Simulating Unary Context-Free Grammars and Pushdown Automata with Finite Automata

Giovanni Pighizzini

Dipartimento di Informatica e Comunicazione
Università degli Studi di Milano
ITALY

P.J. Šafárik University – Košice – Slovak Republic
November 12th, 2008

Outline of the talk

1. Introduction
2. Simulation of unary cfg's by nfa's
3. Simulation of unary cfg's by dfa's
4.

The results presented at points 2, 3, 4, and 5 are from [Pighizzini, Shallit, Wang, 2002].

Outline of the talk

1. Introduction
2. Simulation of unary cfg's by nfa's
3. Simulation of unary cfg's by dfa's
4. Auxiliary pushdown automata
5.

The results presented at points 2, 3, 4, and 5 are from [Pighizzini, Shallit, Wang, 2002].

Outline of the talk

1. Introduction
2. Simulation of unary cfg's by nfa's
3. Simulation of unary cfg's by dfa's
4. Auxiliary pushdown automata
5. Space lower bound for 1auxpda's accepting noncontext-free languages
6.

The results presented at points 2, 3, 4, and 5 are from [Pighizzini, Shallit, Wang, 2002].

Outline of the talk

1. Introduction
2. Simulation of unary cfg's by nfa's
3. Simulation of unary cfg's by dfa's
4. Auxiliary pushdown automata
5. Space lower bound for 1auxpda's accepting noncontext-free languages
6. Extension to bounded languages
7.

The results presented at points 2, 3, 4, and 5 are from [Pighizzini, Shallit, Wang, 2002].

Outline of the talk

1. Introduction
2. Simulation of unary cfg's by nfa's
3. Simulation of unary cfg's by dfa's
4. Auxiliary pushdown automata
5. Space lower bound for 1auxpda's accepting noncontext-free languages
6. Extension to bounded languages
7. Final remarks

The results presented at points 2, 3, 4, and 5 are from [Pighizzini, Shallit, Wang, 2002].

Outline of the talk

1. Introduction
2. Simulation of unary cfg's by nfa's
3. Simulation of unary cfg's by dfa's
4. Auxiliary pushdown automata
5. Space lower bound for 1auxpda's accepting noncontext-free languages
6. Extension to bounded languages
7. Final remarks

The results presented at points 2, 3, 4, and 5 are from [Pighizzini, Shallit, Wang, 2002].

Outline of the talk

1. Introduction
2. Simulation of unary cfg's by nfa's
3. Simulation of unary cfg's by dfa's
4. Auxiliary pushdown automata
5. Space lower bound for 1auxpda's accepting noncontext-free languages
6. Extension to bounded languages
7. Final remarks

The results presented at points 2, 3, 4, and 5 are from [Pighizzini, Shallit, Wang, 2002].

Outline of the talk

1. Introduction
2. Simulation of unary cfg's by nfa's
3. Simulation of unary cfg's by dfa's
4. Auxiliary pushdown automata
5. Space lower bound for 1auxpda's accepting noncontext-free languages
6. Extension to bounded languages
7. Final remarks

The results presented at points 2, 3, 4, and 5 are from [Pighizzini, Shallit, Wang, 2002].

Outline of the talk

1. Introduction
2. Simulation of unary cfg's by nfa's
3. Simulation of unary cfg's by dfa's
4. Auxiliary pushdown automata
5. Space lower bound for 1auxpda's accepting noncontext-free languages
6. Extension to bounded languages
7. Final remarks

The results presented at points 2, 3, 4, and 5 are from [Pighizzini, Shallit, Wang, 2002].

Descriptional complexity

Given

- $\mathcal{C}$, a class of languages
- $\mathcal{S}$, a formal system (e.g., class of devices, class of grammars,..) able to represent all the languages in $\mathcal{C}$

What is the *size* of the representations of the languages in $\mathcal{C}$ by the system $\mathcal{S}$?

Usually, descriptional complexity compares different description for a same class of languages:

- given $\mathcal{S}'$, another formal system able to represent all the languages in $\mathcal{C}$

Descriptional complexity

Given

- $\mathcal{C}$, a class of languages
- $\mathcal{S}$, a formal system (e.g., class of devices, class of grammars,..) able to represent all the languages in $\mathcal{C}$

What is the *size* of the representations of the languages in $\mathcal{C}$ by the system $\mathcal{S}$?

Usually, descriptional complexity compares different description for a same class of languages:

- given $\mathcal{S}'$, another formal system able to represent all the languages in $\mathcal{C}$

What is the *size* of the representations of the languages in $\mathcal{C}$ by the system $\mathcal{S}'$, with respect to the size of their representations by the system $\mathcal{S}$?

Descriptional complexity

Given

- $\mathcal{C}$, a class of languages
- $\mathcal{S}$, a formal system (e.g., class of devices, class of grammars,..) able to represent all the languages in $\mathcal{C}$

What is the *size* of the representations of the languages in $\mathcal{C}$ by the system $\mathcal{S}$?

Usually, descriptional complexity compares different description for a same class of languages:

- given $\mathcal{S}'$, another formal system able to represent all the languages in $\mathcal{C}$

What is the *size* of the representations of the languages in $\mathcal{C}$ by the system $\mathcal{S}'$, with respect to the size of their representations by the system $\mathcal{S}$?

Descriptional complexity

Given

- $\mathcal{C}$, a class of languages
- $\mathcal{S}$, a formal system (e.g., class of devices, class of grammars,..) able to represent all the languages in $\mathcal{C}$

What is the *size* of the representations of the languages in $\mathcal{C}$ by the system $\mathcal{S}$?

Usually, descriptional complexity compares different description for a same class of languages:

- given $\mathcal{S}'$, another formal system able to represent all the languages in $\mathcal{C}$

What is the *size* of the representations of the languages in $\mathcal{C}$ by the system $\mathcal{S}'$, with respect to the size of their representations by the system $\mathcal{S}$?

Descriptional complexity

Given

- $\mathcal{C}$, a class of languages
- $\mathcal{S}$, a formal system (e.g., class of devices, class of grammars,..) able to represent all the languages in $\mathcal{C}$

What is the *size* of the representations of the languages in $\mathcal{C}$ by the system $\mathcal{S}$?

Usually, descriptional complexity compares different description for a same class of languages:

- given $\mathcal{S}'$, another formal system able to represent all the languages in $\mathcal{C}$

What is the *size* of the representations of the languages in $\mathcal{C}$ by the system $\mathcal{S}'$, with respect to the size of their representations by the system $\mathcal{S}$?

Descriptional complexity

Classical example: deterministic vs. nondeterministic automata

- Formal language point of view:

  nondeterministic finite automata are as powerful as deterministic finite automata

- Descriptional complexity point of view:

  Each n-state nfa can be simulated by a 2^n state dfa (upper bound)

  For each integer n there exists a language accepted by an n-state nfa such that the minimun equivalent dfa requires 2^n states

Descriptional complexity

Classical example: deterministic vs. nondeterministic automata

- Formal language point of view:

  nondeterministic finite automata are as powerful as deterministic finite automata

- Descriptional complexity point of view:

  Each n-state nfa can be simulated by a 2^n state dfa (upper bound)

  For each integer n there exists a language accepted by an n-state nfa such that the minimun equivalent dfa requires 2^n states (lower bound)

  Hence:

  The cost, in terms of states, of the simulation of nfa's by dfa's is 2^n.

Descriptional complexity

Classical example: deterministic vs. nondeterministic automata

- Formal language point of view:

  nondeterministic finite automata are as powerful as deterministic finite automata

- Descriptional complexity point of view:

  Each n-state nfa can be simulated by a 2^n state dfa (upper bound)

  For each integer n there exists a language accepted by an n-state nfa such that the minimun equivalent dfa requires 2^n states (lower bound)

  Hence:

  The cost, in terms of states, of the simulation of nfa's by dfa's is 2^n.

Classical example: deterministic vs. nondeterministic automata

- Formal language point of view:

  nondeterministic finite automata are as powerful as deterministic finite automata

- Descriptional complexity point of view:

  Each n-state nfa can be simulated by a 2^n state dfa (upper bound)

  For each integer n there exists a language accepted by an n-state nfa such that the minimun equivalent dfa requires 2^n states (lower bound)

  Hence:

  The cost, in terms of states, of the simulation of nfa's by dfa's is 2^n.

Descriptional complexity

Classical example: deterministic vs. nondeterministic automata

- Formal language point of view:

  nondeterministic finite automata are as powerful as deterministic finite automata

- Descriptional complexity point of view:

  Each n-state nfa can be simulated by a 2^n state dfa (upper bound)

  For each integer n there exists a language accepted by an n-state nfa such that the minimun equivalent dfa requires 2^n states (lower bound)

  Hence:

  The cost, in terms of states, of the simulation of nfa's by dfa's is 2^n.

Descriptional complexity of regular languages

- Different variant of finite automata (one-way/two-way, deterministic/nondeterministic/alternating, ...).
- All of them characterize regular languages
- Many results in the literature compare these models from the descriptional point of view.
- However, we can describe regular languages using more powerful devices or formalisms, as context-free grammars and pushdown automata.

What about the sizes of cfg's describing regular languages vs the sizes of finite automata?

- Different variant of finite automata (one-way/two-way, deterministic/nondeterministic/alternating, ...).
- All of them characterize regular languages
- Many results in the literature compare these models from the descriptional point of view.
- However, we can describe regular languages using more powerful devices or formalisms, as context-free grammars and pushdown automata.

What about the sizes of cfg's describing regular languages vs the sizes of finite automata?

Descriptional complexity of regular languages

- Different variant of finite automata (one-way/two-way, deterministic/nondeterministic/alternating, ...).
- All of them characterize regular languages
- Many results in the literature compare these models from the descriptional point of view.
- However, we can describe regular languages using more powerful devices or formalisms, as context-free grammars and pushdown automata.

What about the sizes of cfg's describing regular languages vs the sizes of finite automata?

Descriptional complexity of regular languages

- Different variant of finite automata (one-way/two-way, deterministic/nondeterministic/alternating, ...).
- All of them characterize regular languages
- Many results in the literature compare these models from the descriptional point of view.
- However, we can describe regular languages using more powerful devices or formalisms, as context-free grammars and pushdown automata.

What about the sizes of cfg's describing regular languages vs the sizes of finite automata?

Descriptional complexity of regular languages

- Different variant of finite automata (one-way/two-way, deterministic/nondeterministic/alternating, ...).
- All of them characterize regular languages
- Many results in the literature compare these models from the descriptional point of view.
- However, we can describe regular languages using more powerful devices or formalisms, as context-free grammars and pushdown automata.

What about the sizes of cfg's describing regular languages vs the sizes of finite automata?

Context-free vs regular: descriptional complexity

Given a context-free grammar (or a pushdown automaton) of size n, generating a regular language, how much is big an equivalent finite automaton, wrt n ?

Theorem ([Meyer and Fischer, 1971])

For any recursive function f and arbitrarily large integers n, there exists a cfg G of size n generating a regular language L, s.t. any dfa accepting L must have at least $f(n)$ states.

As a consequence, the trade-off between context-grammars and finite automata is not recursive.

However... The witness language L is defined over a binary alphabet

Context-free vs regular: descriptional complexity

Given a context-free grammar (or a pushdown automaton) of size n, generating a regular language, how much is big an equivalent finite automaton, wrt n ?

Theorem ([Meyer and Fischer, 1971])

For any recursive function f and arbitrarily large integers n, there exists a cfg G of size n generating a regular language L, s.t. any dfa accepting L must have at least $f(n)$ states.

As a consequence, the trade-off between context-grammars and finite automata is not recursive.

However... The witness language L is defined over a binary alphabet.

What about languages over a one-letter alphabet?

Context-free vs regular: descriptional complexity

Given a context-free grammar (or a pushdown automaton) of size n, generating a regular language, how much is big an equivalent finite automaton, wrt n ?

Theorem ([Meyer and Fischer, 1971])

For any recursive function f and arbitrarily large integers n, there exists a cfg G of size n generating a regular language L, s.t. any dfa accepting L must have at least $f(n)$ states.

As a consequence, the trade-off between context-grammars and finite automata is not recursive.
However... The witness language L is defined over a binary alphabet.

What about languages over a one letter alphabet?

Context-free vs regular: descriptional complexity

Given a context-free grammar (or a pushdown automaton) of size n, generating a regular language, how much is big an equivalent finite automaton, wrt n ?

Theorem ([Meyer and Fischer, 1971])

For any recursive function f and arbitrarily large integers n, there exists a cfg G of size n generating a regular language L, s.t. any dfa accepting L must have at least $f(n)$ states.

As a consequence, the trade-off between context-grammars and finite automata is not recursive.
However... The witness language L is defined over a binary alphabet.

What about languages over a one letter alphabet?

Context-free vs regular: descriptional complexity

Given a context-free grammar (or a pushdown automaton) of size n, generating a regular language, how much is big an equivalent finite automaton, wrt n ?

Theorem ([Meyer and Fischer, 1971])

For any recursive function f and arbitrarily large integers n, there exists a cfg G of size n generating a regular language L, s.t. any dfa accepting L must have at least $f(n)$ states.

As a consequence, the trade-off between context-grammars and finite automata is not recursive.
However... The witness language L is defined over a binary alphabet.

What about languages over a one letter alphabet?

Unary languages

$\Sigma = \{a\}$

Theorem ([Ginsurg and Rice, 1962])

Every unary context-free language is regular.

Hence the classes of unary regular languages and unary *context-free* languages coincide!

Problem

Study the equivalence between unary context-free and regular languages from the descriptional complexity point of view.

$\Sigma = \{a\}$

Theorem ([Ginsurg and Rice, 1962])

Every unary context-free language is regular.

Hence the classes of unary regular languages and unary *context-free* languages coincide!

Problem

Study the equivalence between unary context-free and regular languages from the descriptional complexity point of view.

Descriptional complexity measures

- Deterministic automata (dfa):
  number of states.
- Nondeterministic automata (nfa):
  number of states, or
  number of transitions (more precise).

Descriptional complexity measures

- Deterministic automata (dfa):
  number of states.
- Nondeterministic automata (nfa):
  number of states, or
  number of transitions (more precise).

Descriptional complexity measures

- Deterministic automata (dfa):
  number of states.
- Nondeterministic automata (nfa):
  number of states, or
  number of transitions (more precise).

Descriptional complexity measures

- Deterministic automata (dfa):
  number of states.
- Nondeterministic automata (nfa):
  number of states, or
  number of transitions (more precise).

Descriptional complexity measures

- Deterministic automata (dfa):
  number of states.
- Nondeterministic automata (nfa):
  number of states, or
  number of transitions (more precise).

Descriptional complexity measures

- Deterministic automata (dfa):
  number of states.
- Nondeterministic automata (nfa):
  number of states, or
  number of transitions (more precise).

Descriptional complexity measures

- Context-free grammars:
  number of variables?

  For $n \geq 1$, consider the language $L_n = (a^n)^*$:
  - L_n requires n states to be accepted by dfa or nfa
  - L_n is generated by the grammar with one variable S and the productions

$$S \to a^n \qquad S \to a^n S \qquad S \to \epsilon$$

Thus, the number of variables cannot be a descriptional complexity measure for context-free grammars.
However, for grammars in *Chomsky Normal Form* the number of variables is a "reasonable" measure of complexity [Gruska, 1973].

Descriptional complexity measures

- Context-free grammars:
  number of variables?

  For $n \geq 1$, consider the language $L_n = (a^n)^*$:
  - L_n requires n states to be accepted by dfa or nfa
  - L_n is generated by the grammar with one variable S and the productions

$$S \to a^n \qquad S \to a^n S \qquad S \to \epsilon$$

  Thus, the number of variables cannot be a descriptional complexity measure for context-free grammars.
  However, for grammars in *Chomsky Normal Form* the number of variables is a "reasonable" measure of complexity [Gruska, 1973].

Descriptional complexity measures

- Context-free grammars:
  number of variables?

For $n \geq 1$, consider the language $L_n = (a^n)^*$:

- L_n requires n states to be accepted by dfa or nfa
- L_n is generated by the grammar with one variable S and the productions

$$S \rightarrow a^n \qquad S \rightarrow a^n S \qquad S \rightarrow \epsilon$$

Thus, the number of variables cannot be a descriptional complexity measure for context-free grammars.
However, for grammars in *Chomsky Normal Form* the number of variables is a "reasonable" measure of complexity [Gruska, 1973].

- Context-free grammars:
  number of variables?

  For $n \geq 1$, consider the language $L_n = (a^n)^*$:

  - L_n requires n states to be accepted by dfa or nfa
  - L_n is generated by the grammar with one variable S and the productions

$$S \to a^n \qquad S \to a^n S \qquad S \to \epsilon$$

Thus, the number of variables cannot be a descriptional complexity measure for context-free grammars.
However, for grammars in *Chomsky Normal Form* the number of variables is a “reasonable” measure of complexity [Gruska, 1973].

Descriptional complexity measures

- Context-free grammars:
  number of variables?

  For $n \geq 1$, consider the language $L_n = (a^n)^*$:
  - L_n requires n states to be accepted by dfa or nfa
  - L_n is generated by the grammar with one variable S and the productions

$$S \to a^n \qquad S \to a^n S \qquad S \to \epsilon$$

Thus, the number of variables cannot be a descriptional complexity measure for context-free grammars.
However, for grammars in *Chomsky Normal Form* the number of variables is a “reasonable” measure of complexity [Gruska, 1973].

Descriptional complexity measures

- Context-free grammars:
  number of variables?

  For $n \geq 1$, consider the language $L_n = (a^n)^*$:
  - L_n requires n states to be accepted by dfa or nfa
  - L_n is generated by the grammar with one variable S and the productions

$$S \to a^n \qquad S \to a^n S \qquad S \to \epsilon$$

Thus, the number of variables cannot be a descriptional complexity measure for context-free grammars.
However, for grammars in *Chomsky Normal Form* the number of variables is a “reasonable” measure of complexity [Gruska, 1973].

Descriptional complexity measures

- Context-free grammars:
  number of variables?

  For $n \geq 1$, consider the language $L_n = (a^n)^*$:
  - L_n requires n states to be accepted by dfa or nfa
  - L_n is generated by the grammar with one variable S and the productions

$$S \to a^n \qquad S \to a^n S \qquad S \to \epsilon$$

  Thus, the number of variables cannot be a descriptional complexity measure for context-free grammars.
  However, for grammars in *Chomsky Normal Form* the number of variables is a "reasonable" measure of complexity [Gruska, 1973].

- Context-free grammars:
  number of variables?

  For $n \geq 1$, consider the language $L_n = (a^n)^*$:
  - L_n requires n states to be accepted by dfa or nfa
  - L_n is generated by the grammar with one variable S and the productions

$$S \to a^n \qquad S \to a^n S \qquad S \to \epsilon$$

Thus, the number of variables cannot be a descriptional complexity measure for context-free grammars.
However, for grammars in *Chomsky Normal Form* the number of variables is a "reasonable" measure of complexity [Gruska, 1973].

Descriptional complexity measures

- Pushdown automata:
  W.l.o.g., we can consider only pda's s.t. push operations add exactly one symbol on the pushdown store.

  The total size of the description of a pda satisfying this restriction is a polynomial function of two parameters:
  - the number of the states
  - the cardinality of the pushdown alphabet.

  Strong relationships have been discovered between descriptional complexities of cfg's and pda's [Goldstine, Price, Wotschke, 1982], e.g.:

Theorem

For any pda satisfying the above restriction, with n states and m pushdown symbols, there exists an equivalent cfg in Chomsky normal form, with $n^2m + 1$ variables.

Descriptional complexity measures

- Pushdown automata:
  W.l.o.g., we can consider only pda's s.t. push operations add exactly one symbol on the pushdown store.

  The total size of the description of a pda satisfying this restriction is a polynomial function of two parameters:
  - the number of the states
  - the cardinality of the pushdown alphabet.

  Strong relationships have been discovered between descriptional complexities of cfg's and pda's [Goldstine, Price, Wotschke, 1982], e.g.:

Theorem

For any pda satisfying the above restriction, with n states and m pushdown symbols, there exists an equivalent cfg in Chomsky normal form, with n^2m+1 variables.

- Pushdown automata:
  W.l.o.g., we can consider only pda's s.t. push operations add exactly one symbol on the pushdown store.

  The total size of the description of a pda satisfying this restriction is a polynomial function of two parameters:
  - the number of the states
  - the cardinality of the pushdown alphabet.

  Strong relationships have been discovered between descriptional complexities of cfg's and pda's [Goldstine, Price, Wotschke, 1982], e.g.:

Theorem

For any pda satisfying the above restriction, with n states and m pushdown symbols, there exists an equivalent cfg in Chomsky normal form, with $n^2 m + 1$ variables.

Descriptional complexity measures

- Pushdown automata:
  W.l.o.g., we can consider only pda's s.t. push operations add exactly one symbol on the pushdown store.

  The total size of the description of a pda satisfying this restriction is a polynomial function of two parameters:
  - the number of the states
  - the cardinality of the pushdown alphabet.

  Strong relationships have been discovered between descriptional complexities of cfg's and pda's [Goldstine, Price, Wotschke, 1982], e.g.:

Theorem

For any pda satisfying the above restriction, with n states and m pushdown symbols, there exists an equivalent cfg in Chomsky normal form, with n^2m+1 variables.

Notations:

- $G = (V, \{a\}, P, S)$
  a cfg grammar in Chomsky normal form
- h
  the number of variables in G
- $T : A \stackrel{\star}{\Rightarrow} \alpha$
  a *parse tree* for the derivation $A \stackrel{\star}{\Rightarrow} \alpha$, with $A \in V$, $\alpha \in (V \cup \{a\})^*$.
- $\nu(T)$
  the set of variables which appear as labels of some nodes in T

Unary cfg $\longrightarrow$ nfa

Notations:

- $G = (V, \{a\}, P, S)$
  a cfg grammar in Chomsky normal form
- h
  the number of variables in G
- $T : A \stackrel{\star}{\Rightarrow} \alpha$
  a *parse tree* for the derivation $A \stackrel{\star}{\Rightarrow} \alpha$, with $A \in V$, $\alpha \in (V \cup \{a\})^*$.
- $\nu(T)$
  the set of variables which appear as labels of some nodes in T

Unary cfg $\longrightarrow$ nfa

Notations:

- $G = (V, \{a\}, P, S)$
  a cfg grammar in Chomsky normal form
- h
  the number of variables in G
- $T : A \stackrel{\star}{\Rightarrow} \alpha$
  a *parse tree* for the derivation $A \stackrel{\star}{\Rightarrow} \alpha$, with $A \in V$, $\alpha \in (V \cup \{a\})^*$.
- $\nu(T)$
  the set of variables which appear as labels of some nodes in T

Unary cfg $\longrightarrow$ nfa

Notations:

- $G = (V, \{a\}, P, S)$
  a cfg grammar in Chomsky normal form
- h
  the number of variables in G
- $T : A \stackrel{\star}{\Rightarrow} \alpha$
  a *parse tree* for the derivation $A \stackrel{\star}{\Rightarrow} \alpha$, with $A \in V$, $\alpha \in (V \cup \{a\})^*$.
- $\nu(T)$
  the set of variables which appear as labels of some nodes in T

Unary cfg $\longrightarrow$ nfa

Notations:

- $G = (V, \{a\}, P, S)$
  a cfg grammar in Chomsky normal form
- h
  the number of variables in G
- $T : A \stackrel{\star}{\Rightarrow} \alpha$
  a *parse tree* for the derivation $A \stackrel{\star}{\Rightarrow} \alpha$, with $A \in V$, $\alpha \in (V \cup \{a\})^*$.
- $\nu(T)$
  the set of variables which appear as labels of some nodes in T

Unary cfg $\longrightarrow$ nfa

Lemma (padding)

Given

- $T : S \stackrel{\star}{\Rightarrow} a^\ell$
- $T' : A \stackrel{+}{\Rightarrow} a^i A a^j$, *with* $A \in \nu(T)$

there exists

- $T'' : S \stackrel{\star}{\Rightarrow} a^{\ell+i+j}$, *with* $\nu(T'') = \nu(T) \cup \nu(T')$

i.e., T can be "padded" with T' in order to get T''.

Unary cfg $\longrightarrow$ nfa

Lemma (padding)

Given

- $T : S \stackrel{\star}{\Rightarrow} a^\ell$
- $T' : A \stackrel{+}{\Rightarrow} a^i A a^j$, *with* $A \in \nu(T)$

there exists

- $T'' : S \stackrel{\star}{\Rightarrow} a^{\ell+i+j}$, *with* $\nu(T'') = \nu(T) \cup \nu(T')$

i.e., T can be “padded” with T' in order to get T''.

Unary cfg $\longrightarrow$ nfa

Lemma (padding)

Given

- $T : S \overset{\star}{\Rightarrow} a^{\ell}$
- $T' : A \overset{+}{\Rightarrow} a^i A a^j$, *with* $A \in \nu(T)$

there exists

- $T'' : S \overset{\star}{\Rightarrow} a^{\ell+i+j}$, *with* $\nu(T'') = \nu(T) \cup \nu(T')$

i.e., T can be "padded" with T' in order to get T''.

Unary cfg $\longrightarrow$ nfa

Lemma (decomposition)

If $T : S \overset{\star}{\Rightarrow} a^\ell$ *and* $\ell > 2^{h-1}$*, then there exist:*

- *a tree* $T_1 : S \overset{\star}{\Rightarrow} a^s$
- *a tree* $T_2 : A \overset{+}{\Rightarrow} a^i A a^j$*, with* $A \in \nu(T_1)$

such that:

- $\nu(T) = \nu(T_1) \cup \nu(T_2)$
- $\ell = s + i + j$*,* $s > 0$ *and* $0 < i + j < 2^h$*.*

In other words, a tree $T : S \overset{\star}{\Rightarrow} a^\ell$ of a "long" string ($\ell > 2^{h-1}$) can be obtained by padding

- a tree $T_1 : S \overset{\star}{\Rightarrow} a^s$ of shortest string
- with a "small" tree $T_2 : A \overset{+}{\Rightarrow} a^i A a^j$, where $A \in \nu(T_1)$ ("small" means $0 < i + j < 2^h$)

Notice that if $s > 2^{h-1}$ then the tree T_1 can be obtained in a similar way.

Unary cfg $\longrightarrow$ nfa

Lemma (decomposition)

If $T : S \stackrel{\star}{\Rightarrow} a^\ell$ and $\ell > 2^{h-1}$, then there exist:

- *a tree $T_1 : S \stackrel{\star}{\Rightarrow} a^s$*
- *a tree $T_2 : A \stackrel{+}{\Rightarrow} a^i A a^j$, with $A \in \nu(T_1)$*

such that:

- $\nu(T) = \nu(T_1) \cup \nu(T_2)$
- $\ell = s + i + j$, $s > 0$ *and* $0 < i + j < 2^h$.

In other words, a tree $T : S \stackrel{\star}{\Rightarrow} a^\ell$ of a “long” string ($\ell > 2^{h-1}$) can be obtained by padding

- a tree $T_1 : S \stackrel{\star}{\Rightarrow} a^s$ of shortest string
- with a “small” tree $T_2 : A \stackrel{+}{\Rightarrow} a^i A a^j$, where $A \in \nu(T_1)$ (“small” means $0 < i + j < 2^h$)

Notice that if $s > 2^{h-1}$ then the tree T_1 can be obtained in a similar way.

Unary cfg $\longrightarrow$ nfa

Lemma (decomposition)

If $T : S \overset{\star}{\Rightarrow} a^\ell$ *and* $\ell > 2^{h-1}$*, then there exist:*

- *a tree* $T_1 : S \overset{\star}{\Rightarrow} a^s$
- *a tree* $T_2 : A \overset{+}{\Rightarrow} a^i A a^j$*, with* $A \in \nu(T_1)$

such that:

- $\nu(T) = \nu(T_1) \cup \nu(T_2)$
- $\ell = s + i + j$*,* $s > 0$ *and* $0 < i + j < 2^h$*.*

In other words, a tree $T : S \overset{\star}{\Rightarrow} a^\ell$ of a "long" string ($\ell > 2^{h-1}$) can be obtained by padding

- a tree $T_1 : S \overset{\star}{\Rightarrow} a^s$ of shortest string
- with a "small" tree $T_2 : A \overset{+}{\Rightarrow} a^i A a^j$, where $A \in \nu(T_1)$ ("small" means $0 < i + j < 2^h$)

Notice that if $s > 2^{h-1}$ then the tree T_1 can be obtained in a similar way.

Unary cfg $\longrightarrow$ nfa

Lemma (decomposition)

If $T : S \stackrel{\star}{\Rightarrow} a^\ell$ and $\ell > 2^{h-1}$, then there exist:

- *a tree $T_1 : S \stackrel{\star}{\Rightarrow} a^s$*
- *a tree $T_2 : A \stackrel{+}{\Rightarrow} a^i A a^j$, with $A \in \nu(T_1)$*

such that:

- $\nu(T) = \nu(T_1) \cup \nu(T_2)$
- *$\ell = s + i + j$, $s > 0$ and $0 < i + j < 2^h$.*

In other words, a tree $T : S \stackrel{\star}{\Rightarrow} a^\ell$ of a “long” string ($\ell > 2^{h-1}$) can be obtained by padding

- a tree $T_1 : S \stackrel{\star}{\Rightarrow} a^s$ of shortest string
- with a “small” tree $T_2 : A \stackrel{+}{\Rightarrow} a^i A a^j$, where $A \in \nu(T_1)$ (“small” means $0 < i + j < 2^h$)

Notice that if $s > 2^{h-1}$ then the tree T_1 can be obtained in a similar way.

Unary cfg $\longrightarrow$ nfa

Lemma (decomposition)

If $T : S \stackrel{\star}{\Rightarrow} a^\ell$ *and* $\ell > 2^{h-1}$*, then there exist:*

- *a tree* $T_1 : S \stackrel{\star}{\Rightarrow} a^s$
- *a tree* $T_2 : A \stackrel{+}{\Rightarrow} a^i A a^j$*, with* $A \in \nu(T_1)$

such that:

- $\nu(T) = \nu(T_1) \cup \nu(T_2)$
- $\ell = s + i + j$*,* $s > 0$ *and* $0 < i + j < 2^h$*.*

In other words, a tree $T : S \stackrel{\star}{\Rightarrow} a^\ell$ of a “long” string ($\ell > 2^{h-1}$) can be obtained by padding

- a tree $T_1 : S \stackrel{\star}{\Rightarrow} a^s$ of shortest string
- with a “small” tree $T_2 : A \stackrel{+}{\Rightarrow} a^i A a^j$, where $A \in \nu(T_1)$ (“small” means $0 < i + j < 2^h$)

Notice that if $s > 2^{h-1}$ then the tree T_1 can be obtained in a similar way.

Unary cfg $\longrightarrow$ nfa

Lemma (decomposition)

If $T : S \overset{\star}{\Rightarrow} a^\ell$ *and* $\ell > 2^{h-1}$*, then there exist:*

- *a tree* $T_1 : S \overset{\star}{\Rightarrow} a^s$
- *a tree* $T_2 : A \overset{+}{\Rightarrow} a^i A a^j$*, with* $A \in \nu(T_1)$

such that:

- $\nu(T) = \nu(T_1) \cup \nu(T_2)$
- $\ell = s + i + j$*,* $s > 0$ *and* $0 < i + j < 2^h$*.*

In other words, a tree $T : S \overset{\star}{\Rightarrow} a^\ell$ of a "long" string ($\ell > 2^{h-1}$) can be obtained by padding

- a tree $T_1 : S \overset{\star}{\Rightarrow} a^s$ of shortest string
- with a "small" tree $T_2 : A \overset{+}{\Rightarrow} a^i A a^j$, where $A \in \nu(T_1)$ ("small" means $0 < i + j < 2^h$)

Notice that if $s > 2^{h-1}$ then the tree T_1 can be obtained in a similar way.

Lemma (decomposition)

If $T : S \overset{\star}{\Rightarrow} a^\ell$ *and* $\ell > 2^{h-1}$*, then there exist:*

- *a tree* $T_1 : S \overset{\star}{\Rightarrow} a^s$
- *a tree* $T_2 : A \overset{+}{\Rightarrow} a^i A a^j$*, with* $A \in \nu(T_1)$

such that:

- $\nu(T) = \nu(T_1) \cup \nu(T_2)$
- $\ell = s + i + j$, $s > 0$ *and* $0 < i + j < 2^h$.

In other words, a tree $T : S \overset{\star}{\Rightarrow} a^\ell$ of a “long” string ($\ell > 2^{h-1}$) can be obtained by padding

- a tree $T_1 : S \overset{\star}{\Rightarrow} a^s$ of shortest string
- with a “small” tree $T_2 : A \overset{+}{\Rightarrow} a^i A a^j$, where $A \in \nu(T_1)$ (“small” means $0 < i + j < 2^h$)

Notice that if $s > 2^{h-1}$ then the tree T_1 can be obtained in a similar way.

Unary cfg $\longrightarrow$ nfa

Lemma (decomposition)

If $T : S \overset{\star}{\Rightarrow} a^\ell$ *and* $\ell > 2^{h-1}$*, then there exist:*

- *a tree* $T_1 : S \overset{\star}{\Rightarrow} a^s$
- *a tree* $T_2 : A \overset{+}{\Rightarrow} a^i A a^j$*, with* $A \in \nu(T_1)$

such that:

- $\nu(T) = \nu(T_1) \cup \nu(T_2)$
- $\ell = s + i + j$, $s > 0$ *and* $0 < i + j < 2^h$.

In other words, a tree $T : S \overset{\star}{\Rightarrow} a^\ell$ of a "long" string ($\ell > 2^{h-1}$) can be obtained by padding

- a tree $T_1 : S \overset{\star}{\Rightarrow} a^s$ of shortest string
- with a "small" tree $T_2 : A \overset{+}{\Rightarrow} a^i A a^j$, where $A \in \nu(T_1)$ ("small" means $0 < i + j < 2^h$)

Notice that if $s > 2^{h-1}$ then the tree T_1 can be obtained in a similar way.

Unary cfg $\longrightarrow$ nfa

Lemma (decomposition)

If $T : S \overset{\star}{\Rightarrow} a^\ell$ *and* $\ell > 2^{h-1}$*, then there exist:*

- *a tree* $T_1 : S \overset{\star}{\Rightarrow} a^s$
- *a tree* $T_2 : A \overset{+}{\Rightarrow} a^i A a^j$*, with* $A \in \nu(T_1)$

such that:

- $\nu(T) = \nu(T_1) \cup \nu(T_2)$
- $\ell = s + i + j$*,* $s > 0$ *and* $0 < i + j < 2^h$*.*

In other words, a tree $T : S \overset{\star}{\Rightarrow} a^\ell$ of a "long" string ($\ell > 2^{h-1}$) can be obtained by padding

- a tree $T_1 : S \overset{\star}{\Rightarrow} a^s$ of shortest string
- with a "small" tree $T_2 : A \overset{+}{\Rightarrow} a^i A a^j$, where $A \in \nu(T_1)$ ("small" means $0 < i + j < 2^h$)

Notice that if $s > 2^{h-1}$ then the tree T_1 can be obtained in a similar way.

Unary cfg $\longrightarrow$ nfa

Lemma (decomposition)

If $T : S \overset{\star}{\Rightarrow} a^\ell$ *and* $\ell > 2^{h-1}$*, then there exist:*

- *a tree* $T_1 : S \overset{\star}{\Rightarrow} a^s$
- *a tree* $T_2 : A \overset{+}{\Rightarrow} a^i A a^j$*, with* $A \in \nu(T_1)$

such that:

- $\nu(T) = \nu(T_1) \cup \nu(T_2)$
- $\ell = s + i + j$, $s > 0$ *and* $0 < i + j < 2^h$.

In other words, a tree $T : S \overset{\star}{\Rightarrow} a^\ell$ of a “long” string ($\ell > 2^{h-1}$) can be obtained by padding

- a tree $T_1 : S \overset{\star}{\Rightarrow} a^s$ of shortest string
- with a “small” tree $T_2 : A \overset{+}{\Rightarrow} a^i A a^j$, where $A \in \nu(T_1)$ (“small” means $0 < i + j < 2^h$)

Notice that if $s > 2^{h-1}$ then the tree T_1 can be obtained in a similar way.

Unary cfg $\rightarrow$ nfa

MAIN IDEA:

we can generate all strings belonging to $L(G)$ by using “small” trees corresponding to derivations of the forms

- $S \overset{\star}{\Rightarrow} a^{\ell}$ and
- $A \overset{+}{\Rightarrow} a^i A a^j$,

where $\ell \leq 2^{h-1}$ and $0 < i + j < 2^h$.

Generation procedure for $L(G)$

nondeterministically select a tree $T_1 : S \overset{\star}{\Rightarrow} a^\ell$, with $\ell \leq 2^{h-1}$
enabled $\leftarrow \nu(T_1)$
iterate $\leftarrow$ nondeterministically choose *true* or *false*
while *iterate* **do**
    nondeterministically select a tree $T_2 : A \overset{+}{\Rightarrow} a^i A a^j$,
        with $0 < i + j < 2^h$ and $A \in$*enabled*
    $\ell \leftarrow \ell + i + j$
    enabled $\leftarrow$*enabled* $\cup \nu(T_2)$
    iterate $\leftarrow$ nondeterministically choose *true* or *false*
endwhile
output a^ℓ

nondeterministically select a tree $T_1 : S \stackrel{\star}{\Rightarrow} a^\ell$, with $\ell \leq 2^{h-1}$
enabled $\leftarrow \nu(T_1)$
iterate $\leftarrow$ nondeterministically choose *true* or *false*
while *iterate* **do**
    nondeterministically select a tree $T_2 : A \stackrel{+}{\Rightarrow} a^i A a^j$,
        with $0 < i + j < 2^h$ and $A \in$*enabled*
    $\ell \leftarrow \ell + i + j$
    enabled $\leftarrow$*enabled* $\cup \nu(T_2)$
    iterate $\leftarrow$ nondeterministically choose *true* or *false*
endwhile
output a^ℓ

Generation procedure for $L(G)$

nondeterministically select a tree $T_1 : S \overset{\star}{\Rightarrow} a^\ell$, with $\ell \leq 2^{h-1}$

enabled $\leftarrow \nu(T_1)$

iterate $\leftarrow$ nondeterministically choose *true* or *false*

while *iterate* **do**

  nondeterministically select a tree $T_2 : A \overset{+}{\Rightarrow} a^i A a^j$,
    with $0 < i + j < 2^h$ and $A \in$ *enabled*

  $\ell \leftarrow \ell + i + j$

  enabled $\leftarrow$ *enabled* $\cup \nu(T_2)$

  iterate $\leftarrow$ nondeterministically choose *true* or *false*

endwhile

output a^ℓ

Generation procedure for $L(G)$

nondeterministically select a tree $T_1 : S \overset{\star}{\Rightarrow} a^\ell$, with $\ell \leq 2^{h-1}$
enabled $\leftarrow \nu(T_1)$
iterate $\leftarrow$ nondeterministically choose *true* or *false*
while *iterate* **do**
    nondeterministically select a tree $T_2 : A \overset{+}{\Rightarrow} a^i A a^j$,
        with $0 < i + j < 2^h$ and $A \in$*enabled*
    $\ell \leftarrow \ell + i + j$
    enabled $\leftarrow$*enabled* $\cup \nu(T_2)$
    iterate $\leftarrow$ nondeterministically choose *true* or *false*
endwhile
output a^ℓ

Generation procedure for $L(G)$

nondeterministically select a tree $T_1 : S \overset{\star}{\Rightarrow} a^\ell$, with $\ell \leq 2^{h-1}$
enabled $\leftarrow \nu(T_1)$
iterate $\leftarrow$ nondeterministically choose *true* or *false*
while *iterate* **do**

nondeterministically select a tree $T_2 : A \overset{+}{\Rightarrow} a^i A a^j$,
with $0 < i + j < 2^h$ and $A \in$*enabled*

$\ell \leftarrow \ell + i + j$
enabled $\leftarrow$*enabled* $\cup \nu(T_2)$
iterate $\leftarrow$ nondeterministically choose *true* or *false*

endwhile
output a^ℓ

Generation procedure for $L(G)$

nondeterministically select a tree $T_1 : S \overset{\star}{\Rightarrow} a^\ell$, with $\ell \leq 2^{h-1}$
enabled $\leftarrow \nu(T_1)$
iterate $\leftarrow$ nondeterministically choose *true* or *false*
while *iterate* **do**

nondeterministically select a tree $T_2 : A \overset{+}{\Rightarrow} a^i A a^j$,
with $0 < i + j < 2^h$ and $A \in$*enabled*

$\ell \leftarrow \ell + i + j$

enabled $\leftarrow$*enabled* $\cup \nu(T_2)$

iterate $\leftarrow$ nondeterministically choose *true* or *false*

endwhile
output a^ℓ

Generation procedure for $L(G)$

nondeterministically select a tree $T_1 : S \overset{\star}{\Rightarrow} a^\ell$, with $\ell \leq 2^{h-1}$
enabled $\leftarrow \nu(T_1)$
iterate $\leftarrow$ nondeterministically choose *true* or *false*
while *iterate* **do**

  nondeterministically select a tree $T_2 : A \overset{+}{\Rightarrow} a^i A a^j$,
    with $0 < i + j < 2^h$ and $A \in$ *enabled*
  $\ell \leftarrow \ell + i + j$
  enabled $\leftarrow$ *enabled* $\cup \nu(T_2)$
  iterate $\leftarrow$ nondeterministically choose *true* or *false*

endwhile
output a^ℓ

Generation procedure for $L(G)$

nondeterministically select a tree $T_1 : S \stackrel{\star}{\Rightarrow} a^\ell$, with $\ell \leq 2^{h-1}$
enabled $\leftarrow \nu(T_1)$
iterate $\leftarrow$ nondeterministically choose *true* or *false*
while *iterate* **do**
  nondeterministically select a tree $T_2 : A \stackrel{+}{\Rightarrow} a^i A a^j$,
    with $0 < i + j < 2^h$ and $A \in$*enabled*
  $\ell \leftarrow \ell + i + j$
  enabled $\leftarrow$*enabled* $\cup \nu(T_2)$
  iterate $\leftarrow$ nondeterministically choose *true* or *false*
endwhile
output a^ℓ

Unary cfg $\to$ nfa

- The strategy of the generation procedure can be implemented by a nfa with ϵ-moves A.
- The states of A are pairs (α, s), where:
  - $\alpha \subseteq V$ represents the variable *enabled*
  - $s < 2^h$ is used to count input factors
- After some simplifications, the number of the states of such an automaton can be reduced to $2^{2h-1} + 1$.

Hence

Theorem (upper bound)

For any unary cfg in Chomsky normal form with h variables, there exists an equivalent nfa with at most $2^{2h-1} + 1$ states.

Unary cfg → nfa

- The strategy of the generation procedure can be implemented by a nfa with ϵ-moves A.
- The states of A are pairs (α, s), where:
  - $\alpha \subseteq V$ represents the variable *enabled*
  - $s < 2^h$ is used to count input factors
- After some simplifications, the number of the states of such an automaton can be reduced to $2^{2h-1} + 1$.

Hence

Theorem (upper bound)

For any unary cfg in Chomsky normal form with h variables, there exists an equivalent nfa with at most $2^{2h-1} + 1$ states.

Unary cfg $\to$ nfa

- The strategy of the generation procedure can be implemented by a nfa with ϵ-moves A.
- The states of A are pairs (α, s), where:
  - $\alpha \subseteq V$ represents the variable *enabled*
  - $s < 2^h$ is used to count input factors
- After some simplifications, the number of the states of such an automaton can be reduced to $2^{2h-1} + 1$.

Hence

Theorem (upper bound)

For any unary cfg in Chomsky normal form with h variables, there exists an equivalent nfa with at most $2^{2h-1} + 1$ states.

Unary cfg $\longrightarrow$ nfa

- The strategy of the generation procedure can be implemented by a nfa with ϵ-moves A.
- The states of A are pairs (α, s), where:
  - $\alpha \subseteq V$ represents the variable *enabled*
  - $s < 2^h$ is used to count input factors
- After some simplifications, the number of the states of such an automaton can be reduced to $2^{2h-1} + 1$.

Hence

Theorem (upper bound)

For any unary cfg in Chomsky normal form with h variables, there exists an equivalent nfa with at most $2^{2h-1} + 1$ states.

- The strategy of the generation procedure can be implemented by a nfa with ϵ-moves A.
- The states of A are pairs (α, s), where:
  - $\alpha \subseteq V$ represents the variable *enabled*
  - $s < 2^h$ is used to count input factors
- After some simplifications, the number of the states of such an automaton can be reduced to $2^{2h-1} + 1$.

Hence

Theorem (upper bound)

For any unary cfg in Chomsky normal form with h variables, there exists an equivalent nfa with at most $2^{2h-1} + 1$ states.

- The strategy of the generation procedure can be implemented by a nfa with ϵ-moves A.
- The states of A are pairs (α, s), where:
  - $\alpha \subseteq V$ represents the variable *enabled*
  - $s < 2^h$ is used to count input factors
- After some simplifications, the number of the states of such an automaton can be reduced to $2^{2h-1} + 1$.

Hence

Theorem (upper bound)

For any unary cfg in Chomsky normal form with h variables, there exists an equivalent nfa with at most $2^{2h-1} + 1$ states.

- The strategy of the generation procedure can be implemented by a nfa with ϵ-moves A.
- The states of A are pairs (α, s), where:
  - $\alpha \subseteq V$ represents the variable *enabled*
  - $s < 2^h$ is used to count input factors
- After some simplifications, the number of the states of such an automaton can be reduced to $2^{2h-1} + 1$.

Hence

Theorem (upper bound)

For any unary cfg in Chomsky normal form with h variables, there exists an equivalent nfa with at most $2^{2h-1} + 1$ states.

Unary cfg → nfa (lower bound)

Theorem (lower bound)

For any integer $h \geq 1$, there exists a unary cfg in Chomsky normal form with h variables, such that any equivalent nfa must have at least $2^{h-1} + 1$ states.

(Sketch of the proof)

For $h > 1$, consider variables $A_0, \ldots, A_{h-1}$ and productions:

- $A_0 \rightarrow a$
- $A_j \rightarrow A_{j-1}A_{j-1}$, for $j = 1, \ldots, h-2$
- $A_{h-1} \rightarrow A_{h-2}A_{h-2} \mid A_{h-1}A_{h-1}$

Then, for $j = 0, \ldots, h-2$: $A_j \overset{\star}{\Rightarrow} a^x$ iff $x = 2^j$.

If A_{h-1} is the start symbol, the language is $(a^{2^{h-1}})^+$

This language needs $2^{h-1} + 1$ states to be accepted by an nfa.

Unary cfg → nfa (lower bound)

Theorem (lower bound)

For any integer $h \geq 1$, there exists a unary cfg in Chomsky normal form with h variables, such that any equivalent nfa must have at least $2^{h-1} + 1$ states.

(Sketch of the proof)

For $h > 1$, consider variables $A_0, \ldots, A_{h-1}$ and productions:

- $A_0 \to a$
- $A_j \to A_{j-1}A_{j-1}$, for $j = 1, \ldots, h-2$
- $A_{h-1} \to A_{h-2}A_{h-2} \mid A_{h-1}A_{h-1}$

Then, for $j = 0, \ldots, h-2$: $A_j \overset{\star}{\Rightarrow} a^x$ iff $x = 2^j$.

If A_{h-1} is the start symbol, the language is $(a^{2^{h-1}})^+$

This language needs $2^{h-1} + 1$ states to be accepted by an nfa.

Unary cfg $\rightarrow$ nfa (lower bound)

Theorem (lower bound)

For any integer $h \geq 1$, there exists a unary cfg in Chomsky normal form with h variables, such that any equivalent nfa must have at least $2^{h-1} + 1$ states.

(Sketch of the proof)

For $h > 1$, consider variables $A_0, \ldots, A_{h-1}$ and productions:

- $A_0 \rightarrow a$
- $A_j \rightarrow A_{j-1}A_{j-1}$, for $j = 1, \ldots, h-2$
- $A_{h-1} \rightarrow A_{h-2}A_{h-2} \mid A_{h-1}A_{h-1}$

Then, for $j = 0, \ldots, h-2$: $A_j \overset{\star}{\Rightarrow} a^x$ iff $x = 2^j$.

If A_{h-1} is the start symbol, the language is $(a^{2^{h-1}})^+$

This language needs $2^{h-1} + 1$ states to be accepted by an nfa.

Unary cfg → nfa (lower bound)

Theorem (lower bound)

For any integer $h \geq 1$, there exists a unary cfg in Chomsky normal form with h variables, such that any equivalent nfa must have at least $2^{h-1} + 1$ states.

(Sketch of the proof)

For $h > 1$, consider variables $A_0, \ldots, A_{h-1}$ and productions:

- $A_0 \to a$
- $A_j \to A_{j-1}A_{j-1}$, for $j = 1, \ldots, h-2$
- $A_{h-1} \to A_{h-2}A_{h-2} \mid A_{h-1}A_{h-1}$

Then, for $j = 0, \ldots, h-2$: $A_j \stackrel{\star}{\Rightarrow} a^x$ iff $x = 2^j$.

If A_{h-1} is the start symbol, the language is $(a^{2^{h-1}})^+$

This language needs $2^{h-1} + 1$ states to be accepted by an nfa.

Unary cfg $\to$ nfa (lower bound)

Theorem (lower bound)

For any integer $h \geq 1$, there exists a unary cfg in Chomsky normal form with h variables, such that any equivalent nfa must have at least $2^{h-1}+1$ states.

(Sketch of the proof)

For $h > 1$, consider variables $A_0, \ldots, A_{h-1}$ and productions:

- $A_0 \to a$
- $A_j \to A_{j-1}A_{j-1}$, for $j = 1, \ldots, h-2$
- $A_{h-1} \to A_{h-2}A_{h-2} \mid A_{h-1}A_{h-1}$

Then, for $j = 0, \ldots, h-2$: $A_j \stackrel{\star}{\Rightarrow} a^x$ iff $x = 2^j$.

If A_{h-1} is the start symbol, the language is $(a^{2^{h-1}})^+$

This language needs $2^{h-1}+1$ states to be accepted by an nfa.

Theorem (lower bound)

For any integer $h \geq 1$, there exists a unary cfg in Chomsky normal form with h variables, such that any equivalent nfa must have at least $2^{h-1} + 1$ states.

(Sketch of the proof)

For $h > 1$, consider variables $A_0, \ldots, A_{h-1}$ and productions:

- $A_0 \to a$
- $A_j \to A_{j-1}A_{j-1}$, for $j = 1, \ldots, h-2$
- $A_{h-1} \to A_{h-2}A_{h-2} \mid A_{h-1}A_{h-1}$

Then, for $j = 0, \ldots, h-2$: $A_j \overset{\star}{\Rightarrow} a^x$ iff $x = 2^j$.

If A_{h-1} is the start symbol, the language is $(a^{2^{h-1}})^+$

This language needs $2^{h-1} + 1$ states to be accepted by an nfa.

Let G be a unary cfg in Chomsky normal form with h variables.

- There exists an equivalent nfa A with at most $2^{2h-1}+1$ states.
- Using the subset construction, we can get an equivalent dfa with $2^{2^{O(h)}}$ states.

...we can do better!

Theorem

$L(G)$ is accepted by a dfa with 2^{h^2} states.

Unary cfg $\rightarrow$ dfa

Let G be a unary cfg in Chomsky normal form with h variables.

- There exists an equivalent nfa A with at most $2^{2h-1}+1$ states.
- Using the subset construction, we can get an equivalent dfa with $2^{2^{O(h)}}$ states.

...we can do better!

Theorem

$L(G)$ is accepted by a dfa with 2^{h^2} states.

Let G be a unary cfg in Chomsky normal form with h variables.

- There exists an equivalent nfa A with at most $2^{2h-1}+1$ states.
- Using the subset construction, we can get an equivalent dfa with $2^{2^{O(h)}}$ states.

...we can do better!

Theorem

$L(G)$ is accepted by a dfa with 2^{h^2} states.

Let G be a unary cfg in Chomsky normal form with h variables.

- There exists an equivalent nfa A with at most $2^{2h-1}+1$ states.
- Using the subset construction, we can get an equivalent dfa with $2^{2^{O(h)}}$ states.

...we can do better!

Theorem

$L(G)$ is accepted by a dfa with 2^{h^2} states.

Unary cfg $\longrightarrow$ dfa

Let G be a unary cfg in Chomsky normal form with h variables.

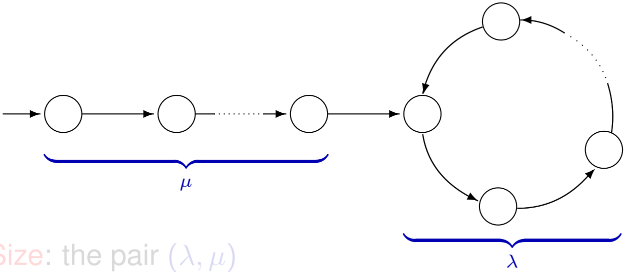
- There exists an equivalent nfa A with at most $2^{2h-1}+1$ states.
- Using the subset construction, we can get an equivalent dfa with $2^{2^{O(h)}}$ states.

...we can do better!

Theorem

$L(G)$ is accepted by a dfa with 2^{h^2} states.

Unary deterministic automata



Size: the pair (λ, μ)

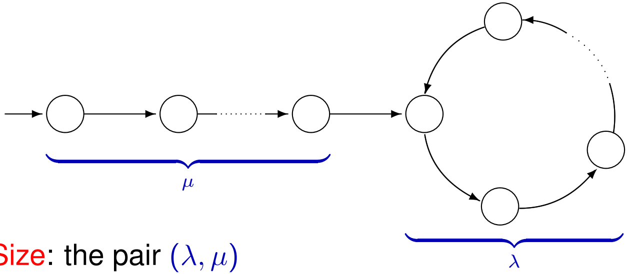
Theorem ([Pighizzini and Shallit, 2002])

- $L_1, \ldots, L_k$ *unary regular languages*
- L_i *accepted by a dfa of size* (λ_i, μ_i), $i = 1, \ldots, k$.

Then $\bigcup_{i=1}^{k} L_i$ *is accepted by a dfa of size*

$$(lcm(\lambda_1, \ldots, \lambda_k), \max(\mu_1, \ldots, \mu_k))$$

Unary deterministic automata



Size: the pair (λ, μ)

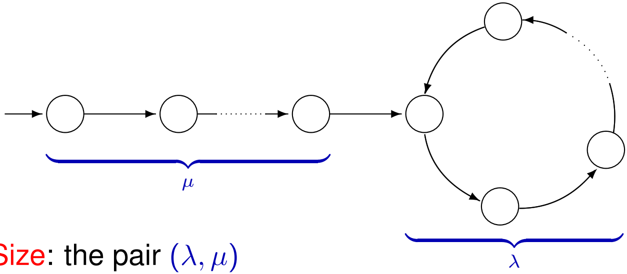
Theorem ([Pighizzini and Shallit, 2002])

- $L_1, \ldots, L_k$ *unary regular languages*
- L_i *accepted by a dfa of size* (λ_i, μ_i), $i = 1, \ldots, k$.

Then $\bigcup_{i=1}^{k} L_i$ *is accepted by a dfa of size*

$$(lcm(\lambda_1, \ldots, \lambda_k), \max(\mu_1, \ldots, \mu_k))$$

Unary deterministic automata



Size: the pair (λ, μ)

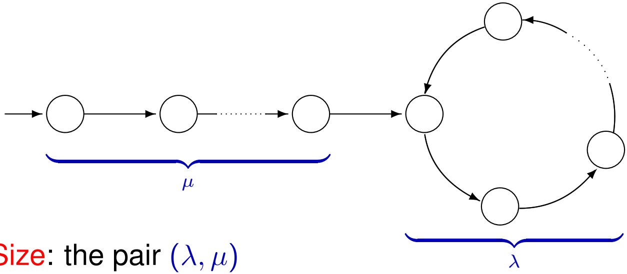
Theorem ([Pighizzini and Shallit, 2002])

- *$L_1, \ldots, L_k$ unary regular languages*
- *L_i accepted by a dfa of size (λ_i, μ_i), $i = 1, \ldots, k$.*

Then $\bigcup_{i=1}^{k} L_i$ is accepted by a dfa of size

$$(lcm(\lambda_1, \ldots, \lambda_k), \max(\mu_1, \ldots, \mu_k))$$

Unary deterministic automata



Size: the pair (λ, μ)

Theorem ([Pighizzini and Shallit, 2002])

- $L_1, \ldots, L_k$ *unary regular languages*
- L_i *accepted by a dfa of size* (λ_i, μ_i), $i = 1, \ldots, k$.

Then $\bigcup_{i=1}^{k} L_i$ *is accepted by a dfa of size*

$$(lcm(\lambda_1, \ldots, \lambda_k), \max(\mu_1, \ldots, \mu_k))$$

Unary cfg $\longrightarrow$ dfa (1)

Given a variable $A \in V$:

- Let a L_A be the set of strings in L generated using A
- A is said to be *cyclic* iff $A \stackrel{+}{\Rightarrow} a^i A a^j$, for some i, j s.t. $0 < i + j < 2^h$,
- If A is cyclic:
  - we set $\lambda_A = i + j$, for an arbitrary chosen pair of integers (i, j) satisfying the above condition.
  - we can prove that

    the language L_A is accepted by a dfa of size (λ_A, μ_A)

    where $\lambda_A < 2^h$ and $\mu_A = 2^{2h} + (2h - 3)2^{h-1} + 2 - h$.

Notice that $L = L_S$. Hence, *if S is cyclic:*

- L is accepted by a dfa of size (λ_S, μ_S)
- Using the above bounds on λ_S and μ_S, we can prove that

  the number of the states is less than 2^{h^2}

Unary cfg $\longrightarrow$ dfa (1)

Given a variable $A \in V$:

- Let a L_A be the set of strings in L generated using A
- A is said to be *cyclic* iff $A \overset{+}{\Rightarrow} a^i A a^j$, for some i, j s.t. $0 < i + j < 2^h$,
- If A is cyclic:
  - we set $\lambda_A = i + j$, for an arbitrary chosen pair of integers (i, j) satisfying the above condition.
  - we can prove that

    the language L_A is accepted by a dfa of size (λ_A, μ_A)

    where $\lambda_A < 2^h$ and $\mu_A = 2^{2h} + (2h - 3)2^{h-1} + 2 - h$.

Notice that $L = L_S$. Hence, *if S is cyclic:*

- L is accepted by a dfa of size (λ_S, μ_S)
- Using the above bounds on λ_S and μ_S, we can prove that

  the number of the states is less than 2^{h^2}

Given a variable $A \in V$:

- Let a L_A be the set of strings in L generated using A
- A is said to be *cyclic* iff $A \stackrel{+}{\Rightarrow} a^i A a^j$, for some i, j s.t. $0 < i + j < 2^h$,
- If A is cyclic:
  - we set $\lambda_A = i + j$, for an arbitrary chosen pair of integers (i, j) satisfying the above condition.
  - we can prove that

    the language L_A is accepted by a dfa of size (λ_A, μ_A)

    where $\lambda_A < 2^h$ and $\mu_A = 2^{2h} + (2h - 3)2^{h-1} + 2 - h$.

Notice that $L = L_S$. Hence, *if S is cyclic:*

- L is accepted by a dfa of size (λ_S, μ_S)
- Using the above bounds on λ_S and μ_S, we can prove that

  the number of the states is less than 2^{h^2}

Unary cfg $\longrightarrow$ dfa (1)

Given a variable $A \in V$:

- Let a L_A be the set of strings in L generated using A
- A is said to be *cyclic* iff $A \stackrel{+}{\Rightarrow} a^i A a^j$, for some i, j s.t. $0 < i + j < 2^h$,
- If A is cyclic:
  - we set $\lambda_A = i + j$, for an arbitrary chosen pair of integers (i, j) satisfying the above condition.
  - we can prove that

    the language L_A is accepted by a dfa of size (λ_A, μ_A)

    where $\lambda_A < 2^h$ and $\mu_A = 2^{2h} + (2h - 3)2^{h-1} + 2 - h$.

Notice that $L = L_S$. Hence, *if S is cyclic:*

- L is accepted by a dfa of size (λ_S, μ_S)
- Using the above bounds on λ_S and μ_S, we can prove that

  the number of the states is less than 2^{h^2}

Unary cfg → dfa (1)

Given a variable $A \in V$:

- Let a L_A be the set of strings in L generated using A
- A is said to be *cyclic* iff $A \stackrel{+}{\Rightarrow} a^i A a^j$, for some i, j s.t. $0 < i + j < 2^h$,
- If A is cyclic:
  - we set $\lambda_A = i + j$, for an arbitrary chosen pair of integers (i, j) satisfying the above condition.
  - we can prove that

    the language L_A is accepted by a dfa of size (λ_A, μ_A)

    where $\lambda_A < 2^h$ and $\mu_A = 2^{2h} + (2h - 3)2^{h-1} + 2 - h$.

Notice that $L = L_S$. Hence, *if S is cyclic:*

- L is accepted by a dfa of size (λ_S, μ_S)
- Using the above bounds on λ_S and μ_S, we can prove that

  the number of the states is less than 2^{h^2}

Unary cfg $\longrightarrow$ dfa (1)

Given a variable $A \in V$:

- Let a L_A be the set of strings in L generated using A
- A is said to be *cyclic* iff $A \overset{+}{\Rightarrow} a^i A a^j$, for some i, j s.t. $0 < i + j < 2^h$,
- If A is cyclic:
  - we set $\lambda_A = i + j$, for an arbitrary chosen pair of integers (i, j) satisfying the above condition.
  - we can prove that

    the language L_A is accepted by a dfa of size (λ_A, μ_A)

    where $\lambda_A < 2^h$ and $\mu_A = 2^{2h} + (2h - 3)2^{h-1} + 2 - h$.

Notice that $L = L_S$. Hence, *if S is cyclic:*

- L is accepted by a dfa of size (λ_S, μ_S)
- Using the above bounds on λ_S and μ_S, we can prove that

  the number of the states is less than 2^{h^2}

Unary cfg $\longrightarrow$ dfa (1)

Given a variable $A \in V$:

- Let a L_A be the set of strings in L generated using A
- A is said to be *cyclic* iff $A \stackrel{+}{\Rightarrow} a^i A a^j$, for some i, j s.t. $0 < i + j < 2^h$,
- If A is cyclic:
  - we set $\lambda_A = i + j$, for an arbitrary chosen pair of integers (i, j) satisfying the above condition.
  - we can prove that

    the language L_A is accepted by a dfa of size (λ_A, μ_A)

    where $\lambda_A < 2^h$ and $\mu_A = 2^{2h} + (2h - 3)2^{h-1} + 2 - h$.

Notice that $L = L_S$. Hence, *if S is cyclic:*

- L is accepted by a dfa of size (λ_S, μ_S)
- Using the above bounds on λ_S and μ_S, we can prove that

  the number of the states is less than 2^{h^2}

Unary cfg $\longrightarrow$ dfa (2)

If S is not cyclic, then we decompose L as:

$$L = L^{\leq 2^{h-1}} \cup \bigcup_{A \in V_p} L_A$$

where V_p denotes the set of cyclic variables.

- $L^{\leq 2^{h-1}}$ is accepted by a dfa of size $(1, 2^{h-1} + 1)$.
- Hence, L is accepted by a dfa of size (λ, μ), where
  - $\lambda = \mathrm{lcm}\{\lambda_A \mid A \in V_p\}$
  - $\mu = \max(2^{h-1} + 1, 2^{2h} + (2h - 3)2^{h-1} + 2 - h)$.
- From $\lambda_A < 2^h$ and $\# V_p < h$, we get that $\lambda \leq (2^h - 1)^{h-1}$.
- By computing $\lambda + \mu$, we finally get that

  the total number of states is less than 2^{h^2}

Unary cfg $\longrightarrow$ dfa (2)

If S is not cyclic, then we decompose L as:

$$L = L^{\leq 2^{h-1}} \cup \bigcup_{A \in V_p} L_A$$

where V_p denotes the set of cyclic variables.

- $L^{\leq 2^{h-1}}$ is accepted by a dfa of size $(1, 2^{h-1}+1)$.
- Hence, L is accepted by a dfa of size (λ, μ), where
  - $\lambda = \mathrm{lcm}\{\lambda_A \mid A \in V_p\}$
  - $\mu = \max(2^{h-1}+1, 2^{2h} + (2h-3)2^{h-1} + 2 - h)$.
- From $\lambda_A < 2^h$ and $\#V_p < h$, we get that $\lambda \leq (2^h - 1)^{h-1}$.
- By computing $\lambda + \mu$, we finally get that

  the total number of states is less than 2^{h^2}

If S is not cyclic, then we decompose L as:

$$L = L^{\leq 2^{h-1}} \cup \bigcup_{A \in V_p} L_A$$

where V_p denotes the set of cyclic variables.

- $L^{\leq 2^{h-1}}$ is accepted by a dfa of size $(1, 2^{h-1}+1)$.
- Hence, L is accepted by a dfa of size (λ, μ), where
  - $\lambda = \text{lcm}\{\lambda_A \mid A \in V_p\}$
  - $\mu = \max(2^{h-1}+1, 2^{2h}+(2h-3)2^{h-1}+2-h)$.
- From $\lambda_A < 2^h$ and $\# V_p < h$, we get that $\lambda \leq (2^h-1)^{h-1}$.
- By computing $\lambda + \mu$, we finally get that

  the total number of states is less than 2^{h^2}

Unary cfg $\longrightarrow$ dfa (2)

If S is not cyclic, then we decompose L as:

$$L = L^{\leq 2^{h-1}} \cup \bigcup_{A \in V_p} L_A$$

where V_p denotes the set of cyclic variables.

- $L^{\leq 2^{h-1}}$ is accepted by a dfa of size $(1, 2^{h-1} + 1)$.
- Hence, L is accepted by a dfa of size (λ, μ), where
  - $\lambda = \mathrm{lcm}\{\lambda_A \mid A \in V_p\}$
  - $\mu = \max(2^{h-1} + 1, 2^{2h} + (2h - 3)2^{h-1} + 2 - h)$.
- From $\lambda_A < 2^h$ and $\# V_p < h$, we get that $\lambda \leq (2^h - 1)^{h-1}$.
- By computing $\lambda + \mu$, we finally get that

  the total number of states is less than 2^{h^2}

Summarizing:

Theorem

For any unary cfg in Chomsky normal form with $h \geq 2$ variables, there exists an equivalent dfa with less than 2^{h^2} states.

The upper bound is tight!!!

Theorem

There is a constant $c > 0$ s.t., for infinitely many integers $h > 0$, there exists a unary cfg in Chomsky normal form with h variables, s.t. any equivalent dfa must have 2^{ch^2} states.

Summarizing:

Theorem

For any unary cfg in Chomsky normal form with $h \geq 2$ variables, there exists an equivalent dfa with less than 2^{h^2} states.

The upper bound is tight!!!

Theorem

There is a constant $c > 0$ s.t., for infinitely many integers $h > 0$, there exists a unary cfg in Chomsky normal form with h variables, s.t. any equivalent dfa must have 2^{ch^2} states.

Summarizing:

Theorem

For any unary cfg in Chomsky normal form with $h \geq 2$ variables, there exists an equivalent dfa with less than 2^{h^2} states.

The upper bound is tight!!!

Theorem

There is a constant $c > 0$ s.t., for infinitely many integers $h > 0$, there exists a unary cfg in Chomsky normal form with h variables, s.t. any equivalent dfa must have 2^{ch^2} states.

Unary cfg $\longrightarrow$ dfa

Summarizing:

Theorem

For any unary cfg in Chomsky normal form with $h \geq 2$ variables, there exists an equivalent dfa with less than 2^{h^2} states.

The upper bound is tight!!!

Theorem

There is a constant $c > 0$ s.t., for infinitely many integers $h > 0$, there exists a unary cfg in Chomsky normal form with h variables, s.t. any equivalent dfa must have 2^{ch^2} states.

Corollary

For any unary pushdown automaton with

- n *states*
- m *pushdown symbols*

s.t. each push add one symbol on the stack, there exist

- *an equivalent nfa with at most* $2^{2n^2m+1}+1$ *states*
- *an equivalent dfa with less than* $2^{n^4m^2+2n^2m+1}$ *states.*

Proof idea:
The pda can be transformed into a cfg in Chomsky normal form with n^2m+1 variables.

...

Unary pushdown automata $\rightarrow$ finite automata

Corollary

For any unary pushdown automaton with

- n *states*
- m *pushdown symbols*

s.t. each push add one symbol on the stack, there exist

- *an equivalent nfa with at most* $2^{2n^2m+1}+1$ *states*
- *an equivalent dfa with less than* $2^{n^4m^2+2n^2m+1}$ *states.*

Proof idea:
The pda can be transformed into a cfg in Chomsky normal form with n^2m+1 variables.

...

Unary pushdown automata $\rightarrow$ finite automata

Corollary

For any unary pushdown automaton with

- n *states*
- m *pushdown symbols*

s.t. each push add one symbol on the stack, there exist

- *an equivalent nfa with at most* $2^{2n^2m+1}+1$ *states*
- *an equivalent dfa with less than* $2^{n^4m^2+2n^2m+1}$ *states.*

Proof idea:
The pda can be transformed into a cfg in Chomsky normal form with n^2m+1 variables.

...

Corollary

For any unary pushdown automaton with

- n *states*
- m *pushdown symbols*

s.t. each push add one symbol on the stack, there exist

- *an equivalent nfa with at most* $2^{2n^2m+1} + 1$ *states*
- *an equivalent dfa with less than* $2^{n^4m^2+2n^2m+1}$ *states.*

Proof idea:
The pda can be transformed into a cfg in Chomsky normal form with n^2m+1 variables.

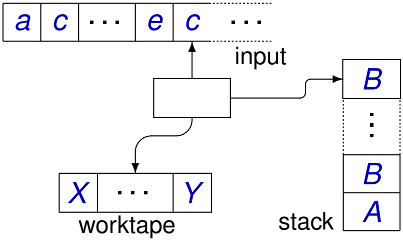
...

Auxiliary pushdown automata (auxpda)

Turing machines augmented with a pushdown store

or, equivalently

(2way) pda augmented with an auxiliary worktape



"SPACE" $\equiv$ worktape

Theorem ([Cook 1971])

Given $L \subseteq \Sigma^*$, $s(n) \geq \log n$, the following statements are equivalent:

1. *L is accepted in $s(n)$ space by a nondeterministic auxpda*
2. *L is accepted in $s(n)$ space by a deterministic auxpda*
3. *L is accepted in $2^{O(s(n))}$ time by a deterministic Turing machine.*

Hence: $DAuxPDA(\log n) = NAuxPDA(\log n) = P$

“SPACE” $\equiv$ worktape

Theorem ([Cook 1971])

Given $L \subseteq \Sigma^*$, $s(n) \geq \log n$, the following statements are equivalent:

1. *L is accepted in $s(n)$ space by a nondeterministic auxpda*
2. *L is accepted in $s(n)$ space by a deterministic auxpda*
3. *L is accepted in $2^{O(s(n))}$ time by a deterministic Turing machine.*

Hence: $DAuxPDA(\log n) = NAuxPDA(\log n) = P$

“SPACE” $\equiv$ worktape

Theorem ([Cook 1971])

Given $L \subseteq \Sigma^*$, $s(n) \geq \log n$, the following statements are equivalent:

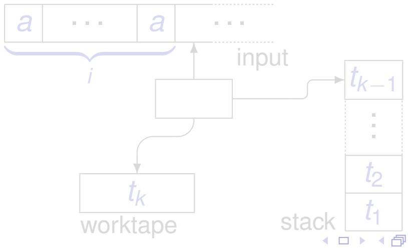
1. *L is accepted in $s(n)$ space by a nondeterministic auxpda*
2. *L is accepted in $s(n)$ space by a deterministic auxpda*
3. *L is accepted in $2^{O(s(n))}$ time by a deterministic Turing machine.*

Hence: $DAuxPDA(\log n) = NAuxPDA(\log n) = P$

The input head can be moved only to the right

Example: how to count the input length.

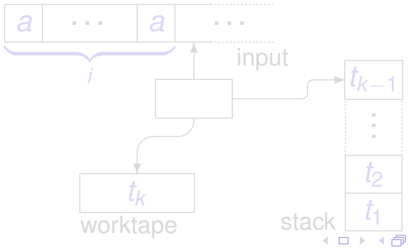
- Given $i \in \mathbf{N}$ consider its binary representation.
- Let $t_1 > \ldots > t_k$ be the sequence of the positions of digits 1, i.e., $i = 2^{t_1} + 2^{t_2} + \ldots + 2^{t_k}$.
- The auxdpa can store i (the length of the scanned input prefix) as follows:



The input head can be moved only to the right

Example: how to count the input length.

- Given $i \in \mathbf{N}$ consider its binary representation.
- Let $t_1 > \ldots > t_k$ be the sequence of the positions of digits 1, i.e., $i = 2^{t_1} + 2^{t_2} + \ldots + 2^{t_k}$.
- The auxdpa can store i (the length of the scanned input prefix) as follows:

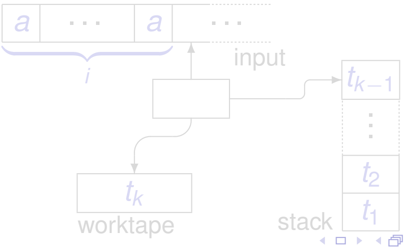


One-way auxiliary pda (1auxpda) [Brandenburg 1977]

The input head can be moved only to the right

Example: how to count the input length.

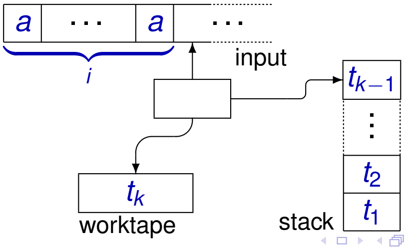
- Given $i \in \mathbf{N}$ consider its binary representation.
- Let $t_1 > \ldots > t_k$ be the sequence of the positions of digits 1, i.e., $i = 2^{t_1} + 2^{t_2} + \ldots + 2^{t_k}$.
- The auxdpa can store i (the length of the scanned input prefix) as follows:



The input head can be moved only to the right

Example: how to count the input length.

- Given $i \in \mathbf{N}$ consider its binary representation.
- Let $t_1 > \ldots > t_k$ be the sequence of the positions of digits 1, i.e., $i = 2^{t_1} + 2^{t_2} + \ldots + 2^{t_k}$.
- The auxdpa can store i (the length of the scanned input prefix) as follows:

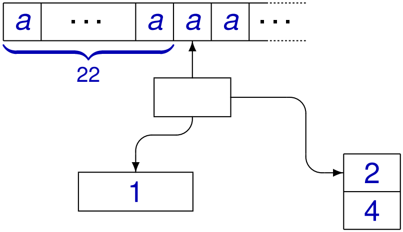


1auxpda: how to count the input length

$$22 = 2^4 + 2^2 + 2^1$$

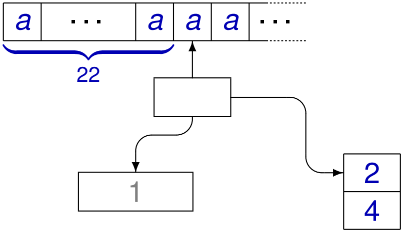
$$23 = 2^4 + 2^2 + 2^1 + 2^0$$

$$22 = 2^4 + 2^2 + 2^1$$



$$23 = 2^4 + 2^2 + 2^1 + 2^0$$

1auxpda: how to count the input length

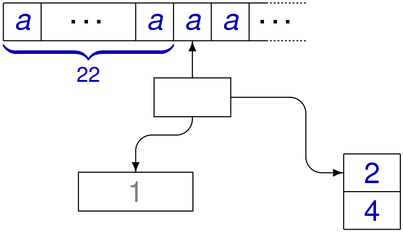


$$22 = 2^4 + 2^2 + 2^1$$

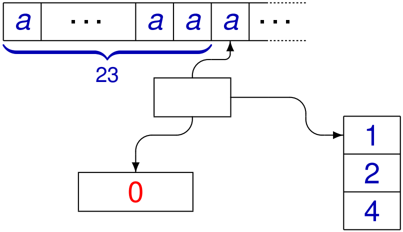
$$23 = 2^4 + 2^2 + 2^1 + 2^0$$

1auxpda: how to count the input length

$$22 = 2^4 + 2^2 + 2^1$$



$$23 = 2^4 + 2^2 + 2^1 + 2^0$$



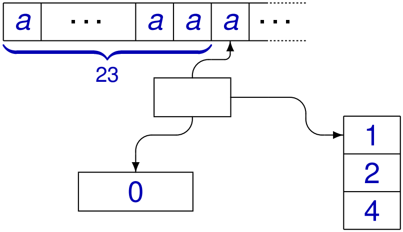
1auxpda: how to count the input length

$$23 = 2^4 + 2^2 + 2^1 + 2^0$$

$$24 = 2^4 + 2^3$$

1auxpda: how to count the input length

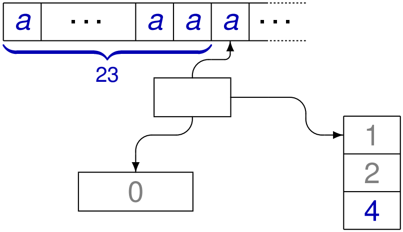
$$23 = 2^4 + 2^2 + 2^1 + 2^0$$



$$24 = 2^4 + 2^3$$

1auxpda: how to count the input length

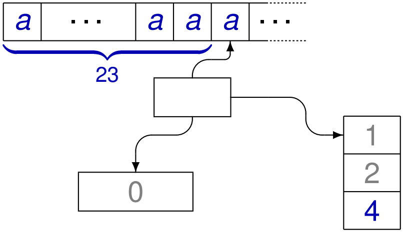
$23 = 2^4 + 2^2 + 2^1 + 2^0$



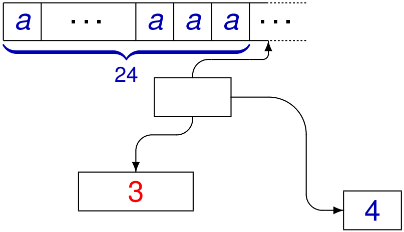
$24 = 2^4 + 2^3$

1auxpda: how to count the input length

$23 = 2^4 + 2^2 + 2^1 + 2^0$



$24 = 2^4 + 2^3$



1auxpda: example

Let $\mathcal{L} = \{a^{2^k} \mid k \geq 0\}$.

We define a 1auxpda M working as follows:

- M scans the input tape, counting the input length.
- When the end of the input is reached, M accepts if and only if the pushdown store is empty.
- If n is the input length, then the largest integer stored on the worktape is $\lfloor \log_2 n \rfloor$.
- It can be represented in $O(\log \log n)$ space.

Hence $\mathcal{L} = \{a^{2^k} \mid k \geq 0\}$ is accepted by the deterministic 1auxpda M in $O(\log \log n)$ space.

1auxpda: example

Let $\mathcal{L} = \{a^{2^k} \mid k \geq 0\}$.
We define a 1auxpda M working as follows:

- M scans the input tape, counting the input length.
- When the end of the input is reached, M accepts if and only if the pushdown store is empty.
- If n is the input length, then the largest integer stored on the worktape is $\lfloor \log_2 n \rfloor$.
- It can be represented in $O(\log \log n)$ space.

Hence $\mathcal{L} = \{a^{2^k} \mid k \geq 0\}$ is accepted by the deterministic 1auxpda M in $O(\log \log n)$ space.

1auxpda: example

Let $\mathcal{L} = \{a^{2^k} \mid k \geq 0\}$.
We define a 1auxpda M working as follows:

- M scans the input tape, counting the input length.
- When the end of the input is reached, M accepts if and only if the pushdown store is empty.
- If n is the input length, then the largest integer stored on the worktape is $\lfloor \log_2 n \rfloor$.
- It can be represented in $O(\log \log n)$ space.

Hence $\mathcal{L} = \{a^{2^k} \mid k \geq 0\}$ is accepted by the deterministic 1auxpda M in $O(\log \log n)$ space.

1auxpda: example

Let $\mathcal{L} = \{a^{2^k} \mid k \geq 0\}$.
We define a 1auxpda M working as follows:

- M scans the input tape, counting the input length.
- When the end of the input is reached, M accepts if and only if the pushdown store is empty.
- If n is the input length, then the largest integer stored on the worktape is $\lfloor \log_2 n \rfloor$.
- It can be represented in $O(\log \log n)$ space.

Hence $\mathcal{L} = \{a^{2^k} \mid k \geq 0\}$ is accepted by the deterministic 1auxpda M in $O(\log \log n)$ space.

1auxpda: example

Let $\mathcal{L} = \{a^{2^k} \mid k \geq 0\}$.
We define a 1auxpda M working as follows:

- M scans the input tape, counting the input length.
- When the end of the input is reached, M accepts if and only if the pushdown store is empty.
- If n is the input length, then the largest integer stored on the worktape is $\lfloor \log_2 n \rfloor$.
- It can be represented in $O(\log \log n)$ space.

Hence $\mathcal{L} = \{a^{2^k} \mid k \geq 0\}$ is accepted by the deterministic 1auxpda M in $O(\log \log n)$ space.

1auxpda: example

Let $\mathcal{L} = \{a^{2^k} \mid k \geq 0\}$.
We define a 1auxpda M working as follows:

- M scans the input tape, counting the input length.
- When the end of the input is reached, M accepts if and only if the pushdown store is empty.
- If n is the input length, then the largest integer stored on the worktape is $\lfloor \log_2 n \rfloor$.
- It can be represented in $O(\log \log n)$ space.

Hence $\mathcal{L} = \{a^{2^k} \mid k \geq 0\}$ is accepted by the deterministic 1auxpda M in $O(\log \log n)$ space.

One-way auxiliary pda (1auxpda)

Problem ([Brandenburg 1977])

What is the minimum amount of space $s(n)$ s.t. 1auxpda working in $s(n)$ space are able to accept noncontext-free languages?

space $s(n)$:

STRONG: *any computation on each input* of length n uses no more than $s(n)$ worktape cells.

WEAK: on each *accepted input* of length n there exists *at least one accepting computation* using no more than $s(n)$ worktape cells.

One-way auxiliary pda (1auxpda)

Problem ([Brandenburg 1977])

What is the minimum amount of space $s(n)$ s.t. 1auxpda working in $s(n)$ space are able to accept noncontext-free languages?

space $s(n)$:

STRONG: *any computation on each input* of length n uses no more than $s(n)$ worktape cells.

WEAK: on each *accepted input* of length n there exists *at least one accepting computation* using no more than $s(n)$ worktape cells.

One-way auxiliary pda (1auxpda)

Problem ([Brandenburg 1977])

What is the minimum amount of space $s(n)$ s.t. 1auxpda working in $s(n)$ space are able to accept noncontext-free languages?

space $s(n)$:

STRONG: *any computation on each input* of length n uses no more than $s(n)$ worktape cells.

WEAK: on each *accepted input* of length n there exists *at least one accepting computation* using no more than $s(n)$ worktape cells.

One-way auxiliary pda (1auxpda)

Problem ([Brandenburg 1977])

What is the minimum amount of space $s(n)$ s.t. 1auxpda working in $s(n)$ space are able to accept noncontext-free languages?

space $s(n)$:

STRONG: *any computation on each input* of length n uses no more than $s(n)$ worktape cells.

WEAK: on each *accepted input* of length n there exists *at least one accepting computation* using no more than $s(n)$ worktape cells.

The space bound should be satisfied by all computations

Theorem ([Brandenburg 1977])

$L \subseteq \Sigma^*$ noncontext-free language accepted by a 1auxpda in strong $s(n)$ space.
Then there exists $c > 0$ such that

$$s(n) \geq c \log \log n$$

infinitely often.

This lower bound is tight!

$$\mathcal{L} = \{a^{2^k} \mid k \geq 0\}.$$

1auxpda in strong space

The space bound should be satisfied by all computations

Theorem ([Brandenburg 1977])

$L \subseteq \Sigma^*$ noncontext-free language accepted by a 1auxpda in strong $s(n)$ space.
Then there exists $c > 0$ such that

$$s(n) \geq c \log \log n$$

infinitely often.

This lower bound is tight!

$$\mathcal{L} = \{a^{2^k} \mid k \geq 0\}.$$

1auxpda in strong space

The space bound should be satisfied by all computations

Theorem ([Brandenburg 1977])

$L \subseteq \Sigma^*$ noncontext-free language accepted by a 1auxpda in strong $s(n)$ space.
Then there exists $c > 0$ such that

$$s(n) \geq c \log \log n$$

infinitely often.

This lower bound is tight!

$$\mathcal{L} = \{a^{2^k} \mid k \geq 0\}.$$

1auxpda in strong space

The space bound should be satisfied by all computations

Theorem ([Brandenburg 1977])

$L \subseteq \Sigma^*$ noncontext-free language accepted by a 1auxpda in strong $s(n)$ space.
Then there exists $c > 0$ such that

$$s(n) \geq c \log \log n$$

infinitely often.

This lower bound is tight!

$$\mathcal{L} = \{a^{2^k} \mid k \geq 0\}.$$

1auxpda in weak space

For each accepted input, at least one accepting computation satisfies the bound.

Theorem ([Chytil 1986])

1. *For each integer $k \geq 2$ there is a language L_k such that*
   - L_k *is accepted by a 1auxpda in weak* $O(\underbrace{\log \ldots \log}_{k} n)$ *space*
   - L_k *cannot be accepted by 1auxpdas in weak* $o(\underbrace{\log \ldots \log}_{k} n)$ *space.*
2. *There exists a noncontext-free language L accepted by a 1auxpda in weak $O(\log^* n)$, where* $\log^* n = \min\{k \mid \underbrace{\log \ldots \log}_{k} n \leq 1\}$.

Languages L_k and L are defined over *binary alphabets.*

1auxpda in weak space

For each accepted input, at least one accepting computation satisfies the bound.

Theorem ([Chytil 1986])

1. *For each integer $k \geq 2$ there is a language L_k such that*
   - *L_k is accepted by a 1auxpda in weak $O(\underbrace{\log \ldots \log}_{k} n)$ space*
   - *L_k cannot be accepted by 1auxpdas in weak $o(\underbrace{\log \ldots \log}_{k} n)$ space.*
2. *There exists a noncontext-free language L accepted by a 1auxpda in weak $O(\log^* n)$, where* $\log^* n = \min\{k \mid \underbrace{\log \ldots \log}_{k} n \leq 1\}$.

Languages L_k and L are defined over *binary alphabets.*

1auxpda in weak space

For each accepted input, at least one accepting computation satisfies the bound.

Theorem ([Chytil 1986])

1. *For each integer $k \geq 2$ there is a language L_k such that*
   - L_k *is accepted by a 1auxpda in weak* $O(\underbrace{\log \ldots \log}_{k} n)$ *space*
   - L_k *cannot be accepted by 1auxpdas in weak* $o(\underbrace{\log \ldots \log}_{k} n)$ *space.*
2. *There exists a noncontext-free language L accepted by a 1auxpda in weak $O(\log^* n)$, where* $\log^* n = \min\{k \mid \underbrace{\log \ldots \log}_{k} n \leq 1\}$.

Languages L_k and L are defined over *binary alphabets.*

1auxpda in weak space

For each accepted input, at least one accepting computation satisfies the bound.

Theorem ([Chytil 1986])

1. *For each integer $k \geq 2$ there is a language L_k such that*
   - L_k *is accepted by a 1auxpda in weak* $O(\underbrace{\log \ldots \log}_{k} n)$ *space*
   - L_k *cannot be accepted by 1auxpdas in weak* $o(\underbrace{\log \ldots \log}_{k} n)$ *space.*
2. *There exists a noncontext-free language L accepted by a 1auxpda in weak $O(\log^* n)$, where* $\log^* n = \min\{k \mid \underbrace{\log \ldots \log}_{k} n \leq 1\}$.

Languages L_k and L are defined over *binary alphabets.*

1auxpda in weak space

For each accepted input, at least one accepting computation satisfies the bound.

Theorem ([Chytil 1986])

1. *For each integer $k \geq 2$ there is a language L_k such that*
   - L_k *is accepted by a 1auxpda in weak* $O(\underbrace{\log \ldots \log}_{k} n)$ *space*
   - L_k *cannot be accepted by 1auxpdas in weak* $o(\underbrace{\log \ldots \log}_{k} n)$ *space.*
2. *There exists a noncontext-free language L accepted by a 1auxpda in weak $O(\log^* n)$, where*
   $\log^* n = \min \{k \mid \underbrace{\log \ldots \log}_{k} n \leq 1\}$.

Languages L_k and L are defined over *binary alphabets.*

1auxpda in weak space

For each accepted input, at least one accepting computation satisfies the bound.

Theorem ([Chytil 1986])

1. *For each integer $k \geq 2$ there is a language L_k such that*
   - L_k *is accepted by a 1auxpda in weak* $O(\underbrace{\log \ldots \log}_{k} n)$ *space*
   - L_k *cannot be accepted by 1auxpdas in weak* $o(\underbrace{\log \ldots \log}_{k} n)$ *space.*
2. *There exists a noncontext-free language L accepted by a 1auxpda in weak $O(\log^* n)$, where* $\log^* n = \min\{k \mid \underbrace{\log \ldots \log}_{k} n \leq 1\}$.

Languages L_k and L are defined over *binary alphabets.*

space bounds for 1auxpda

	general case	unary case
strong space		
weak space		

space bounds for 1auxpda

	general case	unary case
strong space	lower bound $\log \log n$ [1] optimal	lower bound $\log \log n$ [1] optimal
weak space		

1: Brandenburg, 1977

space bounds for 1auxpda

	general case	unary case
strong space	lower bound $\log \log n$ [1] optimal	lower bound $\log \log n$ [1] optimal
weak space	upper bound $\log^* n$ [2]	

1: Brandenburg, 1977

2: Chytil, 1986

	general case	unary case
strong space	lower bound $\log\log n$ [1] optimal	lower bound $\log\log n$ [1] optimal
weak space	upper bound $\log^* n$ [2]	**?**

1: Brandenburg, 1977

2: Chytil, 1986

What about the unary case?

Automaticity [Shallit and Breitbart,1996]

- $A_L : \mathbf{N} \to \mathbf{N}$ *automaticity* of a language $L \subseteq \Sigma^*$:

$$A_L(n) = \text{minimum number of states of a dfa accepting a language } L', \text{ s.t. } L^{\leq n} = L'^{\leq n}$$

i.e., L and L' agree on strings of length $\leq n$.

- If L is regular the $A_L(n)$ is a constant (the size of the minimal dfa accepting L).

What about the automaticity of nonregular languages?

Theorem ([Karp, 1971])

Let $L \subseteq \Sigma^*$ be a nonregular language. Then:

$$A_L(n) \geq \frac{n+3}{2} \text{ for infinitely many integers } n.$$

- $A_L : \mathbf{N} \to \mathbf{N}$ *automaticity* of a language $L \subseteq \Sigma^*$:

$$A_L(n) = \text{minimum number of states of a dfa accepting a language } L', \text{ s.t. } L^{\leq n} = L'^{\leq n}$$

i.e., L and L' agree on strings of length $\leq n$.

- If L is regular the $A_L(n)$ is a constant (the size of the minimal dfa accepting L).

What about the automaticity of nonregular languages?

Theorem ([Karp, 1971])

Let $L \subseteq \Sigma^*$ be a nonregular language. Then:

$$A_L(n) \geq \frac{n+3}{2} \text{ for infinitely many integers } n.$$

- $A_L : \mathbf{N} \to \mathbf{N}$ *automaticity* of a language $L \subseteq \Sigma^*$:

$$A_L(n) = \text{minimum number of states of a dfa accepting a language } L', \text{ s.t. } L^{\leq n} = L'^{\leq n}$$

i.e., L and L' agree on strings of length $\leq n$.

- If L is regular the $A_L(n)$ is a constant (the size of the minimal dfa accepting L).

What about the automaticity of nonregular languages?

Theorem ([Karp, 1971])

Let $L \subseteq \Sigma^*$ be a nonregular language. Then:

$$A_L(n) \geq \frac{n+3}{2} \text{ for infinitely many integers } n.$$

- $A_L : \mathbf{N} \to \mathbf{N}$ *automaticity* of a language $L \subseteq \Sigma^*$:

$$A_L(n) = \text{minimum number of states of a dfa accepting a language } L', \text{ s.t. } L^{\leq n} = L'^{\leq n}$$
i.e., L and L' agree on strings of length $\leq n$.

- If L is regular the $A_L(n)$ is a constant (the size of the minimal dfa accepting L).

What about the automaticity of nonregular languages?

Theorem ([Karp, 1971])

Let $L \subseteq \Sigma^*$ be a nonregular language. Then:

$$A_L(n) \geq \frac{n+3}{2} \text{ for infinitely many integers } n.$$

1auxpda and weak space: unary case

- M: unary 1auxpda accepting in weak $s(n)$ space a nonregular language L.
- M_n: a pda whose states encode the configurations of M using $s(n)$ space, for a given $n \geq 1$.
  - M_n has $h = 2^{O(s(n))}$ states.
  - $L(M_n)^{\leq n} = L^{\leq n}$
- A_n: a dfa simulating M_n
  - A_n has $2^{h^2} = 2^{2^{O(s(n))}}$ states.
  - $L(A_n)^{\leq n} = L^{\leq n}$
- By the result of Karp, the number of states of A_n must be at least $\frac{n+3}{2}$, i.o.

Hence the space $s(n)$ must grow at least as $\log\log n$

1auxpda and weak space: unary case

- M: unary 1auxpda accepting in weak $s(n)$ space a nonregular language L.
- M_n: a pda whose states encode the configurations of M using $s(n)$ space, for a given $n \geq 1$.
  - M_n has $h = 2^{O(s(n))}$ states.
  - $L(M_n)^{\leq n} = L^{\leq n}$
- A_n: a dfa simulating M_n
  - A_n has $2^{h^2} = 2^{2^{O(s(n))}}$ states.
  - $L(A_n)^{\leq n} = L^{\leq n}$
- By the result of Karp, the number of states of A_n must be at least $\frac{n+3}{2}$, i.o.

Hence the space $s(n)$ must grow at least as $\log \log n$

1auxpda and weak space: unary case

- M: unary 1auxpda accepting in weak $s(n)$ space a nonregular language L.
- M_n: a pda whose states encode the configurations of M using $s(n)$ space, for a given $n \geq 1$.
  - M_n has $h = 2^{O(s(n))}$ states.
  - $L(M_n)^{\leq n} = L^{\leq n}$
- A_n: a dfa simulating M_n
  - A_n has $2^{h^2} = 2^{2^{O(s(n))}}$ states.
  - $L(A_n)^{\leq n} = L^{\leq n}$
- By the result of Karp, the number of states of A_n must be at least $\frac{n+3}{2}$, i.o.

Hence the space $s(n)$ must grow at least as $\log\log n$

1auxpda and weak space: unary case

- M: unary 1auxpda accepting in weak $s(n)$ space a nonregular language L.
- M_n: a pda whose states encode the configurations of M using $s(n)$ space, for a given $n \geq 1$.
  - M_n has $h = 2^{O(s(n))}$ states.
  - $L(M_n)^{\leq n} = L^{\leq n}$
- A_n: a dfa simulating M_n
  - A_n has $2^{h^2} = 2^{2^{O(s(n))}}$ states.
  - $L(A_n)^{\leq n} = L^{\leq n}$
- By the result of Karp, the number of states of A_n must be at least $\frac{n+3}{2}$, i.o.

Hence the space $s(n)$ must grow at least as $\log \log n$

1auxpda and weak space: unary case

- M: unary 1auxpda accepting in weak $s(n)$ space a nonregular language L.
- M_n: a pda whose states encode the configurations of M using $s(n)$ space, for a given $n \geq 1$.
  - M_n has $h = 2^{O(s(n))}$ states.
  - $L(M_n)^{\leq n} = L^{\leq n}$
- A_n: a dfa simulating M_n
  - A_n has $2^{h^2} = 2^{2^{O(s(n))}}$ states.
  - $L(A_n)^{\leq n} = L^{\leq n}$
- By the result of Karp, the number of states of A_n must be at least $\frac{n+3}{2}$, i.o.

Hence the space $s(n)$ must grow at least as $\log \log n$

1auxpda and weak space: unary case

- M: unary 1auxpda accepting in weak $s(n)$ space a nonregular language L.
- M_n: a pda whose states encode the configurations of M using $s(n)$ space, for a given $n \geq 1$.
  - M_n has $h = 2^{O(s(n))}$ states.
  - $L(M_n)^{\leq n} = L^{\leq n}$
- A_n: a dfa simulating M_n
  - A_n has $2^{h^2} = 2^{2^{O(s(n))}}$ states.
  - $L(A_n)^{\leq n} = L^{\leq n}$
- By the result of Karp, the number of states of A_n must be at least $\frac{n+3}{2}$, i.o.

Hence the space $s(n)$ must grow at least as $\log \log n$

1auxpda and weak space: unary case

- M: unary 1auxpda accepting in weak $s(n)$ space a nonregular language L.
- M_n: a pda whose states encode the configurations of M using $s(n)$ space, for a given $n \geq 1$.
  - M_n has $h = 2^{O(s(n))}$ states.
  - $L(M_n)^{\leq n} = L^{\leq n}$
- A_n: a dfa simulating M_n
  - A_n has $2^{h^2} = 2^{2^{O(s(n))}}$ states.
  - $L(A_n)^{\leq n} = L^{\leq n}$
- By the result of Karp, the number of states of A_n must be at least $\frac{n+3}{2}$, i.o.

Hence the space $s(n)$ must grow at least as $\log \log n$

We have obtained the following:

Theorem

Let M be a unary auxpda accepting a non-context-free language L in weak $s(n)$ space. Then $s(n) \notin o(\log \log n)$.

The optimality can be proved again by considering $\mathcal{L} = \{a^{2^n} \mid n \geq 0\}$.

We have obtained the following:

Theorem

Let M be a unary auxpda accepting a non-context-free language L in weak $s(n)$ space. Then $s(n) \notin o(\log \log n)$.

The optimality can be proved again by considering $\mathcal{L} = \{a^{2^n} \mid n \geq 0\}$.

One-way auxiliary pushdown automata

	general case	unary case
strong space	lower bound $\log \log n$ [1] optimal	lower bound $\log \log n$ [1] optimal
weak space	upper bound $\log^* n$ [2]	**?**

1: Brandenburg, 1977

2: Chytil, 1986

One-way auxiliary pushdown automata

	general case	unary case
strong space	lower bound $\log\log n$ [1] optimal	lower bound $\log\log n$ [1] optimal
weak space	upper bound $\log^* n$ [2]	lower bound $\log\log n$ optimal

1: Brandenburg, 1977

2: Chytil, 1986

Extension to bounded languages

Bounded languages:
Subsets of $w_1^* w_2^* \dots w_n^*$, for given words $w_1, \dots, w_n$
(*letter bounded* if $w_1, \dots, w_n \in \Sigma$).

- The class bounded regular languages is *properly* included in that of bounded cfl's, e.g., $\{a^n b^n \mid n \geq 0\}$.
- Bounded cfl's can be accepted by *finite turn* pda's

Problem: Find a tight upper bound $f(h)$ for the size of *finite turn* pda's equivalent to cfg's with h variables.

Bounded languages:
Subsets of $w_1^* w_2^* \ldots w_n^*$, for given words $w_1, \ldots, w_n$
(*letter bounded* if $w_1, \ldots, w_n \in \Sigma$).

- The class bounded regular languages is *properly* included in that of bounded cfl's, e.g., $\{a^n b^n \mid n \geq 0\}$.
- Bounded cfl's can be accepted by *finite turn* pda's.

Problem: Find a tight upper bound $f(h)$ for the size of *finite turn* pda's equivalent to cfg's with h variables.

Extension to bounded languages

Bounded languages:
Subsets of $w_1^* w_2^* \dots w_n^*$, for given words $w_1, \dots, w_n$
(*letter bounded* if $w_1, \dots, w_n \in \Sigma$).

- The class bounded regular languages is *properly* included in that of bounded cfl's, e.g., $\{a^n b^n \mid n \geq 0\}$.
- Bounded cfl's can be accepted by *finite turn* pda's.

Problem: Find a tight upper bound $f(h)$ for the size of *finite turn* pda's equivalent to cfg's with h variables.

Extension to bounded languages

Bounded languages:
Subsets of $w_1^* w_2^* \dots w_n^*$, for given words $w_1, \dots, w_n$
(*letter bounded* if $w_1, \dots, w_n \in \Sigma$).

- The class bounded regular languages is *properly* included in that of bounded cfl's, e.g., $\{a^n b^n \mid n \geq 0\}$.
- Bounded cfl's can be accepted by *finite turn* pda's.

Problem: Find a tight upper bound $f(h)$ for the size of *finite turn* pda's equivalent to cfg's with h variables.

Extension to bounded languages [Malcher, Pighizzini, 2007]

- The *generation procedure* used to simulate unary cfg's with nfa's can be extended in order to simulate cfg's generating *letter bounded* languages with finite turn pda's.
- Using a suitable homomorphism such a simulation can be extended also to the case of *bounded* languages

Theorem

Each bounded context-free language generated by a cfg with h variables in Chomsky normal form is accepted by a finite-turn pda with 2^h states and $O(1)$ stack symbols.

Extension to bounded languages [Malcher, Pighizzini, 2007]

- The *generation procedure* used to simulate unary cfg's with nfa's can be extended in order to simulate cfg's generating *letter bounded* languages with finite turn pda's.
- Using a suitable homomorphism such a simulation can be extended also to the case of *bounded* languages

Theorem

Each bounded context-free language generated by a cfg with h variables in Chomsky normal form is accepted by a finite-turn pda with 2^h states and $O(1)$ stack symbols.

Extension to bounded languages [Malcher, Pighizzini, 2007]

- The *generation procedure* used to simulate unary cfg's with nfa's can be extended in order to simulate cfg's generating *letter bounded* languages with finite turn pda's.
- Using a suitable homomorphism such a simulation can be extended also to the case of *bounded* languages

Theorem

Each bounded context-free language generated by a cfg with h variables in Chomsky normal form is accepted by a finite-turn pda with 2^h states and $O(1)$ stack symbols.

Extension to bounded languages [Malcher, Pighizzini, 2007]

Even this upper bound in tight.

In particular:

For all integers $m \geq 1, h \geq 1$ there exists a language $L_{m,h} \subseteq a_1^* a_2^* \ldots a_m^*$ s.t.:

- $L_{m,h}$ is generated by a cfg in CNF with h variables

Extension to bounded languages [Malcher, Pighizzini, 2007]

Even this upper bound in tight.

In particular:

For all integers $m \geq 1, h \geq 1$ there exists a language $L_{m,h} \subseteq a_1^* a_2^* \dots a_m^*$ s.t.:

- $L_{m,h}$ is generated by a cfg in Cnf with h variables
- $L_{m,h}$ is accepted by a $(m-1)$-turn pda of size $2^{O(h)}$
-

Extension to bounded languages [Malcher, Pighizzini, 2007]

Even this upper bound in tight.
In particular:

For all integers $m \geq 1$, $h \geq 1$ there exists a language $L_{m,h} \subseteq a_1^* a_2^* \ldots a_m^*$ s.t.:

- $L_{m,h}$ is generated by a cfg in Cnf with h variables
- $L_{m,h}$ is accepted by a $(m-1)$-turn pda of size $2^{O(h)}$
- for each $k \geq m-1$, every k-turn pda accepting $L_{m,h}$ has size *at least* 2^{ch}, for a constant c and each h sufficiently large

Extension to bounded languages [Malcher, Pighizzini, 2007]

Even this upper bound in tight.
In particular:

For all integers $m \geq 1$, $h \geq 1$ there exists a language $L_{m,h} \subseteq a_1^* a_2^* \ldots a_m^*$ s.t.:

- $L_{m,h}$ is generated by a cfg in Cnf with h variables
- $L_{m,h}$ is accepted by a $(m-1)$-turn pda of size $2^{O(h)}$
- for each $k \geq m-1$, every k-turn pda accepting $L_{m,h}$ has size *at least* 2^{ch}, for a constant c and each h sufficiently large
- for each $k < m-1$, $L_{m,h}$ cannot be accepted by k-turn pda's (regardless of the size).

Extension to bounded languages [Malcher, Pighizzini, 2007]

Even this upper bound in tight.
In particular:

For all integers $m \geq 1, h \geq 1$ there exists a language $L_{m,h} \subseteq a_1^* a_2^* \ldots a_m^*$ s.t.:

- $L_{m,h}$ is generated by a cfg in Cnf with h variables
- $L_{m,h}$ is accepted by a $(m-1)$-turn pda of size $2^{O(h)}$
- for each $k \geq m-1$, every k-turn pda accepting $L_{m,h}$ has size *at least* 2^{ch}, for a constant c and each n sufficiently large
- for each $k < m-1$, $L_{m,h}$ cannot be accepted by k-turn pda's (regardless of the size).

Extension to bounded languages [Malcher, Pighizzini, 2007]

Even this upper bound in tight.
In particular:

For all integers $m \geq 1, h \geq 1$ there exists a language $L_{m,h} \subseteq a_1^* a_2^* \ldots a_m^*$ s.t.:

- $L_{m,h}$ is generated by a cfg in Cnf with h variables
- $L_{m,h}$ is accepted by a $(m-1)$-turn pda of size $2^{O(h)}$
- for each $k \geq m-1$, every k-turn pda accepting $L_{m,h}$ has size *at least* 2^{ch}, for a constant c and each n sufficiently large
- for each $k < m-1$, $L_{m,h}$ cannot be accepted by k-turn pda's (regardless of the size).

Extension to bounded languages [Malcher, Pighizzini, 2007]

Even this upper bound in tight.
In particular:

For all integers $m \geq 1, h \geq 1$ there exists a language $L_{m,h} \subseteq a_1^* a_2^* \ldots a_m^*$ s.t.:

- $L_{m,h}$ is generated by a cfg in Cnf with h variables
- $L_{m,h}$ is accepted by a $(m-1)$-turn pda of size $2^{O(h)}$
- for each $k \geq m-1$, every k-turn pda accepting $L_{m,h}$ has size *at least* 2^{ch}, for a constant c and each n sufficiently large
- for each $k < m-1$, $L_{m,h}$ cannot be accepted by k-turn pda's (regardless of the size).

Further investigations: dpda's vs finite automata

Let M be a unary pda with n states and m stack symbols, s.t. each push adds exactly one symbol.

We proved that M can be simulated by a dfa with with $2^{O(n^4 m^2)}$ states.

What about the deterministic case?

Theorem ([Pighizzini, 2008])

If M is deterministic then it can be simulated by a dfa with $2^{O(nm)}$ states.
Furthermore, such a simulation in tight.

Let M be a unary pda with n states and m stack symbols, s.t. each push adds exactly one symbol.

We proved that M can be simulated by a dfa with with $2^{O(n^4 m^2)}$ states.

What about the deterministic case?

Theorem ([Pighizzini, 2008])

If M is deterministic then it can be simulated by a dfa with $2^{O(nm)}$ states.
Furthermore, such a simulation in tight.

Further investigations: dpda's vs finite automata

Let M be a unary pda with n states and m stack symbols, s.t. each push adds exactly one symbol.

We proved that M can be simulated by a dfa with with $2^{O(n^4 m^2)}$ states.

What about the deterministic case?

Theorem ([Pighizzini, 2008])

If M is deterministic then it can be simulated by a dfa with $2^{O(nm)}$ states.
Furthermore, such a simulation in tight.

One-way auxiliary pda's

- 1auxpda's with an input alphabet of at least two symbols can recognize noncontext-free languages using very slowly increasing (but nonconstant) weak space.
- Unary 1auxpda's must use weak space growing at least as $\log\log n$ to recognize noncontext-free languages.

What about space lower bounds for noncontext-free acceptance, for 1auxpda's, with some other kinds of restrictions?

Examples: bounded languages, finite-turn 1auxpda.

One-way auxiliary pda's

- 1auxpda's with an input alphabet of at least two symbols can recognize noncontext-free languages using very slowly increasing (but nonconstant) weak space.
- Unary 1auxpda's must use weak space growing at least as $\log\log n$ to recognize noncontext-free languages.

What about space lower bounds for noncontext-free acceptance, for 1auxpda's, with some other kinds of restrictions?

Examples: bounded languages, finite-turn 1auxpda.

One-way auxiliary pda's

- 1auxpda's with an input alphabet of at least two symbols can recognize noncontext-free languages using very slowly increasing (but nonconstant) weak space.
- Unary 1auxpda's must use weak space growing at least as $\log \log n$ to recognize noncontext-free languages.

What about space lower bounds for noncontext-free acceptance, for 1auxpda's, with some other kinds of restrictions?

Examples: bounded languages, finite-turn 1auxpda.