The hardest LL(k) language

Mikhail Mrykhin and Alexander Okhotin

St. Petersburg State University, Russia

August 18, 2021

Part I

Introduction to hardest languages

The problem and motivation

- NP-hard sets: polynomial-time reductions.

The problem and motivation

- NP-hard sets: polynomial-time reductions.
- Homomorphisms: weakest possible reductions.

The problem and motivation

- NP-hard sets: polynomial-time reductions.
- Homomorphisms: weakest possible reductions.

Hardest language in a family

L_0: for all L there is h_L with $L = h_L^{-1}(L_0)$.

The problem and motivation

- NP-hard sets: polynomial-time reductions.
- Homomorphisms: weakest possible reductions.

Hardest language in a family

L_0: for all L there is h_L with $L = h_L^{-1}(L_0)$.

- Greibach's "hardest context-free language" (1973).

The problem and motivation

- NP-hard sets: polynomial-time reductions.
- Homomorphisms: weakest possible reductions.

Hardest language in a family

L_0: for all L there is h_L with $L = h_L^{-1}(L_0)$.

- Greibach's "hardest context-free language" (1973).
- A parser for L_0 can parse every language.

Known results

- Which standard families have hardest languages?

Known results

- Which standard families have hardest languages?

Model	Result	Author(s)	Year
Context-free languages	+	Greibach	1973
Deterministic languages	-	Greibach	1974
Linear grammars	-	Boasson and Nivat	1977
Counter automata	-	Autebert	1979
Regular languages	-	Culik and Maurer	1979
Two-sided nondeterministic stack automata	+	Rytter	1981
Multihead [stack] automata	+	Miyano	1983
Conjunctive grammars	+	Okhotin	2016
Linear conjunctive grammars	-	Mrykhin and Okhotin	2021
Linear-time cellular automata	+	Mrykhin and Okhotin	2021

Known results

- Which standard families have hardest languages?

Model	Result	Author(s)	Year
Context-free languages	+	Greibach	1973
Deterministic languages	-	Greibach	1974
Linear grammars	-	Boasson and Nivat	1977
Counter automata	-	Autebert	1979
Regular languages	-	Culik and Maurer	1979
Two-sided nondeterministic stack automata	+	Rytter	1981
Multihead [stack] automata	+	Miyano	1983
Conjunctive grammars	+	Okhotin	2016
Linear conjunctive grammars	-	Mrykhin and Okhotin	2021
Linear-time cellular automata	+	Mrykhin and Okhotin	2021

- The LL(k) grammars?

Part II

The curious case of LL(k) grammars

The LL(k) grammars

- A classical family with linear-time parsing.

The LL(k) grammars

- A classical family with linear-time parsing.
- String read from left to right, top-down parsing.

The LL(k) grammars

- A classical family with linear-time parsing.
- String read from left to right, top-down parsing.
- The rule to apply determined by the next k input symbols.

The LL(k) grammars

- A classical family with linear-time parsing.
- String read from left to right, top-down parsing.
- The rule to apply determined by the next k input symbols.
- Parsing table: $T : N \times \Sigma^{\leqslant k} \to R$.

The LL(k) grammars

- A classical family with linear-time parsing.
- String read from left to right, top-down parsing.
- The rule to apply determined by the next k input symbols.
- Parsing table: $T\colon N \times \Sigma^{\leqslant k} \to R$.
- LL($k+1$) grammars more powerful than LL(k).

The LL(k) grammars

- A classical family with linear-time parsing.
- String read from left to right, top-down parsing.
- The rule to apply determined by the next k input symbols.
- Parsing table: $T: N \times \Sigma^{\leqslant k} \to R$.
- LL($k+1$) grammars more powerful than LL(k).
- Greibach normal form: all rules $X \to sY_1 \dots Y_\ell$, with $s \in \Sigma$.

The simplest case: LL(1)

Theorem

There is a hardest language for LL(1) grammars in Greibach normal form.

The simplest case: LL(1)

Theorem

There is a hardest language for LL(1) grammars in Greibach normal form.

- In a rule $X \to sY_1 \dots Y_\ell$, the lookahead is s.

The simplest case: LL(1)

Theorem

There is a hardest language for LL(1) grammars in Greibach normal form.

- In a rule $X \to sY_1 \dots Y_\ell$, the lookahead is s.
- Encoded rule: $\rho(X_i \to sX_{j_1} \dots X_{j_\ell}) = a^{j_\ell} c \dots a^{j_1} c$

The simplest case: LL(1)

Theorem

There is a hardest language for LL(1) grammars in Greibach normal form.

- In a rule $X \to sY_1 \dots Y_\ell$, the lookahead is s.
- Encoded rule: $\rho(X_i \to sX_{j_1} \dots X_{j_\ell}) = a^{j_\ell} c \dots a^{j_1} c$
- Image of a symbol: $h(s) = \Big(\prod_{i=1}^{n} b\rho(T(X_i, s)) \Big) \#$

The simplest case: LL(1)

Theorem

There is a hardest language for LL(1) grammars in Greibach normal form.

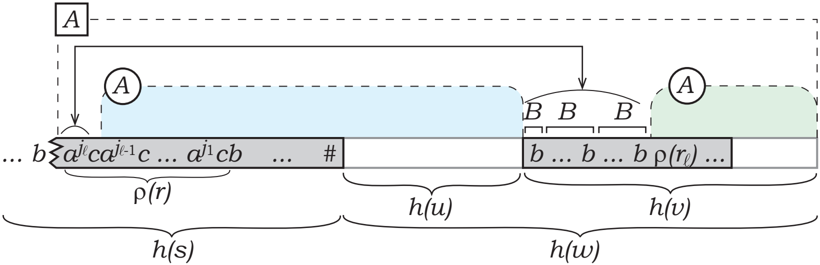
- In a rule $X \to sY_1 \dots Y_\ell$, the lookahead is s.
- Encoded rule: $\rho(X_i \to sX_{j_1} \dots X_{j_\ell}) = a^{j_\ell}c \dots a^{j_1}c$
- Image of a symbol: $h(s) = \Big(\prod_{i=1}^{n} b\rho(T(X_i, s))\Big)\#$
- a and b link nonterminals to rules

The simplest case: LL(1)

Theorem

There is a hardest language for LL(1) grammars in Greibach normal form.

- In a rule $X \to sY_1 \dots Y_\ell$, the lookahead is s.
- Encoded rule: $\rho(X_i \to sX_{j_1} \dots X_{j_\ell}) = a^{j_\ell}c \dots a^{j_1}c$
- Image of a symbol: $h(s) = \Big(\prod_{i=1}^{n} b\rho(T(X_i, s))\Big)\#$
- a and b link nonterminals to rules



Next simplest case: LL(2)

- Would this work for LL(2) grammars?..

Next simplest case: LL(2)

- Would this work for LL(2) grammars?..

Theorem

The family of LL(2) grammars in Greibach normal form cannot be reduced to a single LL(k) language by homomorphisms.

Next simplest case: LL(2)

- Would this work for LL(2) grammars?..

Theorem

The family of LL(2) grammars in Greibach normal form cannot be reduced to a single LL(k) language by homomorphisms.

- $L = \Sigma \cup \{aa \mid a \in \Sigma\}$

Next simplest case: LL(2)

- Would this work for LL(2) grammars?..

Theorem

The family of LL(2) grammars in Greibach normal form cannot be reduced to a single LL(k) language by homomorphisms.

- $L = \Sigma \cup \{aa \mid a \in \Sigma\}$
- If reducible, then both $h(a)$ and $h(aa)$ must be in L_0.

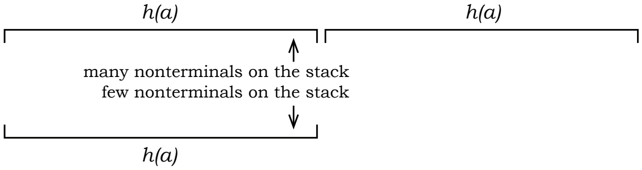
Next simplest case: LL(2)

- Would this work for LL(2) grammars?..

Theorem

The family of LL(2) grammars in Greibach normal form cannot be reduced to a single LL(k) language by homomorphisms.

- $L = \Sigma \cup \{aa \mid a \in \Sigma\}$
- If reducible, then both $h(a)$ and $h(aa)$ must be in L_0.
- LL(k) parser for L_0 after reading the k-th last symbol of $h(a)$.



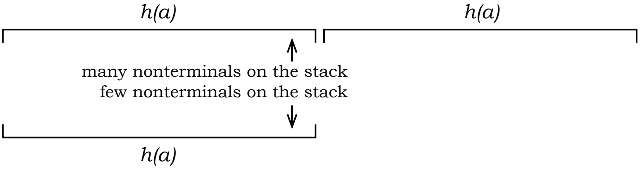
Next simplest case: LL(2)

- Would this work for LL(2) grammars?..

Theorem

The family of LL(2) grammars in Greibach normal form cannot be reduced to a single LL(k) language by homomorphisms.

- $L = \Sigma \cup \{aa \mid a \in \Sigma\}$
- If reducible, then both $h(a)$ and $h(aa)$ must be in L_0.
- LL(k) parser for L_0 after reading the k-th last symbol of $h(a)$.
  - On input $h(aa)$, must encode a in the stack.



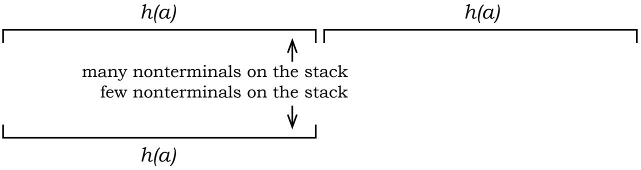
Next simplest case: LL(2)

- Would this work for LL(2) grammars?..

Theorem

The family of LL(2) grammars in Greibach normal form cannot be reduced to a single LL(k) language by homomorphisms.

- $L = \Sigma \cup \{aa \mid a \in \Sigma\}$
- If reducible, then both $h(a)$ and $h(aa)$ must be in L_0.
- LL(k) parser for L_0 after reading the k-th last symbol of $h(a)$.
  - On input $h(aa)$, must encode a in the stack.
  - On input $h(a)$, at most k symbols in the stack.



What makes LL(2) different from LL(1)?

- LL(1): a rule $X \to sY_1 \dots Y_\ell$ contains its lookahead.

What makes LL(2) different from LL(1)?

- LL(1): a rule $X \to sY_1 \dots Y_\ell$ contains its lookahead.
- LL(2): may be split between two rules.

What makes LL(2) different from LL(1)?

- LL(1): a rule $X \to sY_1 \dots Y_\ell$ contains its lookahead.
- LL(2): may be split between two rules.
- How to bufferize these lookaheads?

What makes LL(2) different from LL(1)?

- LL(1): a rule $X \to sY_1 \ldots Y_\ell$ contains its lookahead.
- LL(2): may be split between two rules.
- How to bufferize these lookaheads?
- Different queue contents enumerated.

What makes LL(2) different from LL(1)?

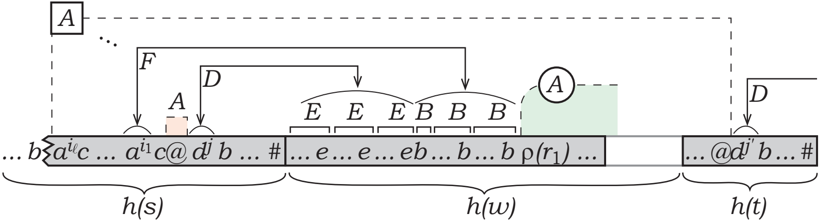
- LL(1): a rule $X \to sY_1 \dots Y_\ell$ contains its lookahead.
- LL(2): may be split between two rules.
- How to bufferize these lookaheads?
- Different queue contents enumerated.
- Queue contents listed in symbols' images.

What makes LL(2) different from LL(1)?

- LL(1): a rule $X \to sY_1 \dots Y_\ell$ contains its lookahead.
- LL(2): may be split between two rules.
- How to bufferize these lookaheads?
- Different queue contents enumerated.
- Queue contents listed in symbols' images.
- . . . and linked to each other like nonterminals.

What makes LL(2) different from LL(1)?

- LL(1): a rule $X \to sY_1 \dots Y_\ell$ contains its lookahead.
- LL(2): may be split between two rules.
- How to bufferize these lookaheads?
- Different queue contents enumerated.
- Queue contents listed in symbols' images.
- ...and linked to each other like nonterminals.



Generalized reduction

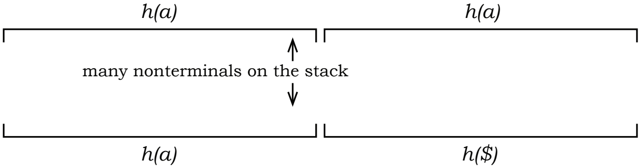
- Still cannot parse the last $k - 1$ symbols remaining.

Generalized reduction

- Still cannot parse the last $k - 1$ symbols remaining.
- Add an explicit endmarker to parse the "tail".

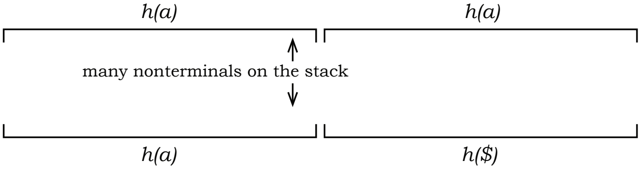
Generalized reduction

- Still cannot parse the last $k - 1$ symbols remaining.
- Add an explicit endmarker to parse the “tail”.



Generalized reduction

- Still cannot parse the last $k - 1$ symbols remaining.
- Add an explicit endmarker to parse the "tail".



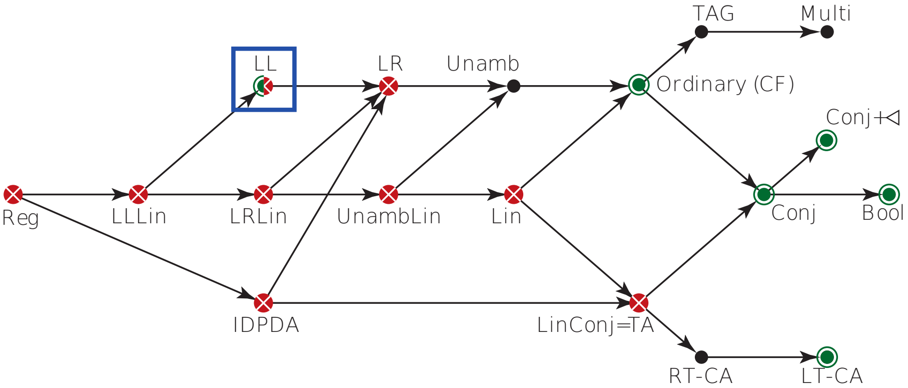
Theorem

There exists such LL(1) language L_0 that for every LL(k) language L there exists a representation of $L\$$ as $h_L^{-1}(L_0)$ for some homomorphism h_L.

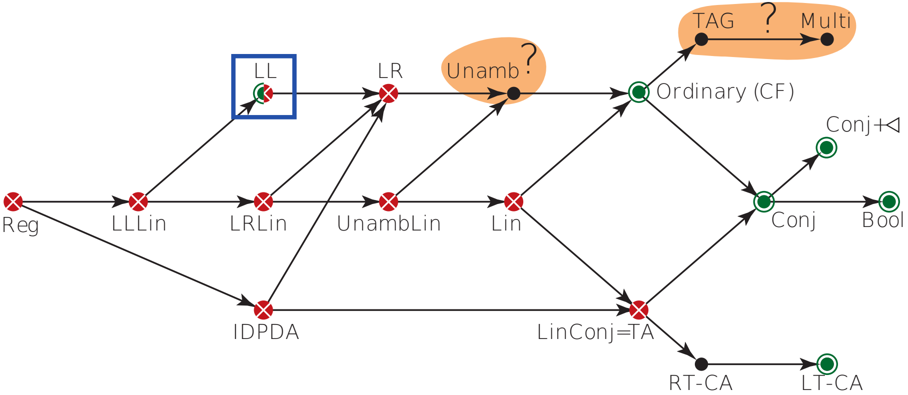
Part III

Conclusion

Potential avenues of research

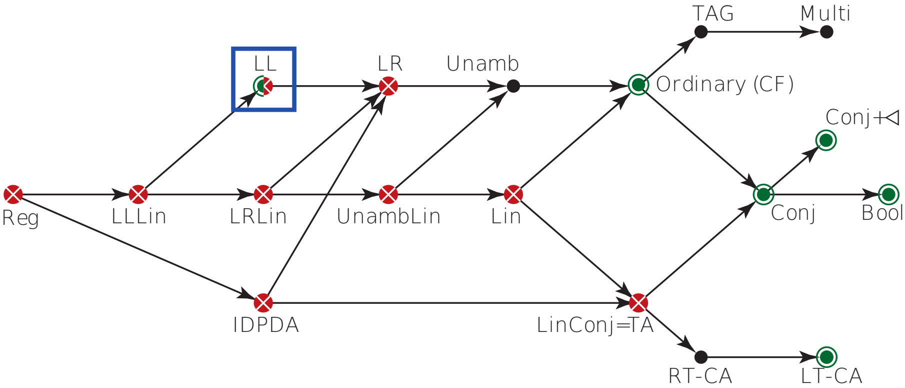


Potential avenues of research



- old problem: still open for some classic language families

Potential avenues of research



- new problem: generalized reductions for “already solved” families?

Thanks for your attention!