Parikh's theorem from the complexity viewpoint

Dmitry Chistikov

University of Warwick, United Kingdom

YR-OWLS, 03 June 2020

State complexity

Program size complexity of problem:
the minimum size of program that solves the problem

State complexity of language $\mathcal{L}$:
the minimum size of NFA that accepts $\mathcal{L}$

Why study these measures?

- We want to understand what makes problems difficult
- Programs and their models become data (e.g., in verification), hence minimization questions
- Limitations of models of computation $\Longrightarrow$ analysis algorithms

Parikh image

[Parikh (1961); in JACM (1966)]

Commutative/Parikh mapping:

$$\psi(\mathcal{L}) = \{ \ (m_1, \ldots, m_r) \colon \exists\, w \in \mathcal{L} \text{ with exactly } m_i \text{ occurrences of } a_i \ \} \subseteq \mathbb{N}^r$$

where $\Sigma = \{a_1, \ldots, a_r\}$ and $\mathcal{L} \subseteq \Sigma^*$

Examples

$$\psi\big(\{ \ a\,a\,b\,b\,b\,b\,a \ \}\big) = \{(3,4)\}$$

$$\psi\big(\{ \ a^m b^m \colon m \geq 0 \ \}\big) = \psi\big((ab)^*\big) = \{(m,m) \colon m \geq 0\}$$

Parikh's theorem

Rohit J. Parikh

Theorem

For every context-free language there exists a regular language with the same Parikh image.

Applications of Parikh's theorem

Simple applications in formal language theory:

- Unary context-free languages are regular

  [cf. Ginsburg, Rice (1962)]

- $\{a^{m^2} : m \geq 0\}$ and $\{a^{2^m} : m \geq 0\}$ are not regular

Many applications in verification of infinite-state systems!

Through the ages: Proof ideas

- Safe unpumping
  [Parikh (1966)]
- Small-index derivations
  [Esparza, Ganty, Kiefer, Luttenberger (2011)]
- Presburger description via balance and connectivity
  [Verma, Seidl, Schwentick, CADE'05]

Outline

1. Why Parikh's theorem from the complexity viewpoint?

2. One-counter languages: upper bound

3. One-counter languages: lower bound

Outline

1. Why Parikh's theorem from the complexity viewpoint?

2. One-counter languages: upper bound

3. One-counter languages: lower bound

Parikh's theorem, revisited

(from the complexity viewpoint)

Theorem

For every context-free grammar G there exists
a nondeterministic finite-state automaton $\mathcal{A}$
with at most $4^{|G|+1}$ states such that $\psi(\mathcal{L}(G)) = \psi(\mathcal{L}(\mathcal{A}))$.

Parikh's theorem: lower bound

$$\begin{aligned} A_n &\rightarrow A_{n-1}A_{n-1} \\ &\dots \\ A_4 &\rightarrow A_3A_3 \\ A_3 &\rightarrow A_2A_2 \\ A_2 &\rightarrow A_1A_1 \\ A_1 &\rightarrow a \end{aligned}$$

Nonterminal A_n generates just one word of length 2^n.
Every NFA that accepts this languages must have $> 2^n$ states.

An application: Availability languages

[Hoenicke, Meyer, Olderog, CONCUR'10]
Defined with regular expressions + following feature:

$$(\text{regexp with } \checkmark)_{\text{constraint}} :$$

"only keep w where each prefix ending with ✓ satisfies
a Presburger constraint on the number of occurrences of letters"

Language emptiness: decidable in **TOWER**
[Abdulla et al., FSTTCS'15]

An application: Availability languages

[Hoenicke, Meyer, Olderog, CONCUR'10]

$$\Big(\, (a^* b^* c^* \checkmark)_{\#a \geq \#b} \quad \checkmark \, \Big)_{\#b \geq \#c} :$$

defines $\{a^n b^m n^k : n \geq m \geq k\}$

Language emptiness: decidable in **TOWER**

[Abdulla et al., FSTTCS'15]

An application: Availability languages

[Hoenicke, Meyer, Olderog, CONCUR'10]

$$\Big(\ (a^* b^* c^* \checkmark)_{\#a \geq \#b} \quad \checkmark\ \Big)_{\#b \geq \#c}:$$

defines $\{a^n b^m n^k \colon n \geq m \geq k\}$

Language emptiness: decidable in **TOWER**

[Abdulla et al., FSTTCS'15]

An application: Availability languages

[Hoenicke, Meyer, Olderog, CONCUR'10]

$$\Big(\, (a^* b^* c^* \checkmark)_{\#a \geq \#b} \quad \checkmark \, \Big)_{\#b \geq \#c} :$$

defines $\{a^n b^m n^k \colon n \geq m \geq k\}$

Language emptiness: decidable in $\mathbf{TOWER}$

[Abdulla et al., FSTTCS'15]

An application: Availability languages

[Hoenicke, Meyer, Olderog, CONCUR'10]

$$\Big(\, \big(a^* b^* c^* \checkmark\big)_{\#a \geq \#b} \quad \checkmark \, \Big)_{\#b \geq \#c} :$$

defines $\{a^n b^m n^k \colon n \geq m \geq k\}$

Language emptiness: decidable in **TOWER**

[Abdulla et al., FSTTCS'15]

An application: Availability languages

[Hoenicke, Meyer, Olderog, CONCUR'10]

$$\Big(\ (a^*b^*c^*\checkmark)_{\#a\geq\#b}\quad\checkmark\ \Big)_{\#b\geq\#c}:$$

defines $\{a^n b^m n^k \colon n \geq m \geq k\}$

Language emptiness: decidable in $\mathbf{TOWER}$

[Abdulla et al., FSTTCS'15]

An application: Availability languages

[Hoenicke, Meyer, Olderog, CONCUR'10]

$$\Big(\ (a^*b^*c^*\checkmark)_{\#a\geq\#b}\quad\checkmark\ \Big)_{\#b\geq\#c}:$$

defines $\{a^n b^m n^k : n \geq m \geq k\}$

Language emptiness: decidable in **TOWER**

[Abdulla et al., FSTTCS'15]

An application: Availability languages

[Hoenicke, Meyer, Olderog, CONCUR'10]

$\Big((a^*b^*c^*\checkmark)_{\#a \geq \#b} \quad \checkmark \Big)_{\#b \geq \#c}$:

defines $\{a^n b^m n^k \colon n \geq m \geq k\}$

Language emptiness: decidable in **TOWER**

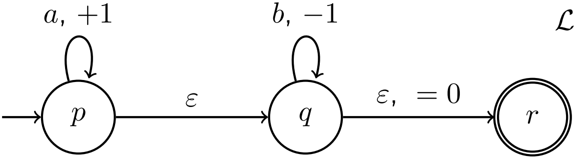
[Abdulla et al., FSTTCS'15]

Relies on NFA for Parikh image of one-counter languages.

One-counter automata (OCA)

$=$ Pushdown automata with exactly 1 non-bottom stack symbol

Example:



$\mathcal{L} = \{a^m b^m : m \geq 0\}$

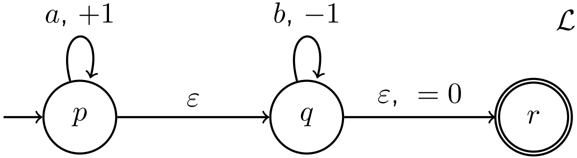
Key feature:
Non-negative integer counter that supports $+1$, -1, test for 0

Input tape: a finite word $w \in \Sigma^*$, which can be **accepted**
Language: all accepted words

One-counter automata (OCA)

= Pushdown automata with exactly 1 non-bottom stack symbol

Example:



$\mathcal{L} = \{a^m b^m : m \geq 0\}$

Regular < One-counter < Context-free languages

Separating examples: $\{a^m b^m : m \geq 0\}$, $\{ww^{\text{rev}} : w \in \Sigma^*\}$

Reasoning about OCA

Language universality is undecidable

[Valiant, 1973]

Deterministic case: language equivalence is in **PSPACE**

[Valiant and Paterson, 1973]

Deterministic case: language equivalence is **NL**-complete

[Böhm, Göller, Jančar, STOC'13]

Reasoning about OCA

Language universality is undecidable

[Valiant, 1973]

Deterministic case: language equivalence is in **PSPACE**

[Valiant and Paterson, 1973]

Deterministic case: language equivalence is **NL**-complete

[Böhm, Göller, Jančar, STOC'13]

Shortest accepted words are polynomial

[Latteux (1983)]

Parikh's theorem, revisited

(from the complexity viewpoint)

Theorem

For every context-free grammar G there exists
a nondeterministic finite-state automaton $\mathcal{A}$
with at most $4^{|G|+1}$ states such that $\psi(\mathcal{L}(G)) = \psi(\mathcal{L}(\mathcal{A}))$.

Theorem

There exists G such that $\mathcal{A}$ has to be exponentially big.

Parikh's theorem, revisited
(from the complexity viewpoint)

Theorem
For every context-free grammar G there exists
a nondeterministic finite-state automaton $\mathcal{A}$
with at most $4^{|G|+1}$ states such that $\psi(\mathcal{L}(G)) = \psi(\mathcal{L}(\mathcal{A}))$.

Theorem
There exists G such that $\mathcal{A}$ has to be exponentially big.

What if $\mathcal{L}$ is the language of a one-counter automaton?

Upper bound **remains valid**. Lower bound **fails**.

Outline

1. Why Parikh's theorem from the complexity viewpoint?

2. One-counter languages: upper bound

3. One-counter languages: lower bound

Parikh's theorem for OCL: upper bound

Atig, Chistikov, Hofman, Kumar, Saivasan, Zetzsche, LICS'16

Theorem

For every one-counter automaton $\mathcal{A}$ with n states
there exists a nondeterministic finite-state automaton $\mathcal{B}$
with at most $n^{O(\log n)}$ states such that $\psi(\mathcal{L}(\mathcal{A})) = \psi(\mathcal{L}(\mathcal{B}))$.

Proof strategy

A. Bound the number of reversals by $\mathrm{poly}(n)$

B. Transform reversal-bounded OCA into NFA

Bounding the number of reversals: ingredients

1. Process counter updates in batches:

   keep `todo` $\in [-n, n]$ in control state,
   then flush it into the counter

2. Shift around simple cycles:

   Do all increasing cycles as soon as possible.
   Do all decreasing cycles as late as possible.

Bounding the number of reversals: ingredients

1. Process counter updates in batches:

   keep `todo` $\in [-n, n]$ in control state,
   then flush it into the counter

2. Shift around simple cycles:

   Do all increasing cycles as soon as possible.
   Do all decreasing cycles as late as possible.

Claim:
Can find another OCA $\mathcal{A}'$ of size $\mathrm{poly}(n)$ such that

$$\psi(L(\mathcal{A})) = \psi(\text{runs of } \mathcal{A}' \text{ with } \mathrm{poly}(n) \text{ reversals})$$

Proof strategy

A. Bound the number of reversals by $\mathrm{poly}(n)$

B. Transform reversal-bounded OCA into NFA

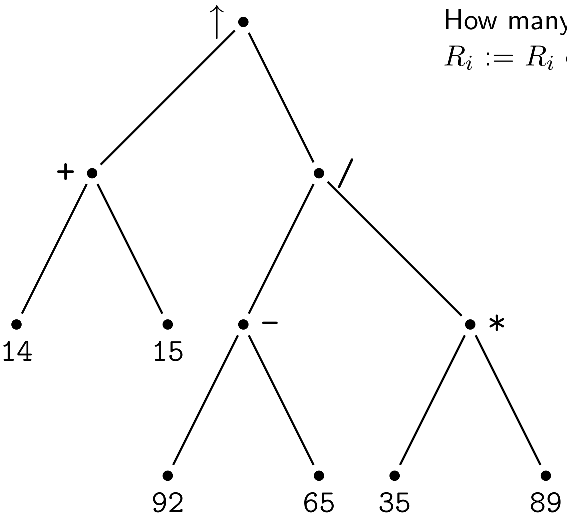
From mountains to trees

Complexity measure for trees

Intuition:

- Trees with small number of nodes are simple
- Unbalanced trees (e.g., single long branches) are simple
- Complete binary trees are complex

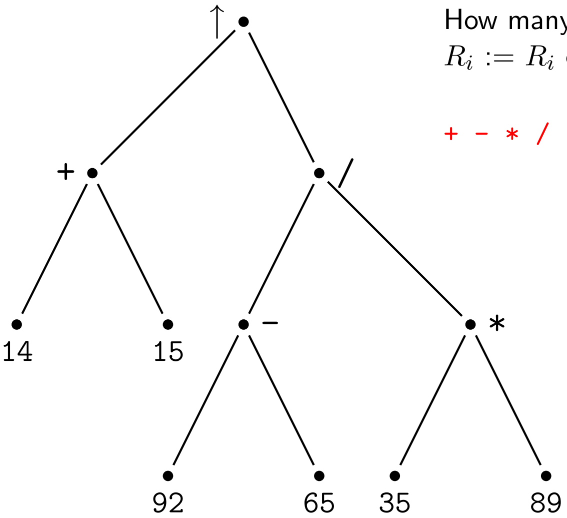
Evaluating arithmetic expressions



How many registers are needed?

$R_i := R_i \text{ op } R_j$

Evaluating arithmetic expressions

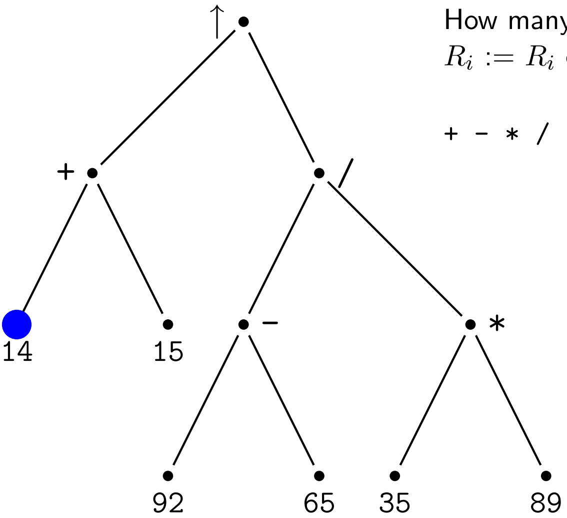


How many registers are needed?

$R_i := R_i \text{ op } R_j$

+ − * / ↑

Evaluating arithmetic expressions

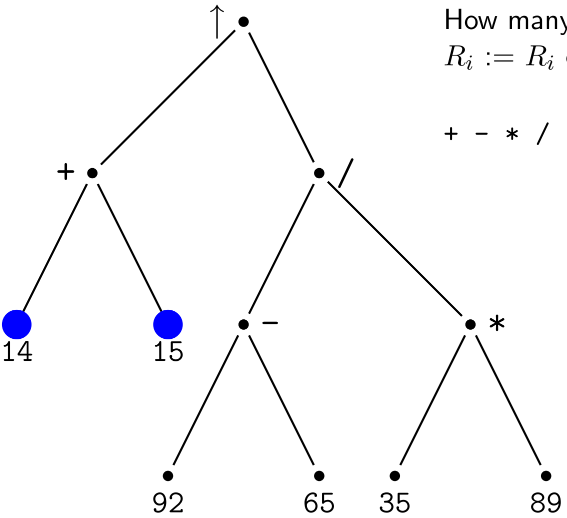


How many registers are needed?

$R_i := R_i$ op R_j

+ - * / ↑

Evaluating arithmetic expressions

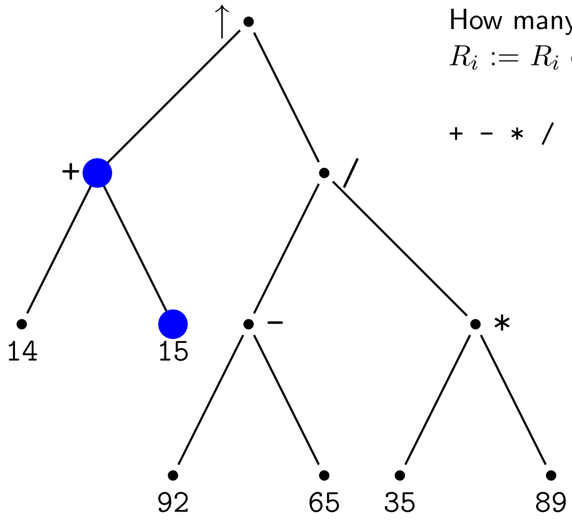


How many registers are needed?

$R_i := R_i$ op R_j

+ − * / ↑

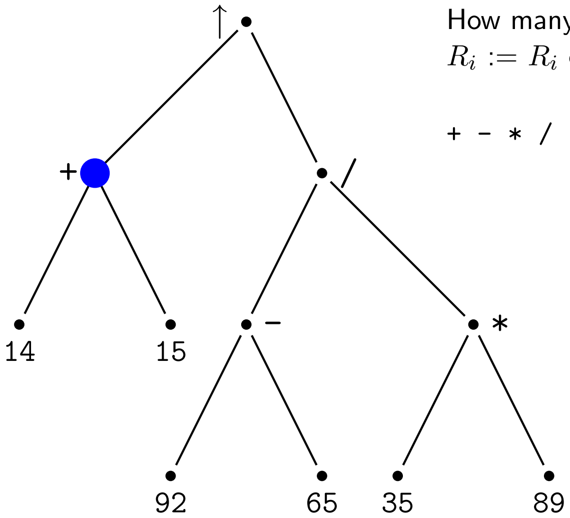
Evaluating arithmetic expressions



How many registers are needed?
$R_i := R_i$ op R_j

+ − * / ↑

Evaluating arithmetic expressions

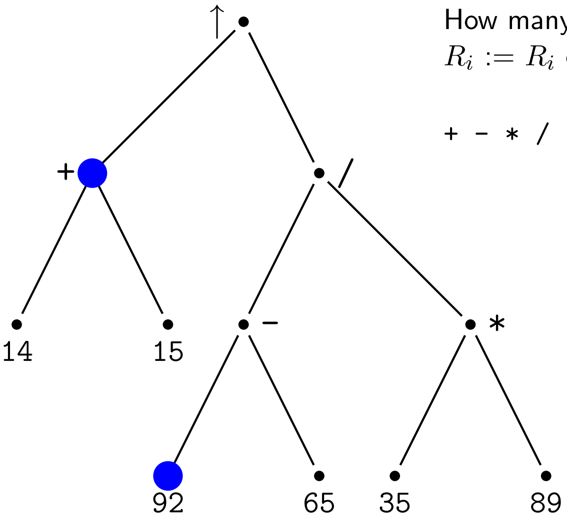


How many registers are needed?

$R_i := R_i$ op R_j

`+ - * / ↑`

Evaluating arithmetic expressions

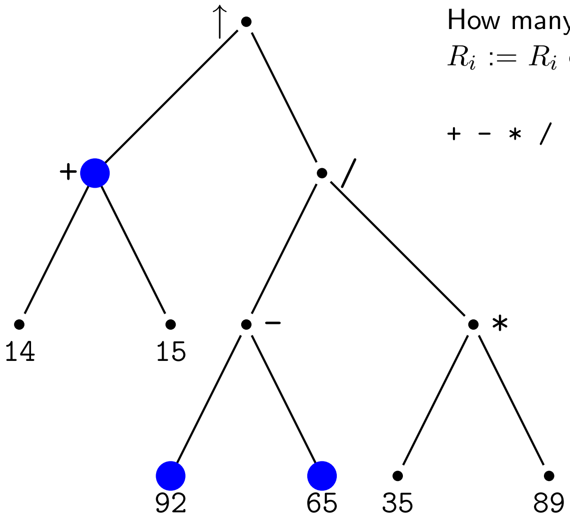


How many registers are needed?

$R_i := R_i$ op R_j

+ - * / ↑

Evaluating arithmetic expressions

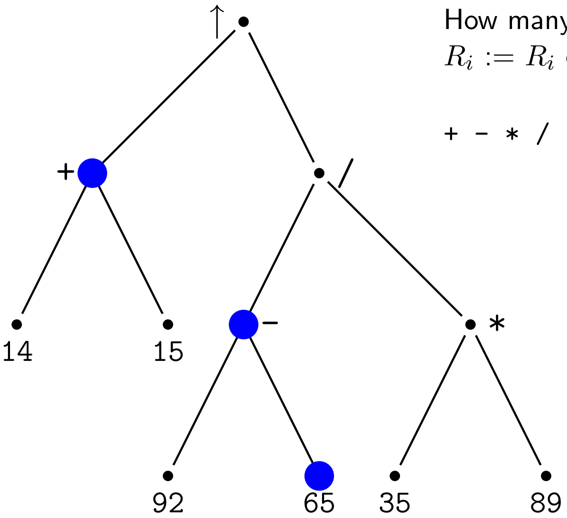


How many registers are needed?

$R_i := R_i$ op R_j

+ − * / ↑

Evaluating arithmetic expressions

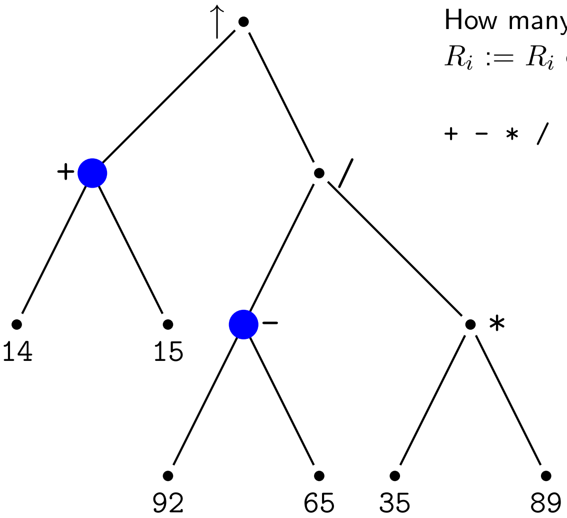


How many registers are needed?

$R_i := R_i$ op R_j

`+ - * / ↑`

Evaluating arithmetic expressions

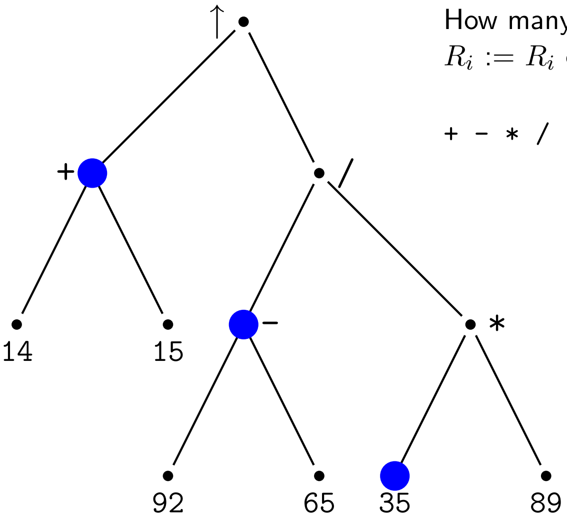


How many registers are needed?

$R_i := R_i$ op R_j

+ - * / ↑

Evaluating arithmetic expressions

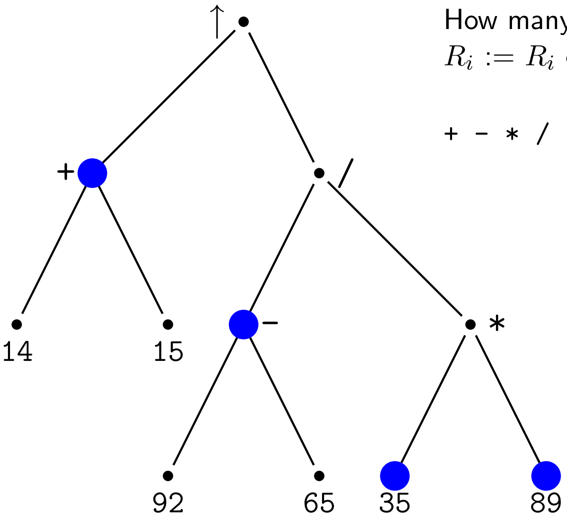


How many registers are needed?

$R_i := R_i$ op R_j

`+ - * / ↑`

Evaluating arithmetic expressions

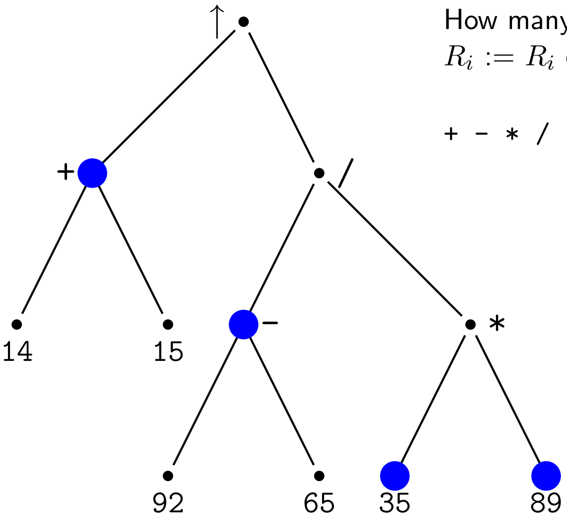


How many registers are needed?

$R_i := R_i$ op R_j

+ − * / ↑

Evaluating arithmetic expressions



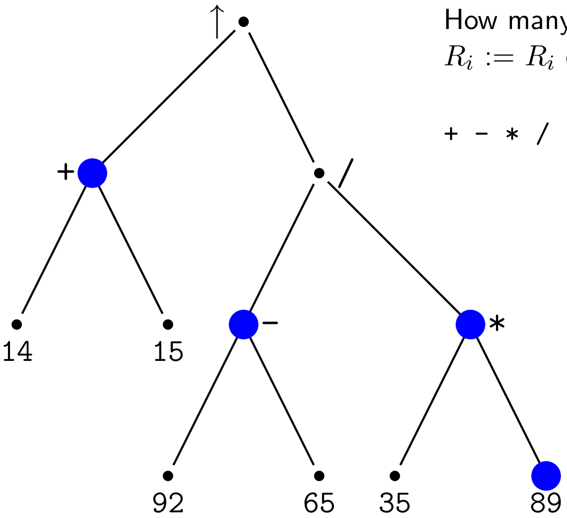
How many registers are needed?

$R_i := R_i$ op R_j

`+ - * / ↑`

4 registers

Evaluating arithmetic expressions

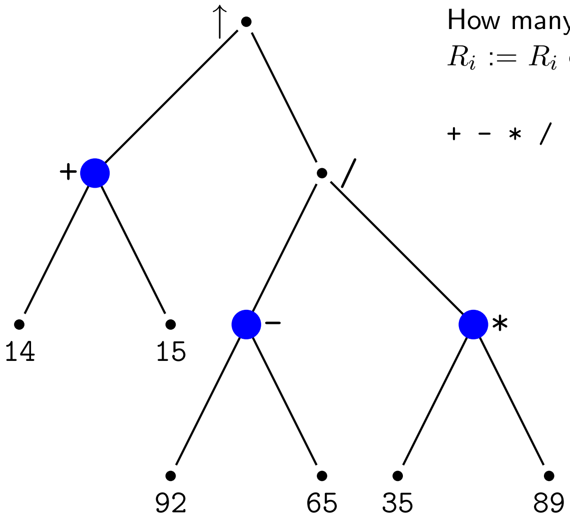


How many registers are needed?
$R_i := R_i \text{ op } R_j$

`+ - * / ↑`

4 registers

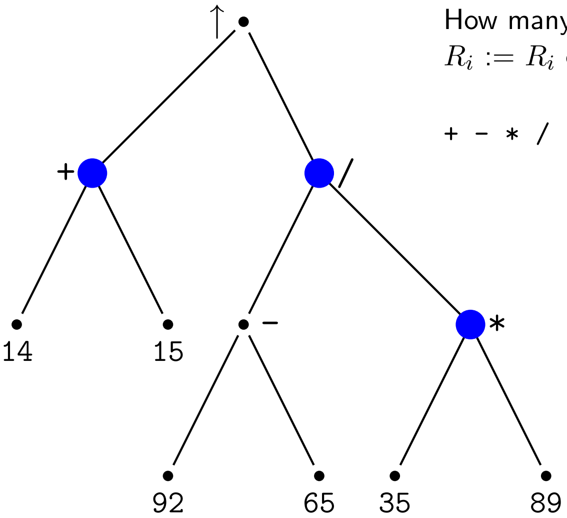
Evaluating arithmetic expressions



How many registers are needed?
$R_i := R_i \text{ op } R_j$

`+ - * / ↑` 4 registers

Evaluating arithmetic expressions



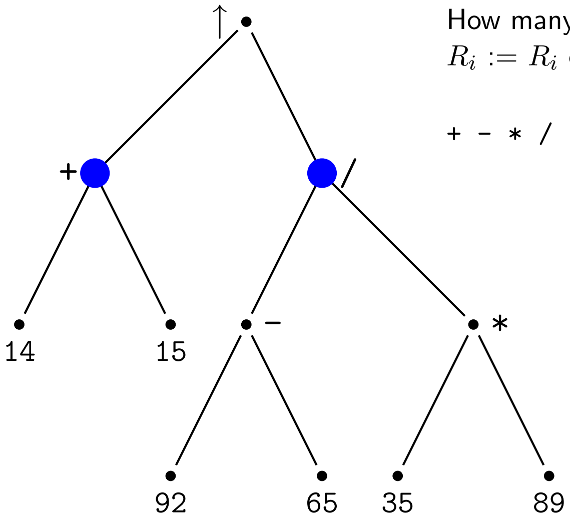
How many registers are needed?

$R_i := R_i \text{ op } R_j$

`+ - * / ↑`

4 registers

Evaluating arithmetic expressions

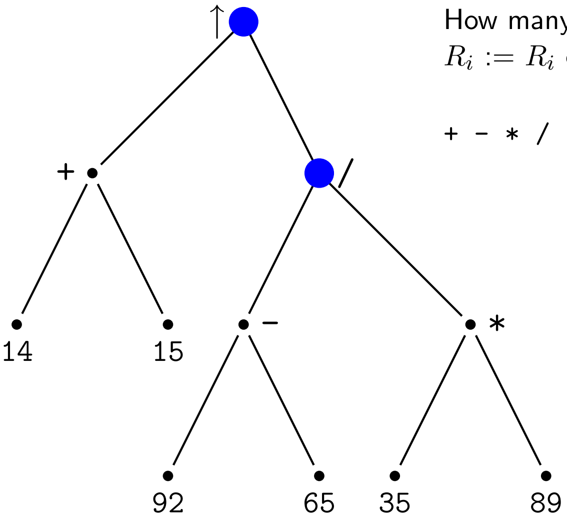


How many registers are needed?

$R_i := R_i \text{ op } R_j$

`+ - * / ↑` 4 registers

Evaluating arithmetic expressions

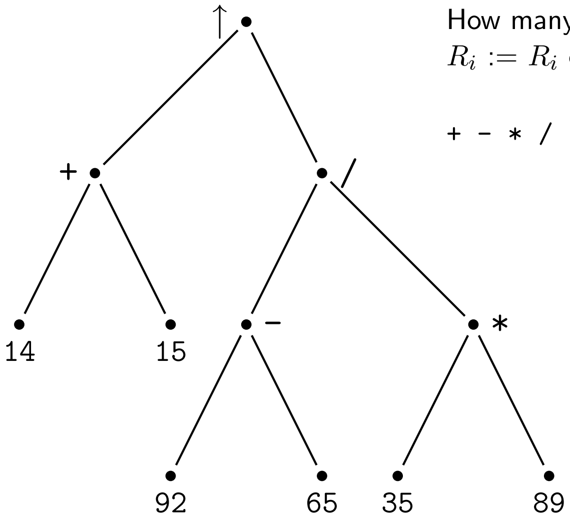


How many registers are needed?

$R_i := R_i$ op R_j

`+ - * / ↑` 4 registers

Evaluating arithmetic expressions

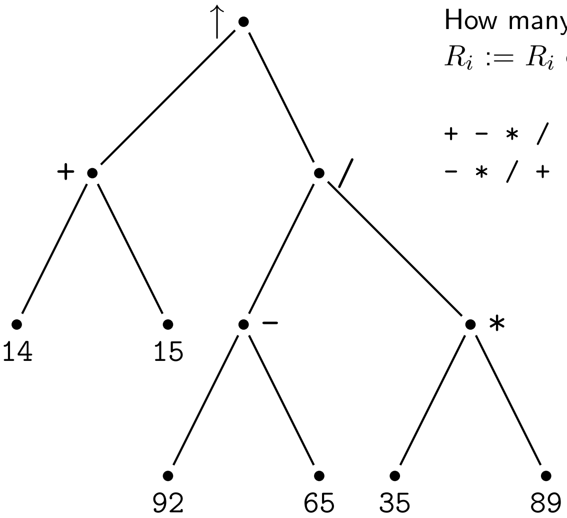


How many registers are needed?

$R_i := R_i$ op R_j

+ − * / ↑ 4 registers

Evaluating arithmetic expressions



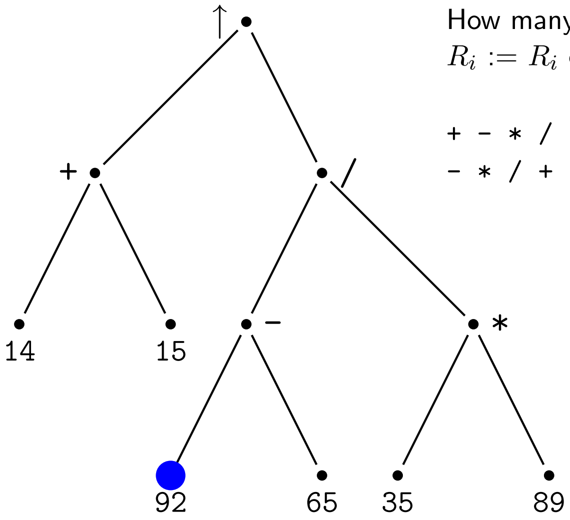
How many registers are needed?

$R_i := R_i$ op R_j

`+ - * / ↑` 4 registers

`- * / + ↑` 3 registers

Evaluating arithmetic expressions



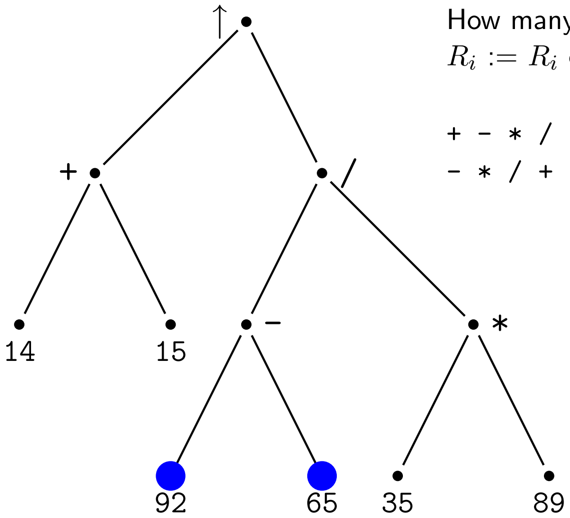
How many registers are needed?

$R_i := R_i \text{ op } R_j$

+ − * / ↑ 4 registers

− * / + ↑ 3 registers

Evaluating arithmetic expressions



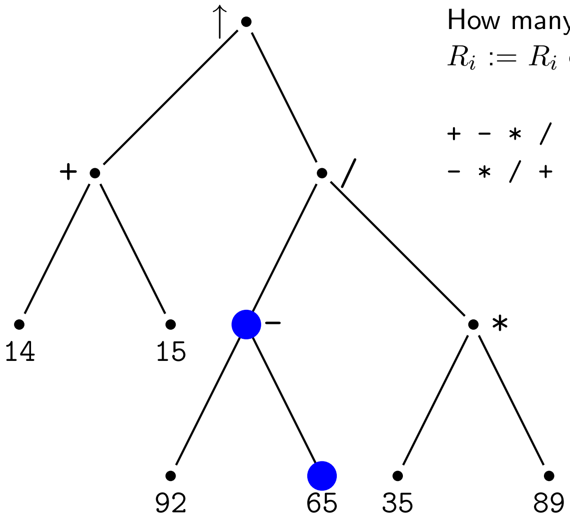
How many registers are needed?

$R_i := R_i$ op R_j

`+ - * / ↑` 4 registers

`- * / + ↑` 3 registers

Evaluating arithmetic expressions

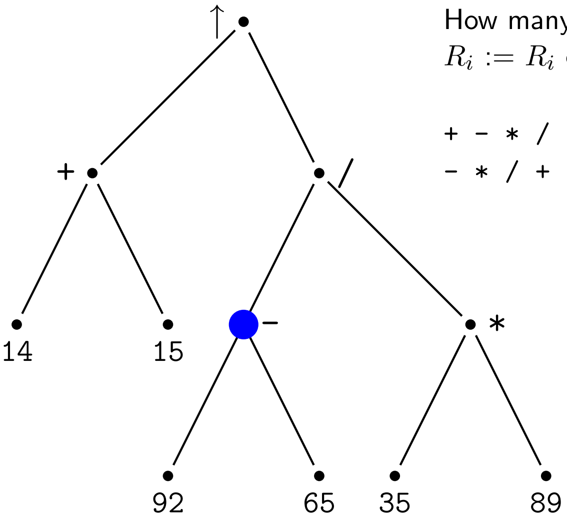


How many registers are needed?

$R_i := R_i$ op R_j

`+ - * / ↑`	4 registers
`- * / + ↑`	3 registers

Evaluating arithmetic expressions



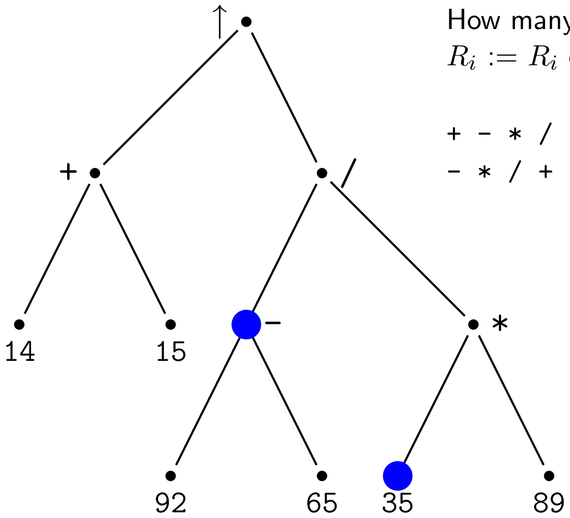
How many registers are needed?

$R_i := R_i$ op R_j

`+ - * / ↑` 4 registers

`- * / + ↑` 3 registers

Evaluating arithmetic expressions



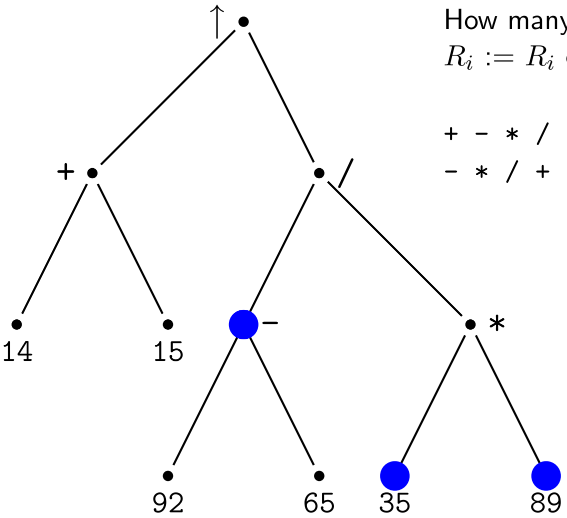
How many registers are needed?

$R_i := R_i \text{ op } R_j$

+ − * / ↑ 4 registers

− * / + ↑ 3 registers

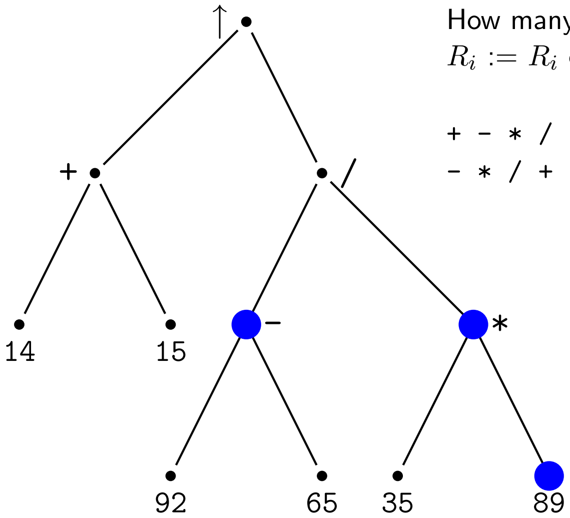
Evaluating arithmetic expressions



How many registers are needed?
$R_i := R_i \text{ op } R_j$

`+ - * / ↑` 4 registers
`- * / + ↑` 3 registers

Evaluating arithmetic expressions



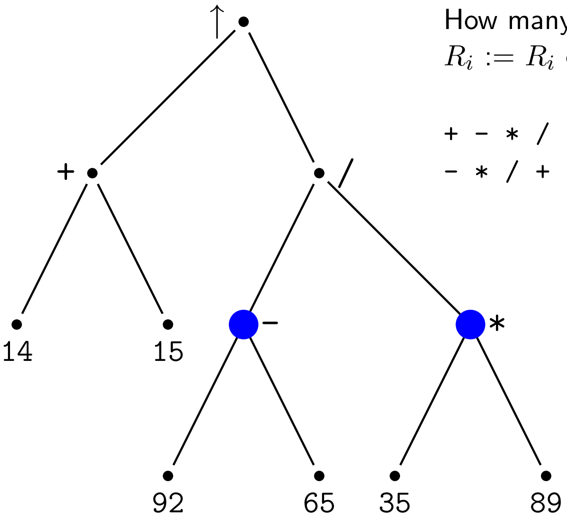
How many registers are needed?

$R_i := R_i$ op R_j

`+ - * / ↑` 4 registers

`- * / + ↑` 3 registers

Evaluating arithmetic expressions



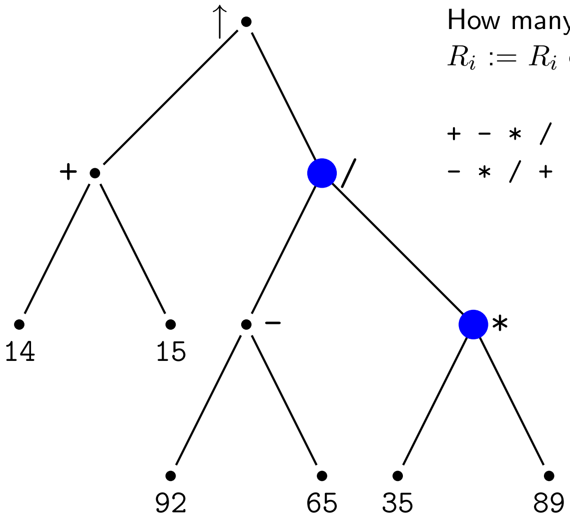
How many registers are needed?

$R_i := R_i \text{ op } R_j$

`+ - * / ↑` 4 registers

`- * / + ↑` 3 registers

Evaluating arithmetic expressions



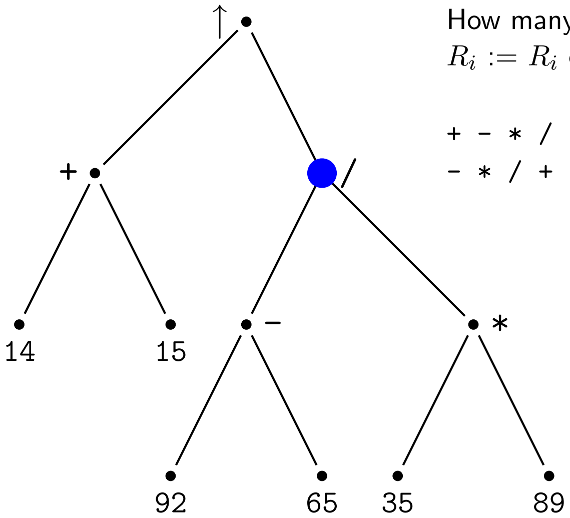
How many registers are needed?

$R_i := R_i \text{ op } R_j$

`+ - * / ↑` 4 registers

`- * / + ↑` 3 registers

Evaluating arithmetic expressions

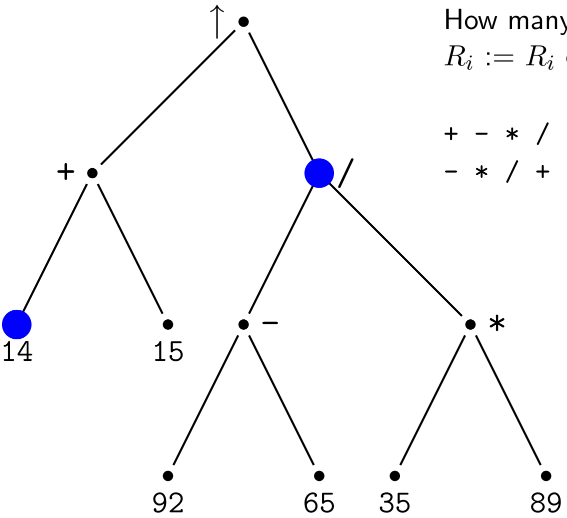


How many registers are needed?

$R_i := R_i \text{ op } R_j$

+ − * / ↑	4 registers
− * / + ↑	3 registers

Evaluating arithmetic expressions

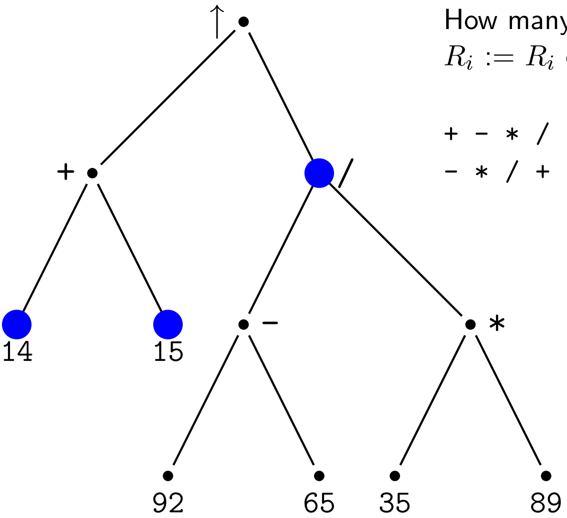


How many registers are needed?

$R_i := R_i$ op R_j

`+ - * / ↑`	4 registers
`- * / + ↑`	3 registers

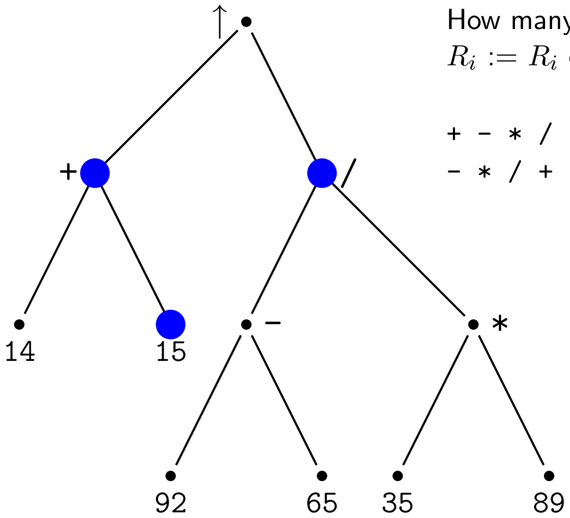
Evaluating arithmetic expressions



How many registers are needed?
$R_i := R_i \text{ op } R_j$

`+ - * / ↑` 4 registers
`- * / + ↑` 3 registers

Evaluating arithmetic expressions



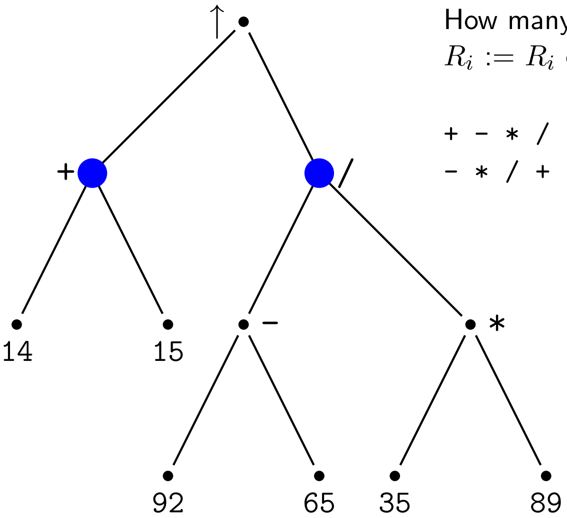
How many registers are needed?

$R_i := R_i$ op R_j

+ − * / ↑ 4 registers

− * / + ↑ 3 registers

Evaluating arithmetic expressions



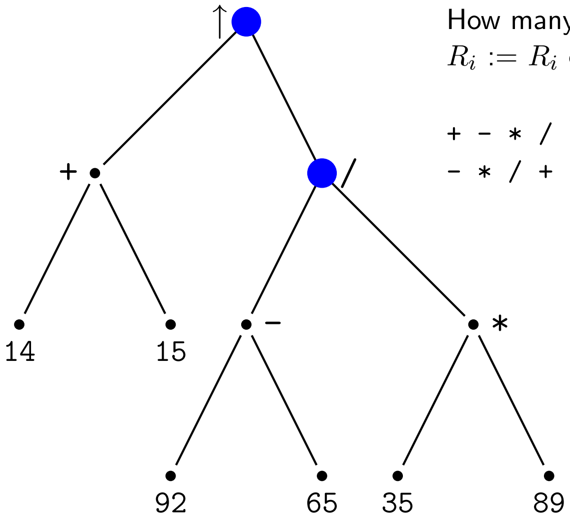
How many registers are needed?

$R_i := R_i \text{ op } R_j$

+ − * / ↑ 4 registers

− * / + ↑ 3 registers

Evaluating arithmetic expressions



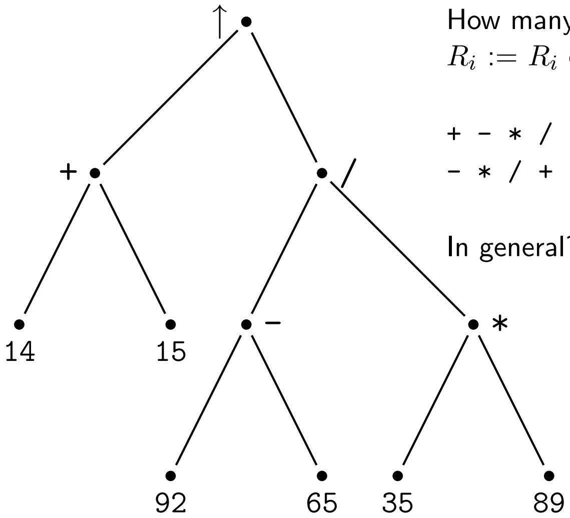
How many registers are needed?

$R_i := R_i$ op R_j

+ − * / ↑ 4 registers

− * / + ↑ 3 registers

Evaluating arithmetic expressions



How many registers are needed?

$R_i := R_i$ op R_j

`+ - * / ↑` 4 registers

`- * / + ↑` 3 registers

In general?

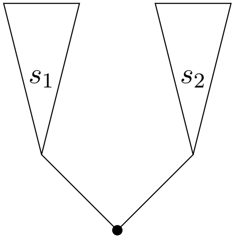
Smallest number of registers

= black pebbling number
= 1 + Strahler number
= 1 + max height of an embedded complete binary tree

[Horton (1945), Strahler (1952), Ershov (1958)]
[survey: Esparza et al., LATA'14]

Strahler number $s(\text{tree})$:

$$\bullet \mapsto 0$$



$$\mapsto \begin{cases} \max(s_1, s_2), & s_1 \neq s_2 \\ \max(s_1, s_2) + 1, & s_1 = s_2 \end{cases}$$

Putting things together: obligations

New NFA $\mathcal{B}$ guesses a tree with $\mathrm{poly}(n)$ leaves:

- The tree is traversed from root to leaves
- Whenever $\mathcal{B}$ **does not enter** a subtree, it records obligation on the stack
- Obligations are discharged later

Putting things together: obligations

New NFA $\mathcal{B}$ guesses a tree with $\mathrm{poly}(n)$ leaves:

- The tree is traversed from root to leaves
- Whenever $\mathcal{B}$ **does not enter** a subtree, it records obligation on the stack
- Obligations are discharged later

For a good strategy, $O(\log n)$ obligations suffice (Strahler!).

There are $\mathrm{poly}(n)$ possible obligations.

Transforming stack of height $O(\log n)$ to NFA: $n^{O(\log n)}$ states.

Parikh's theorem for OCL: upper bound

Atig, Chistikov, Hofman, Kumar, Saivasan, Zetzsche, LICS'16

Theorem

For every one-counter automaton $\mathcal{A}$ with n states
there exists a nondeterministic finite-state automaton $\mathcal{B}$
with at most $n^{O(\log n)}$ states such that $\psi(\mathcal{L}(\mathcal{A})) = \psi(\mathcal{L}(\mathcal{B}))$.

Outline

1. Why Parikh's theorem from the complexity viewpoint?
2. One-counter languages: upper bound
3. One-counter languages: lower bound

Parikh's theorem for OCL: lower bound

Chistikov, Vyalyi, LICS'20

Theorem

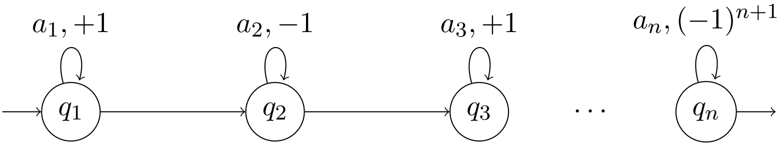
There exists a one-counter automaton $\mathcal{A}$ with n states such that every nondeterministic finite-state automaton $\mathcal{B}$ with $\psi(\mathcal{L}(\mathcal{A})) = \psi(\mathcal{L}(\mathcal{B}))$ has size

$$n^{\Omega(\sqrt{\log n / \log\log n})}.$$

Recall the upper bound:

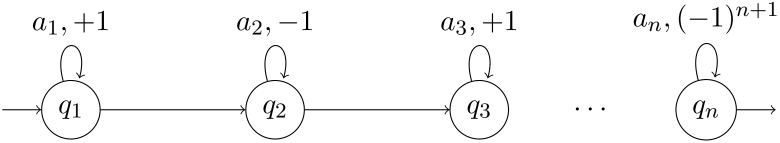
$$n^{O(\log n)}$$

Proof attempt: many trees to remember?



$\texttt{up}^* \ \texttt{down}^* \ \texttt{up}^* \ \texttt{down}^* \ \texttt{up}^* \ \texttt{down}^*$

Proof attempt: many trees to remember?



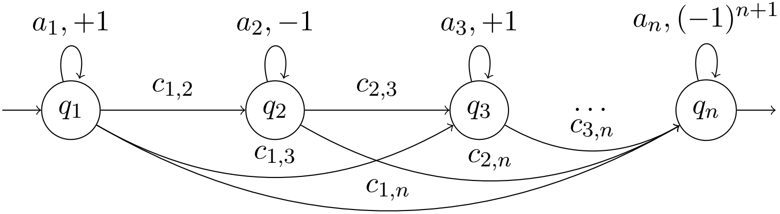
$\texttt{up}^* \ \texttt{down}^* \ \texttt{up}^* \ \texttt{down}^* \ \texttt{up}^* \ \texttt{down}^*$

For $n = 6$, accepts words $a_1^{\ell_1} a_2^{\ell_2} a_3^{\ell_3} a_4^{\ell_4} a_5^{\ell_5} a_6^{\ell_6}$ such that:

- $\ell_1 - \ell_2 \geq 0$
- $\ell_1 - \ell_2 + \ell_3 - \ell_4 \geq 0$
- $\ell_1 - \ell_2 + \ell_3 - \ell_4 + \ell_5 - \ell_6 = 0$

NFA can ignore trees: $(a_1 a_2)^*(a_1 a_4)^*(a_1 a_6)^*(a_3 a_4)^*(a_3 a_6)^*(a_5 a_6)^*$

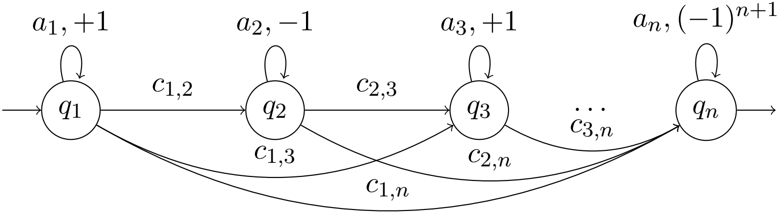
Another attempt: many subsets of states to remember?



A variant of this OCA is provably the hardest example.

[Atig et al., LICS'16]

Another attempt: many subsets of states to remember?



A variant of this OCA is provably the hardest example.

[Atig et al., LICS'16]

What's happening for each subset?

Re-pairing problem

Defined for Dyck words

$$(\; (\;) \; (\;) \;)$$

Re-pairing problem

Defined for Dyck words over $\{+, -\}$

$$+ + - + - -$$

Re-pairing problem

Defined for Dyck words over $\{+, -\}$

$$+ + - + - -$$

Move: erase any pair of $+$ and $-$ such that $+$ is to the left of $-$

Re-pairing problem

Defined for Dyck words over $\{+, -\}$

$$+ \qquad + - -$$

Move: erase any pair of $+$ and $-$ such that $+$ is to the left of $-$

Re-pairing problem

Defined for Dyck words over $\{+, -\}$

$$+ \quad\quad + - -$$

Move: erase any pair of $+$ and $-$ such that $+$ is to the left of $-$

General goal: erase everything

Re-pairing problem

Defined for Dyck words over $\{+, -\}$

$$+ \qquad -$$

Move: erase any pair of $+$ and $-$ such that $+$ is to the left of $-$

General goal: erase everything

Re-pairing problem

Defined for Dyck words over $\{+, -\}$

Move: erase any pair of $+$ and $-$ such that $+$ is to the left of $-$

General goal: erase everything

Re-pairing problem

Defined for Dyck words over $\{+, -\}$

Move: erase any pair of $+$ and $-$ such that $+$ is to the left of $-$

General goal: erase everything

Objective: minimize the maximum width seen during the play

Width $=$ number of 'islands' of signs separated by blank space

Re-pairing problem

Defined for Dyck words over $\{+, -\}$

$$+ + - + - -$$

width: $1 \rightarrow$

Move: erase any pair of $+$ and $-$ such that $+$ is to the left of $-$

General goal: erase everything

Objective: minimize the maximum width seen during the play

Width $=$ number of 'islands' of signs separated by blank space

Re-pairing problem

Defined for Dyck words over $\{+, -\}$

$$+ \qquad + - -$$

width: $1 \rightarrow 2 \rightarrow$

Move: erase any pair of $+$ and $-$ such that $+$ is to the left of $-$

General goal: erase everything

Objective: minimize the maximum width seen during the play

Width $=$ number of ‘islands’ of signs separated by blank space

Re-pairing problem

Defined for Dyck words over $\{+, -\}$

$$+ \qquad\qquad -$$

width: $1 \to 2 \to 2 \to$

Move: erase any pair of $+$ and $-$ such that $+$ is to the left of $-$

General goal: erase everything

Objective: minimize the maximum width seen during the play

Width $=$ number of 'islands' of signs separated by blank space

Re-pairing problem

Defined for Dyck words over $\{+,-\}$

width: $1 \to 2 \to 2 \to 0$

Move: erase any pair of $+$ and $-$ such that $+$ is to the left of $-$

General goal: erase everything

Objective: minimize the maximum width seen during the play

Width $=$ number of 'islands' of signs separated by blank space

Re-pairing problem

Defined for Dyck words over $\{+, -\}$

width: $1 \rightarrow 2 \rightarrow 2 \rightarrow 0$

Move: erase any pair of $+$ and $-$ such that $+$ is to the left of $-$

General goal: erase everything

Objective: minimize the maximum width seen during the play

Width $=$ number of 'islands' of signs separated by blank space

Re-pairing problem

Defined for Dyck words over $\{+, -\}$

Width of this re-pairing $=$ **2**

Move: erase any pair of $+$ and $-$ such that $+$ is to the left of $-$

General goal: erase everything

Objective: minimize the maximum width seen during the play

Width $=$ number of 'islands' of signs separated by blank space

Re-pairing problem

Defined for Dyck words over $\{+, -\}$

$$+ + - + - -$$

Width of this re-pairing = **2**

Move: erase any pair of $+$ and $-$ such that $+$ is to the left of $-$

General goal: erase everything

Objective: minimize the maximum width seen during the play

Width $=$ number of 'islands' of signs separated by blank space

Minimizing width of re-pairings

The width of a Dyck word is the minimum width of its re-pairings.

Minimizing width of re-pairings

The width of a Dyck word is the minimum width of its re-pairings.

Do all Dyck words have re-pairings of width ≤ 2020 ?

Minimizing width of re-pairings

The width of a Dyck word is the minimum width of its re-pairings.

Do all Dyck words have re-pairings of width ≤ 2020 ?

Can we prove lower bounds on the width?

Width of words and NFA size: strategy

1. There are sequences of words with unbounded width:

$$\text{width}(Y_n) \to \infty$$

2. Lower bounds on width imply lower bounds on NFA size:

$$n^{\Omega(\text{width}(w_n))}$$

Simple re-pairings

1. Every Dyck word w has a re-pairing of width $O(\log |w|)$.

Simple re-pairings

1. Every Dyck word w has a re-pairing of width $O(\log |w|)$.
   This re-pairing is simple: always pairs up matching signs.

Simple re-pairings

1. Every Dyck word w has a re-pairing of width $O(\log |w|)$.
   This re-pairing is simple: always pairs up matching signs.

2. For simple re-pairings, we know the optimal width
   up to a multiplicative constant.

   For Dyck words associated with binary trees:
   height of the largest complete binary tree that is a minor
   (Strahler number, tree dimension).

   Technique: black-and-white pebble games.
   [Lengauer and Tarjan (1980)]

How powerful are simple re-pairings?

$$\underbrace{++\ldots++}_{k}\,w\,\underbrace{--\ldots--}_{k}.$$

How powerful are simple re-pairings?

Not very powerful: The width of

$$\underbrace{+ + \ldots + +}_{k} w \underbrace{- - \ldots - -}_{k}.$$

is at most 2 if $k \geq |w|/2$.

But w can have big complete binary subtrees.

$\Longrightarrow$ Growing gap between simple and non-simple re-pairings

Width of words and NFA size: results

1. There are sequences of words with unbounded width

$$\mathsf{width}(Y_n) = \Omega(\sqrt{\log n / \log\log n})$$

2. This implies lower bounds on NFA size:

$$n^{\Omega(\sqrt{\log n / \log\log n})}$$

Parikh's theorem for OCL: lower bound

Chistikov, Vyalyi, LICS'20

Theorem

There exists a one-counter automaton $\mathcal{A}$ with n states such that every nondeterministic finite-state automaton $\mathcal{B}$ with $\psi(\mathcal{L}(\mathcal{A})) = \psi(\mathcal{L}(\mathcal{B}))$ has size

$$n^{\Omega(\sqrt{\log n / \log\log n})}.$$

Recall the upper bound:

$$n^{O(\log n)}$$

State complexity

Program size complexity of problem:
the minimum size of program that solves the problem

State complexity of language $\mathcal{L}$:
the minimum size of NFA that accepts $\mathcal{L}$

Why study these measures?

- We want to understand what makes problems difficult
- Programs and their models become data (e.g., in verification), hence minimization questions
- Limitations of models of computation $\Longrightarrow$ analysis algorithms

Thank you!

`http://warwick.ac.uk/chdir`