i \in \{1, \dots, k\} :\ f(i) \text{ is defined}\}$$

- consider two distinct strings $<a_{f_1}, <a_{f_2} \in S$, with $f_1 \neq f_2$
- there exists i for which $f_1(i)$ is defined and $f_2(i) \neq f_1(i)$ (or vice versa)

Separator set for $\widehat{L}_{k,n}$
(lower bound proof continued)

$$\widehat{L}_{k,n} = \{<a_f b_g> \mid a_f, b_g \in \Sigma_0^{k,n},\ \exists s \in \{1, \dots, k\} :\ g(f(s)) = s\}.$$

Separator set:

$$S = \{<a_f \mid a_f \in X_{\text{func}},\ \exists i \in \{1, \dots, k\} :\ f(i) \text{ is defined}\}$$

- consider two distinct strings $<a_{f_1}, <a_{f_2} \in S$, with $f_1 \neq f_2$
- there exists i for which $f_1(i)$ is defined and $f_2(i) \neq f_1(i)$ (or vice versa)
- define

$$g(j) = \begin{cases} i, & \text{if } j = f_1(i), \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

Separator set for $\widehat{L}_{k,n}$
(lower bound proof continued)

$$\widehat{L}_{k,n} = \{<a_f b_g> \mid a_f, b_g \in \Sigma_0^{k,n},\ \exists s \in \{1, \ldots, k\} :\ g(f(s)) = s\}.$$

Separator set:

$$S = \{<a_f \mid a_f \in X_{\text{func}},\ \exists i \in \{1, \ldots, k\} :\ f(i) \text{ is defined}\}$$

- consider two distinct strings $<a_{f_1}, <a_{f_2} \in S$, with $f_1 \neq f_2$
- there exists i for which $f_1(i)$ is defined and $f_2(i) \neq f_1(i)$ (or vice versa)
- define
$$g(j) = \begin{cases} i, & \text{if } j = f_1(i), \\ \text{undefined}, & \text{otherwise.} \end{cases}$$
- $<a_{f_1} b_g> \in \widehat{L}_{k,n}$ and $<a_{f_2} b_g> \notin \widehat{L}_{k,n}$

k-entry DIDPDA to DIDPDA

- For k-entry DIDPDAs the bound is tight

Theorem

- *A k-entry DIDPDA with n states and m stack symbols can be simulated by a DIDPDA with* $(n+1)^k - 1$ *states and* m^k *stack symbols.*

k-entry DIDPDA to DIDPDA

- For k-entry DIDPDAs the bound is tight

Theorem

- *A k-entry DIDPDA with n states and m stack symbols can be simulated by a DIDPDA with $(n+1)^k - 1$ states and m^k stack symbols.*
- *For every $n \geqslant k$, there exists a k-entry DIDPDA with n states and k stack symbols, defined over an alphabet depending on n and k, such that any equivalent DIDPDA needs at least $(n+1)^k - 1$ states.*

k-entry DIDPDA to DIDPDA

- For k-entry DIDPDAs the bound is tight

Theorem

- *A k-entry DIDPDA with n states and m stack symbols can be simulated by a DIDPDA with $(n+1)^k - 1$ states and m^k stack symbols.*
- *For every $n \geqslant k$, there exists a k-entry DIDPDA with n states and k stack symbols, defined over an alphabet depending on n and k, such that any equivalent DIDPDA needs at least $(n+1)^k - 1$ states.*

- Above lower bound uses a growing alphabet.

k-entry DIDPDA to DIDPDA

- For k-entry DIDPDAs the bound is tight

Theorem

- *A k-entry DIDPDA with n states and m stack symbols can be simulated by a DIDPDA with $(n+1)^k - 1$ states and m^k stack symbols.*
- *For every $n \geqslant k$, there exists a k-entry DIDPDA with n states and k stack symbols, defined over an alphabet depending on n and k, such that any equivalent DIDPDA needs at least $(n+1)^k - 1$ states.*

- Above lower bound uses a growing alphabet.
- For 6-symbol alphabet: lower bound $(\frac{n}{4})^k$ on the size of a DIDPDA simulating a k-entry automaton with n states and k stack symbols.

k-path NIDPDA to DIDPDA

For a k-path NIDPDA with n states and m stack symbols:

- *upper bound:*

$$\sum_{i=1}^{k}(n+1)^i \cdot i^i \text{ states and } \sum_{i=1}^{k} m^i \text{ stack symbols.}$$

k-path NIDPDA to DIDPDA

For a k-path NIDPDA with n states and m stack symbols:

- *upper bound:*

$$\sum_{i=1}^{k}(n+1)^i \cdot i^i \text{ states and } \sum_{i=1}^{k} m^i \text{ stack symbols.}$$

- *lower bound:* $(n+1)^k - 1$ states

k-path NIDPDA to DIDPDA

For a k-path NIDPDA with n states and m stack symbols:

- *upper bound:*

$$\sum_{i=1}^{k}(n+1)^i \cdot i^i \text{ states and } \sum_{i=1}^{k} m^i \text{ stack symbols.}$$

- *lower bound:* $(n+1)^k - 1$ states
- The lower bound uses a k-entry DIDPDA (discussed above).

k-path NIDPDA to DIDPDA

For a k-path NIDPDA with n states and m stack symbols:

- *upper bound:*

$$\sum_{i=1}^{k}(n+1)^i \cdot i^i \text{ states and } \sum_{i=1}^{k} m^i \text{ stack symbols.}$$

- *lower bound:* $(n+1)^k - 1$ states
- The lower bound uses a k-entry DIDPDA (discussed above).
- *Open questions:*
  - Can the lower bound be improved for general k-entry DIDPDA?
  - Can the upper bound construction be improved?

NIDPDA to a k-path NIDPDA

Theorem

For any $k \geq 1$, the worst-case number of states in a k-path NIDPDA equivalent to an NIDPDA with n states is $2^{\Theta(n^2)}$.

Proof inspired by an argument by Goldstine, Kintala and Wotschke (1990) for NFAs with finite branching:

NIDPDA to a k-path NIDPDA

Theorem

For any $k \geq 1$, the worst-case number of states in a k-path NIDPDA equivalent to an NIDPDA with n states is $2^{\Theta(n^2)}$.

Proof inspired by an argument by Goldstine, Kintala and Wotschke (1990) for NFAs with finite branching:

- Let A be an NIDPDA with "maximal" size blow-up for determinization.

NIDPDA to a k-path NIDPDA

Theorem

For any $k \geq 1$, the worst-case number of states in a k-path NIDPDA equivalent to an NIDPDA with n states is $2^{\Theta(n^2)}$.

Proof inspired by an argument by Goldstine, Kintala and Wotschke (1990) for NFAs with finite branching:

- Let A be an NIDPDA with "maximal" size blow-up for determinization.
- Consider the language $(\$L(A)\$)^*$

NIDPDA to a k-path NIDPDA

Theorem

For any $k \geq 1$, the worst-case number of states in a k-path NIDPDA equivalent to an NIDPDA with n states is $2^{\Theta(n^2)}$.

Proof inspired by an argument by Goldstine, Kintala and Wotschke (1990) for NFAs with finite branching:

- Let A be an NIDPDA with "maximal" size blow-up for determinization.
- Consider the language $(\$L(A)\$)^*$
- k-path NIDPDA for $(\$L(A)\$)^*$ cannot be smaller than a minimal DIDPDA.

Deciding the k-path property

- The k-path property of an NIDPDA refers to its computations on all possible inputs

Deciding the k-path property

- The k-path property of an NIDPDA refers to its computations on all possible inputs
  - given the syntactic specification of an NIDPDA A, it is not at all clear whether or not A has this property

Deciding the k-path property

- The k-path property of an NIDPDA refers to its computations on all possible inputs
  - given the syntactic specification of an NIDPDA A, it is not at all clear whether or not A has this property

Lemma

Given an NIDPDA A with n states and a number $k \geqslant 1$, one can decide in time poly$(k^k \cdot n^k)$ whether or not A has the k-path property.

Deciding the k-path property

- The k-path property of an NIDPDA refers to its computations on all possible inputs
  - given the syntactic specification of an NIDPDA A, it is not at all clear whether or not A has this property

Lemma

Given an NIDPDA A with n states and a number $k \geqslant 1$, one can decide in time poly$(k^k \cdot n^k)$ whether or not A has the k-path property.

- Algorithm constructs a DIDPDA B which accepts input w if A has more than k computations on w
  - decide emptiness for B (in polynomial time)

Deciding the k-path property

- The k-path property of an NIDPDA refers to its computations on all possible inputs
  - given the syntactic specification of an NIDPDA A, it is not at all clear whether or not A has this property

Lemma

Given an NIDPDA A with n states and a number $k \geqslant 1$, one can decide in time poly$(k^k \cdot n^k)$ whether or not A has the k-path property.

- Algorithm constructs a DIDPDA B which accepts input w if A has more than k computations on w
  - decide emptiness for B (in polynomial time)

Theorem

For a fixed $k \geqslant 1$, checking whether or not a given NIDPDA has the k-path property is P-complete.

Deciding the k-path property

- The k-path property of an NIDPDA refers to its computations on all possible inputs
  - given the syntactic specification of an NIDPDA A, it is not at all clear whether or not A has this property

Lemma

Given an NIDPDA A with n states and a number $k \geqslant 1$, one can decide in time poly$(k^k \cdot n^k)$ whether or not A has the k-path property.

- Algorithm constructs a DIDPDA B which accepts input w if A has more than k computations on w
  - decide emptiness for B (in polynomial time)

Theorem

For a fixed $k \geqslant 1$, checking whether or not a given NIDPDA has the k-path property is P-complete.

- Proof uses reduction from DIDPDA emptiness problem (in logspace)

Deciding the finite path property

(with the value of k arbitrary)

- Finite path property for NFAs can be determined by analyzing the transition graph

Deciding the finite path property

(with the value of k arbitrary)

- Finite path property for NFAs can be determined by analyzing the transition graph
- The method does not work in the precense of stack operations

Deciding the finite path property

(with the value of k arbitrary)

- Finite path property for NFAs can be determined by analyzing the transition graph
- The method does not work in the precense of stack operations
  - It is undecidable whether a (general) nondeterministic PDA is 3-path

Deciding the finite path property

(with the value of k arbitrary)

- Finite path property for NFAs can be determined by analyzing the transition graph
- The method does not work in the precense of stack operations
  - It is undecidable whether a (general) nondeterministic PDA is 3-path

Open problem

Is it decidable whether or not a given NIDPDA has the finite path property?

Conclusion

Main open problems:

- What is the precise size blow-up of determinizing k-path NIDPDAs ?

Conclusion

Main open problems:

- What is the precise size blow-up of determinizing k-path NIDPDAs ?
- Can the finite path property be decided effectively ?
  - Not just a question of complexity – even the decidability status of the question is open

Conclusion

Main open problems:

- What is the precise size blow-up of determinizing k-path NIDPDAs ?
- Can the finite path property be decided effectively ?
  - Not just a question of complexity – even the decidability status of the question is open
- The amount of nondeterminism can be limited by a function on input length (analogously as has been done with NFAs).
  Questions:

Conclusion

Main open problems:

- What is the precise size blow-up of determinizing k-path NIDPDAs ?
- Can the finite path property be decided effectively ?
  - Not just a question of complexity – even the decidability status of the question is open
- The amount of nondeterminism can be limited by a function on input length (analogously as has been done with NFAs).
  Questions:
  - Descriptional complexity of determinization

Conclusion

Main open problems:

- What is the precise size blow-up of determinizing k-path NIDPDAs ?
- Can the finite path property be decided effectively ?
  - Not just a question of complexity – even the decidability status of the question is open
- The amount of nondeterminism can be limited by a function on input length (analogously as has been done with NFAs).
  Questions:
  - Descriptional complexity of determinization
  - Characterization of possible nondeterminism growth rates for an NIDPDA

Conclusion

Extended models

- *Alternating input-driven pushdown automata* L. Bozzelli (2007), C. Dax, F. Klaedtke (2011), M. Schuster, T. Schwentick (2014)

Conclusion

Extended models

- *Alternating input-driven pushdown automata* L. Bozzelli (2007), C. Dax, F. Klaedtke (2011), M. Schuster, T. Schwentick (2014)
  - Non-emptiness for alternating IDPDA is complete for doubly exponential time (Bozzelli 2007)

Conclusion

Extended models

- *Alternating input-driven pushdown automata* L. Bozzelli (2007), C. Dax, F. Klaedtke (2011), M. Schuster, T. Schwentick (2014)
  - Non-emptiness for alternating IDPDA is complete for doubly exponential time (Bozzelli 2007)
  - Uniform membership for alternating IDPDA is PSPACE-complete (Schuster & Schwentick 2014)

Conclusion

Extended models

- *Alternating input-driven pushdown automata* L. Bozzelli (2007), C. Dax, F. Klaedtke (2011), M. Schuster, T. Schwentick (2014)
  - Non-emptiness for alternating IDPDA is complete for doubly exponential time (Bozzelli 2007)
  - Uniform membership for alternating IDPDA is PSPACE-complete (Schuster & Schwentick 2014)
  - *Open:* What is the descriptional complexity trade-off between alternating IDPDAs and nondeterministic (resp. deterministic) IDPDAs

Conclusion

Extended models

- *Alternating input-driven pushdown automata* L. Bozzelli (2007), C. Dax, F. Klaedtke (2011), M. Schuster, T. Schwentick (2014)
  - Non-emptiness for alternating IDPDA is complete for doubly exponential time (Bozzelli 2007)
  - Uniform membership for alternating IDPDA is PSPACE-complete (Schuster & Schwentick 2014)
  - *Open:* What is the descriptional complexity trade-off between alternating IDPDAs and nondeterministic (resp. deterministic) IDPDAs
- *Two-way input-driven pushdown automata*

Conclusion

Extended models

- *Alternating input-driven pushdown automata* L. Bozzelli (2007), C. Dax, F. Klaedtke (2011), M. Schuster, T. Schwentick (2014)
  - Non-emptiness for alternating IDPDA is complete for doubly exponential time (Bozzelli 2007)
  - Uniform membership for alternating IDPDA is PSPACE-complete (Schuster & Schwentick 2014)
  - *Open:* What is the descriptional complexity trade-off between alternating IDPDAs and nondeterministic (resp. deterministic) IDPDAs
- *Two-way input-driven pushdown automata*
  - Natural definition of 2-way IDPDA: when moving right-to-left, the automaton pushes at symbols of Σ_{-1} and pops at symbols of Σ_{+1}

Conclusion

Extended models

- *Alternating input-driven pushdown automata* L. Bozzelli (2007), C. Dax, F. Klaedtke (2011), M. Schuster, T. Schwentick (2014)
  - Non-emptiness for alternating IDPDA is complete for doubly exponential time (Bozzelli 2007)
  - Uniform membership for alternating IDPDA is PSPACE-complete (Schuster & Schwentick 2014)
  - *Open:* What is the descriptional complexity trade-off between alternating IDPDAs and nondeterministic (resp. deterministic) IDPDAs
- *Two-way input-driven pushdown automata*
  - Natural definition of 2-way IDPDA: when moving right-to-left, the automaton pushes at symbols of Σ_{-1} and pops at symbols of Σ_{+1}
  - The class of recognized languages remains the same

Conclusion

Extended models

- *Alternating input-driven pushdown automata* L. Bozzelli (2007), C. Dax, F. Klaedtke (2011), M. Schuster, T. Schwentick (2014)
  - Non-emptiness for alternating IDPDA is complete for doubly exponential time (Bozzelli 2007)
  - Uniform membership for alternating IDPDA is PSPACE-complete (Schuster & Schwentick 2014)
  - *Open:* What is the descriptional complexity trade-off between alternating IDPDAs and nondeterministic (resp. deterministic) IDPDAs
- *Two-way input-driven pushdown automata*
  - Natural definition of 2-way IDPDA: when moving right-to-left, the automaton pushes at symbols of Σ_{-1} and pops at symbols of Σ_{+1}
  - The class of recognized languages remains the same
  - Descriptional complexity of 2-way IDPDA to 1-way IDPDA conversion

Conclusion

Extended models

- *Alternating input-driven pushdown automata* L. Bozzelli (2007), C. Dax, F. Klaedtke (2011), M. Schuster, T. Schwentick (2014)
  - Non-emptiness for alternating IDPDA is complete for doubly exponential time (Bozzelli 2007)
  - Uniform membership for alternating IDPDA is PSPACE-complete (Schuster & Schwentick 2014)
  - *Open:* What is the descriptional complexity trade-off between alternating IDPDAs and nondeterministic (resp. deterministic) IDPDAs
- *Two-way input-driven pushdown automata*
  - Natural definition of 2-way IDPDA: when moving right-to-left, the automaton pushes at symbols of Σ_{-1} and pops at symbols of Σ_{+1}
  - The class of recognized languages remains the same
  - Descriptional complexity of 2-way IDPDA to 1-way IDPDA conversion
  - Complexity of decision problems

Questions?