On the Semantics of Regular Expression Parsing In the Wild

Brink van der Merwe[1]
Martin Berglund[2]

August 18, 2015

[1]Stellenbosch University, abvdm@cs.sun.ac.za
[2]Umeå University?, mbe@cs.umu.se

Overview and Context

From the pure theoretical perspective this covers:

Prioritized (non-det) finite automata (pNFA), an automata model *syntactically constraining* computation paths, leading to *unambiguity*.

The corresponding definition of *prioritized transducers* (pTr) similarly constrained, these are *functional*.

Algorithms for *equivalence checking* and efficient *parsing* are given.
A useful normal form is given in the process.

Overview and Context

From the pure theoretical perspective this covers:

Prioritized (non-det) finite automata (pNFA), an automata model *syntactically constraining* computation paths, leading to *unambiguity*.

The corresponding definition of *prioritized transducers* (pTr) similarly constrained, these are *functional*.

Algorithms for *equivalence checking* and efficient *parsing* are given.
A useful normal form is given in the process.

The practical context is the core motivation however: modelling the semantics of *real-world regular expressions*. *Analyzing Catastrophic Backtracking Behavior in Practical Regular Expression Matching* (B.v.d. Merwe, F. Drewes, M. Berglund) at AFL'14 preshadowed pNFA/pTr.

Twisted Practice: Non-Determinism and Determinism

The theory provided membership tests ($w \in \mathcal{L}(E)$?), but in practice applications want *parsing*:

```
w =~ /(.*)@umu.se|(.*)@(.*)/
if ($1)
    ...check umu username
else
    ...some non-umu domain in $3
```

(Also, DFA matchers are usually unacceptably large)

Twisted Practice: Non-Determinism and Determinism

The theory provided membership tests ($w \in \mathcal{L}(E)$?), but in practice applications want *parsing*:

```
w =~ /(.*)@umu.se|(.*)@(.*)/
if ($1)
    ...check umu username
else
    ...some non-umu domain in $3
```

(Also, DFA matchers are usually unacceptably large)

Non-trivial regular expressions are usually ambiguous, and non-determinism must not leak! Laziness/promises? Parse forests? *Choosing one parse in a deterministic way.*

Twisted Practice: Non-Determinism and Determinism

The theory provided membership tests ($w \in \mathcal{L}(E)$?), but in practice applications want *parsing*:

```
w =~ /(.*)@umu.se|(.*)@(.*)/
if ($1)
    ...check umu username
else
    ...some non-umu domain in $3
```

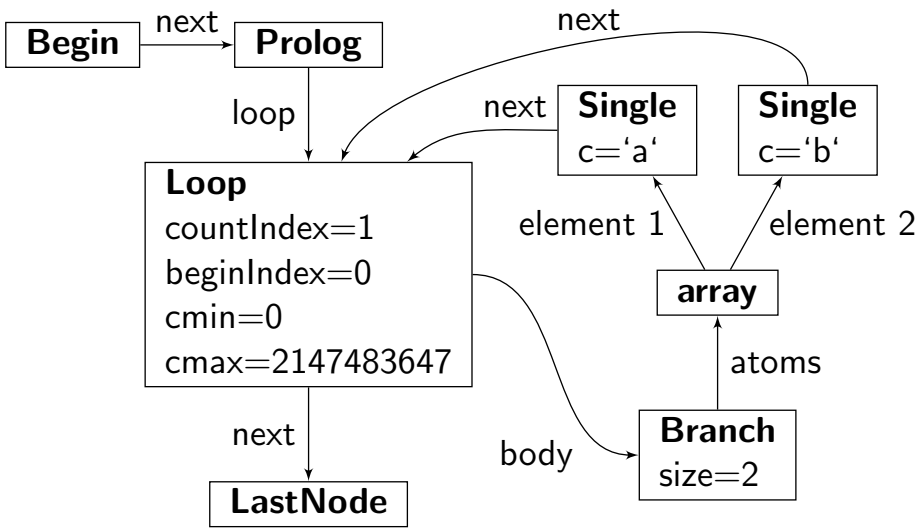
(Also, DFA matchers are usually unacceptably large)

Non-trivial regular expressions are usually ambiguous, and non-determinism must not leak! Laziness/promises? Parse forests? *Choosing one parse in a deterministic way.*

UNIX tools popularized *greedy semantics*: match as much as possible as early as possible. No formal specification. POSIX is *sort of* specified, but is uncommon and inconsistently interpreted

The Greedy Perl Semantics

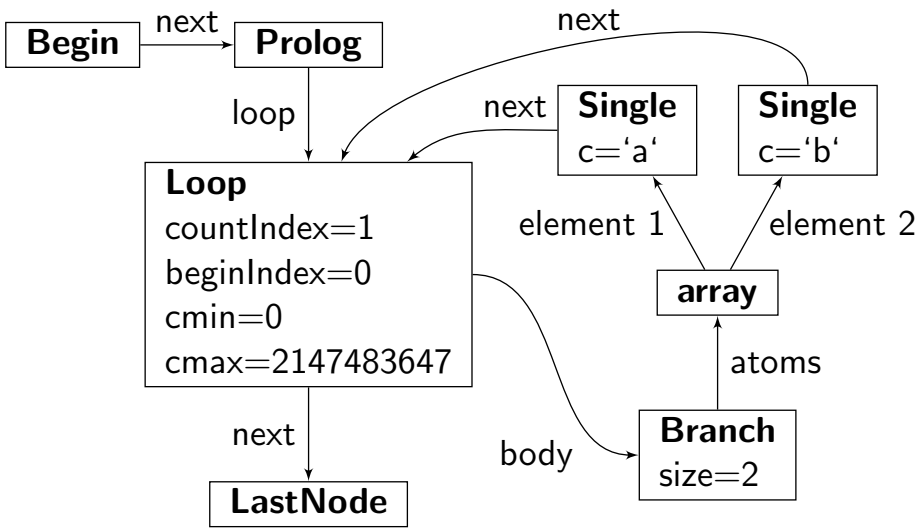
Regular expression semantics are primarily defined by the standard implementation technique: *depth-first*[3] *backtracking search* on some internal representation of the regex. $(a|b)^*$ in Java below



[3] The definition of which is not well agreed upon.

The Greedy Perl Semantics

Regular expression semantics are primarily defined by the standard implementation technique: *depth-first*[3] *backtracking search* on some internal representation of the regex. $(a|b)^*$ in Java below



Many new operators intertwined with this implementation details are now de facto standard, for example *atomic subgroups*..

Common implementations (Java, PCRE, etc.) are dangerously slow (*catastrophic backtracking*), unspecified, and are often inconsistent

[3] The definition of which is not well agreed upon.

The Model

We define prioritized non-deterministic finite automata (pNFA) to capture backtracking-style semantics in a formal way:

$$A = (Q_1, Q_2, \Sigma, q_0, \delta_1, \delta_2, F)$$

- Q_1 and Q_2 are *finite sets of states*, let $Q := Q_1 \cup Q_2$,
- Σ is the *input alphabet*,
- $q_0 \in Q$ is the *initial state*,
- $\delta_1 : Q_1 \times \Sigma \to Q$ is the *deterministic transition* function,
- $\delta_2 : Q_2 \to Q^*$ is the *prioritized non-det. transition* function,
- $F \subseteq Q_1$ are the final states.

A can be collapsed into a language-equivalent NFA by treating δ_2 as collections of ε transitions.

The Model

We define prioritized non-deterministic finite automata (pNFA) to capture backtracking-style semantics in a formal way:

$$A = (Q_1, Q_2, \Sigma, q_0, \delta_1, \delta_2, F)$$

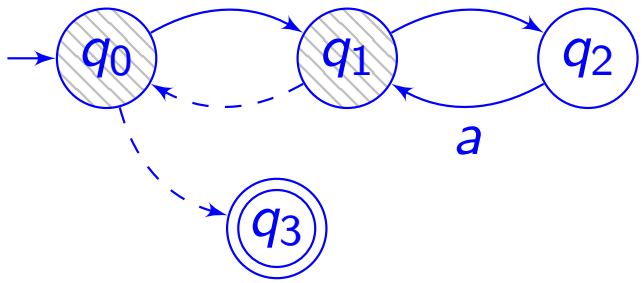
- Q_1 and Q_2 are *finite sets of states*, let $Q := Q_1 \cup Q_2$,
- Σ is the *input alphabet*,
- $q_0 \in Q$ is the *initial state*,
- $\delta_1 : Q_1 \times \Sigma \to Q$ is the *deterministic transition* function,
- $\delta_2 : Q_2 \to Q^*$ is the *prioritized non-det. transition* function,
- $F \subseteq Q_1$ are the final states.

A can be collapsed into a language-equivalent NFA by treating δ_2 as collections of ε transitions.

The purpose: δ_2 dictates the *priority* of each otherwise non-deterministic choice. The *accepting* computation path for a string is the *highest priority* computation path for that string.

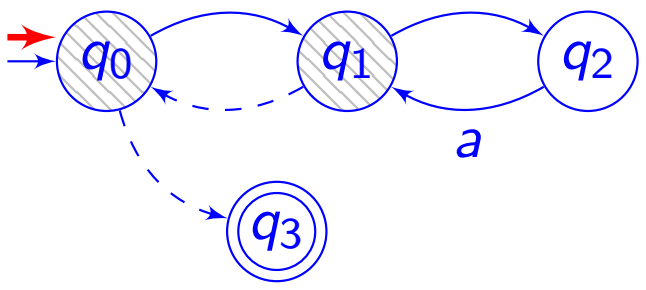
Examples: pNFA semantics and Java behavior

The pNFA for Javas view of $(a*)*$ is $A = (\{q_2, q_3\}, \{q_0, q_1\}, \{a\}, q_0, \{(q_2, a, q_1)\}, \{[q_0, (q_1, q_3)], [q_1, (q_2, q_0)]\}, q_3)$.



Examples: pNFA semantics and Java behavior

The pNFA for Javas view of $(a*)*$ is $A = (\{q_2, q_3\}, \{q_0, q_1\}, \{a\}, q_0, \{(q_2, a, q_1)\}, \{[q_0, (q_1, q_3)], [q_1, (q_2, q_0)]\}, q_3)$.

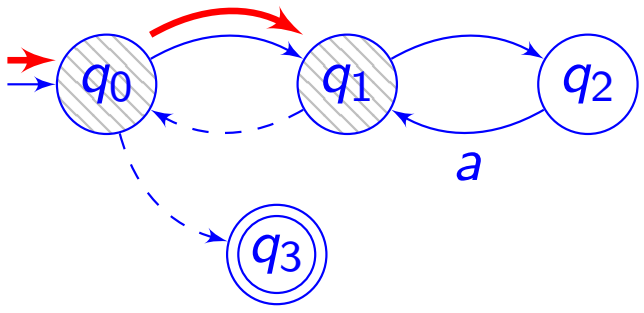


Matching the string aa.

If it were an NFA: infinitely many ways to do it.

Examples: pNFA semantics and Java behavior

The pNFA for Javas view of $(a*)*$ is $A = (\{q_2, q_3\}, \{q_0, q_1\}, \{a\}, q_0, \{(q_2, a, q_1)\}, \{[q_0, (q_1, q_3)], [q_1, (q_2, q_0)]\}, q_3)$.

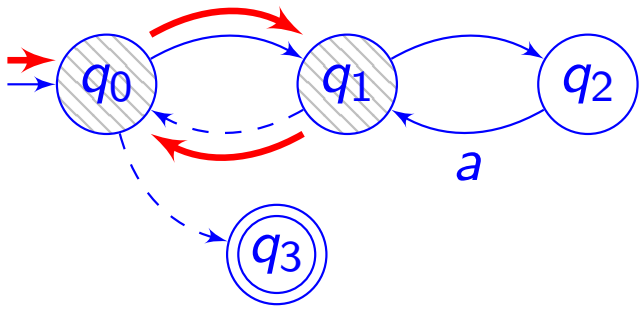


Matching the string aa.

If it were an NFA: infinitely many ways to do it.

Examples: pNFA semantics and Java behavior

The pNFA for Javas view of $(a*)*$ is $A = (\{q_2, q_3\}, \{q_0, q_1\}, \{a\}, q_0, \{(q_2, a, q_1)\}, \{[q_0, (q_1, q_3)], [q_1, (q_2, q_0)]\}, q_3)$.

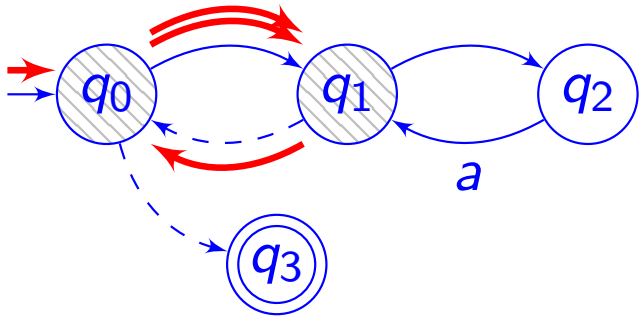


Matching the string aa.

If it were an NFA: infinitely many ways to do it.

Examples: pNFA semantics and Java behavior

The pNFA for Javas view of $(a*)*$ is $A = (\{q_2, q_3\}, \{q_0, q_1\}, \{a\}, q_0, \{(q_2, a, q_1)\}, \{[q_0, (q_1, q_3)], [q_1, (q_2, q_0)]\}, q_3)$.

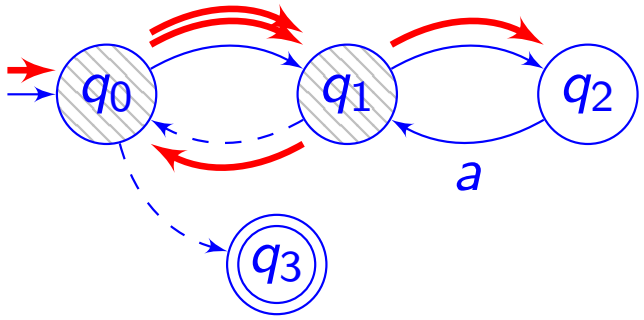


Matching the string aa.

If it were an NFA: infinitely many ways to do it.

Examples: pNFA semantics and Java behavior

The pNFA for Javas view of $(a*)*$ is $A = (\{q_2, q_3\}, \{q_0, q_1\}, \{a\}, q_0, \{(q_2, a, q_1)\}, \{[q_0, (q_1, q_3)], [q_1, (q_2, q_0)]\}, q_3)$.

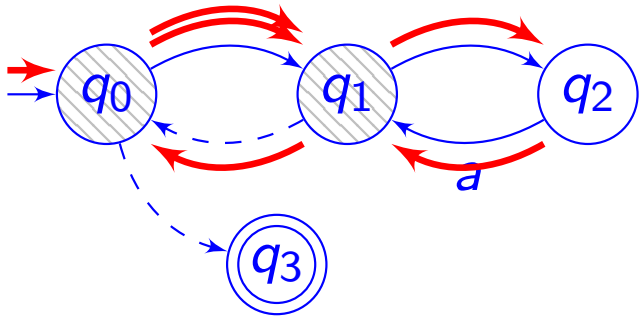


Matching the string aa.

If it were an NFA: infinitely many ways to do it.

Examples: pNFA semantics and Java behavior

The pNFA for Javas view of $(a*)*$ is $A = (\{q_2, q_3\}, \{q_0, q_1\}, \{a\}, q_0, \{(q_2, a, q_1)\}, \{[q_0, (q_1, q_3)], [q_1, (q_2, q_0)]\}, q_3)$.

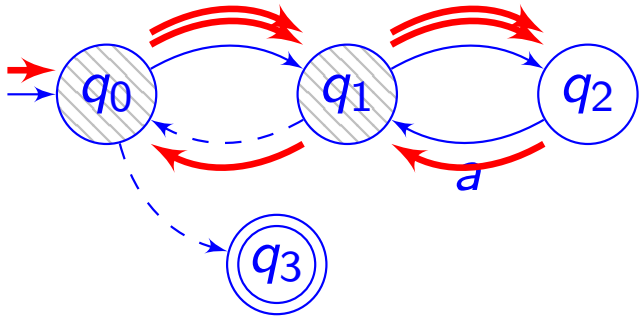


Matching the string aa.

If it were an NFA: infinitely many ways to do it.

Examples: pNFA semantics and Java behavior

The pNFA for Javas view of $(a*)*$ is $A = (\{q_2, q_3\}, \{q_0, q_1\}, \{a\}, q_0, \{(q_2, a, q_1)\}, \{[q_0, (q_1, q_3)], [q_1, (q_2, q_0)]\}, q_3)$.

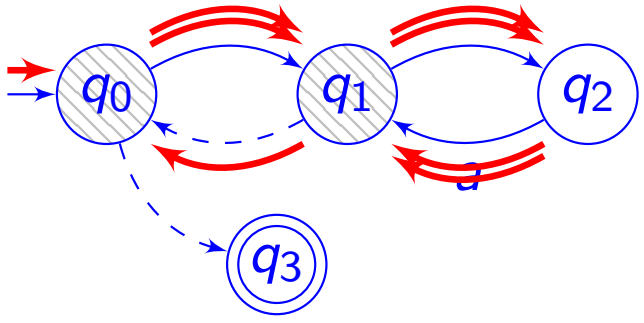


Matching the string aa.

If it were an NFA: infinitely many ways to do it.

Examples: pNFA semantics and Java behavior

The pNFA for Javas view of $(a*)*$ is $A = (\{q_2, q_3\}, \{q_0, q_1\}, \{a\}, q_0, \{(q_2, a, q_1)\}, \{[q_0, (q_1, q_3)], [q_1, (q_2, q_0)]\}, q_3)$.

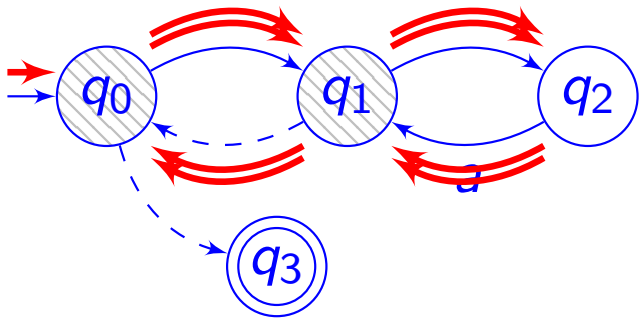


Matching the string aa.

If it were an NFA: infinitely many ways to do it.

Examples: pNFA semantics and Java behavior

The pNFA for Javas view of $(a*)*$ is $A = (\{q_2, q_3\}, \{q_0, q_1\}, \{a\}, q_0, \{(q_2, a, q_1)\}, \{[q_0, (q_1, q_3)], [q_1, (q_2, q_0)]\}, q_3)$.

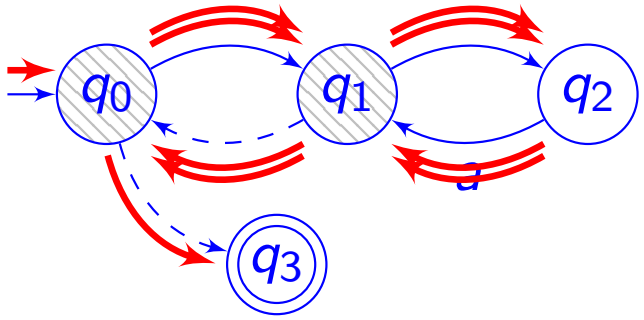


Matching the string aa.

If it were an NFA: infinitely many ways to do it.

Examples: pNFA semantics and Java behavior

The pNFA for Javas view of $(a*)*$ is $A = (\{q_2, q_3\}, \{q_0, q_1\}, \{a\}, q_0, \{(q_2, a, q_1)\}, \{[q_0, (q_1, q_3)], [q_1, (q_2, q_0)]\}, q_3)$.

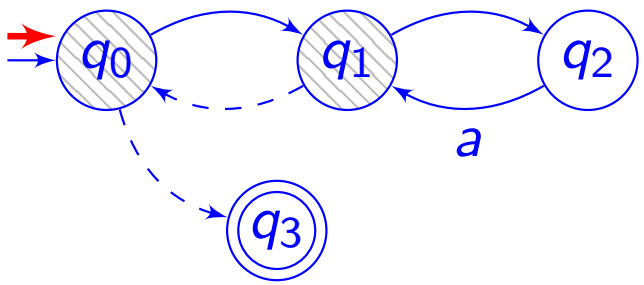


Matching the string aa.

If it were an NFA: infinitely many ways to do it.

Examples: pNFA semantics and Java behavior

The pNFA for Javas view of $(a*)*$ is $A = (\{q_2, q_3\}, \{q_0, q_1\}, \{a\}, q_0, \{(q_2, a, q_1)\}, \{[q_0, (q_1, q_3)], [q_1, (q_2, q_0)]\}, q_3)$.



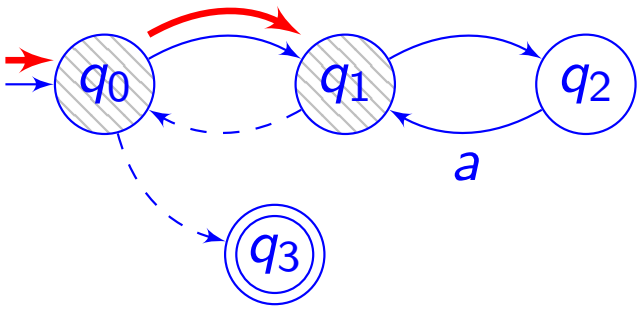
Matching the string aa.

An accepting pNFA path p is the *highest priority* computation path if:

1. It is an accepting path in the usual finite automata sense
2. It cannot use the same δ_2 transition twice in the same all-δ_2 sub-path,
3. No path p' fulfilling 1 and 2 is a prefix of p
4. For any other path p' fulfilling 1–3 the *first* difference between p and p' has p' take a *lower priority state* than p in the *same* δ_2 transition

Examples: pNFA semantics and Java behavior

The pNFA for Javas view of $(a*)*$ is $A = (\{q_2, q_3\}, \{q_0, q_1\}, \{a\}, q_0, \{(q_2, a, q_1)\}, \{[q_0, (q_1, q_3)], [q_1, (q_2, q_0)]\}, q_3)$.



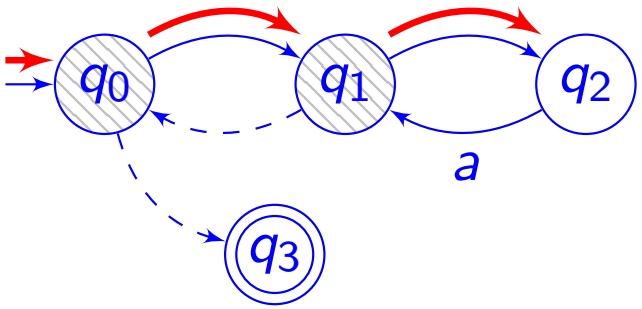
Matching the string aa.

An accepting pNFA path p is the *highest priority* computation path if:

1. It is an accepting path in the usual finite automata sense
2. It cannot use the same δ_2 transition twice in the same all-δ_2 sub-path,
3. No path p' fulfilling 1 and 2 is a prefix of p
4. For any other path p' fulfilling 1–3 the *first* difference between p and p' has p' take a *lower priority state* than p in the *same* δ_2 transition

Examples: pNFA semantics and Java behavior

The pNFA for Javas view of $(a*)*$ is $A = (\{q_2, q_3\}, \{q_0, q_1\}, \{a\}, q_0, \{(q_2, a, q_1)\}, \{[q_0, (q_1, q_3)], [q_1, (q_2, q_0)]\}, q_3)$.



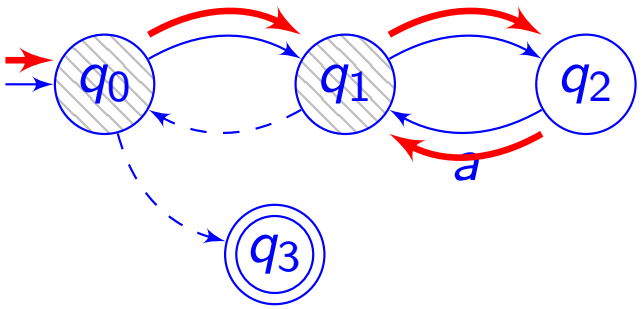
Matching the string aa.

An accepting pNFA path p is the *highest priority* computation path if:

1. It is an accepting path in the usual finite automata sense
2. It cannot use the same δ_2 transition twice in the same all-δ_2 sub-path,
3. No path p' fulfilling 1 and 2 is a prefix of p
4. For any other path p' fulfilling 1–3 the *first* difference between p and p' has p' take a *lower priority state* than p in the *same* δ_2 transition

Examples: pNFA semantics and Java behavior

The pNFA for Javas view of $(a*)*$ is $A = (\{q_2, q_3\}, \{q_0, q_1\}, \{a\}, q_0, \{(q_2, a, q_1)\}, \{[q_0, (q_1, q_3)], [q_1, (q_2, q_0)]\}, q_3)$.



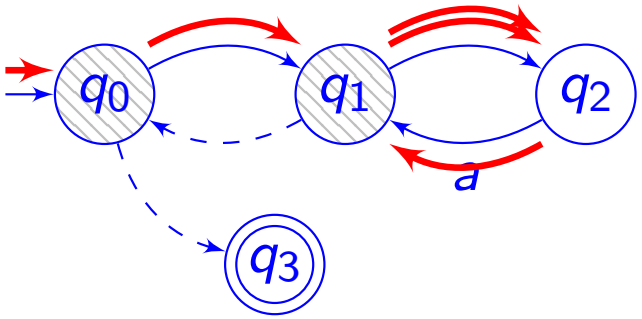
Matching the string aa.

An accepting pNFA path p is the *highest priority* computation path if:

1. It is an accepting path in the usual finite automata sense
2. It cannot use the same δ_2 transition twice in the same all-δ_2 sub-path,
3. No path p' fulfilling 1 and 2 is a prefix of p
4. For any other path p' fulfilling 1–3 the *first* difference between p and p' has p' take a *lower priority state* than p in the *same* δ_2 transition

Examples: pNFA semantics and Java behavior

The pNFA for Javas view of $(a*)*$ is $A = (\{q_2, q_3\}, \{q_0, q_1\}, \{a\}, q_0, \{(q_2, a, q_1)\}, \{[q_0, (q_1, q_3)], [q_1, (q_2, q_0)]\}, q_3)$.



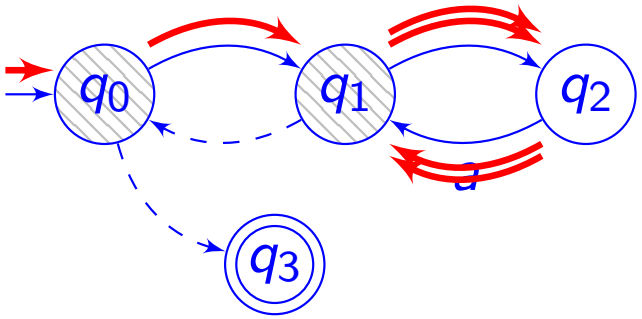
Matching the string aa.

An accepting pNFA path p is the *highest priority* computation path if:

1. It is an accepting path in the usual finite automata sense
2. It cannot use the same δ_2 transition twice in the same all-δ_2 sub-path,
3. No path p' fulfilling 1 and 2 is a prefix of p
4. For any other path p' fulfilling 1–3 the *first* difference between p and p' has p' take a *lower priority state* than p in the *same* δ_2 transition

Examples: pNFA semantics and Java behavior

The pNFA for Javas view of $(a*)*$ is $A = (\{q_2, q_3\}, \{q_0, q_1\}, \{a\}, q_0, \{(q_2, a, q_1)\}, \{[q_0, (q_1, q_3)], [q_1, (q_2, q_0)]\}, q_3)$.



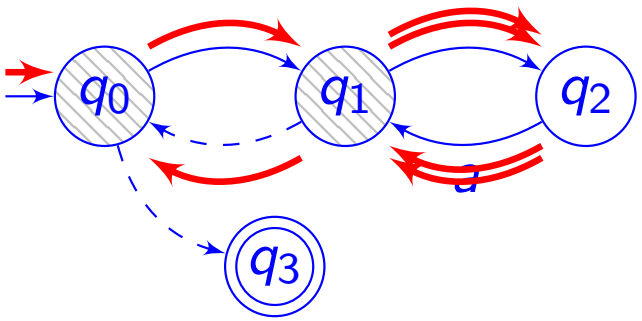
Matching the string aa.

An accepting pNFA path p is the *highest priority* computation path if:

1. It is an accepting path in the usual finite automata sense
2. It cannot use the same δ_2 transition twice in the same all-δ_2 sub-path,
3. No path p' fulfilling 1 and 2 is a prefix of p
4. For any other path p' fulfilling 1–3 the *first* difference between p and p' has p' take a *lower priority state* than p in the *same* δ_2 transition

Examples: pNFA semantics and Java behavior

The pNFA for Javas view of $(a*)*$ is $A = (\{q_2, q_3\}, \{q_0, q_1\}, \{a\}, q_0, \{(q_2, a, q_1)\}, \{[q_0, (q_1, q_3)], [q_1, (q_2, q_0)]\}, q_3)$.



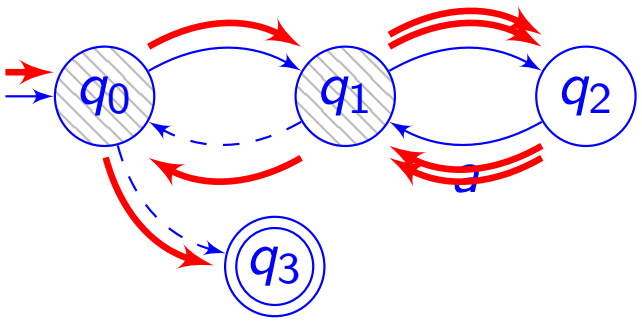
Matching the string aa.

An accepting pNFA path p is the *highest priority* computation path if:

1. It is an accepting path in the usual finite automata sense
2. It cannot use the same δ_2 transition twice in the same all-δ_2 sub-path,
3. No path p' fulfilling 1 and 2 is a prefix of p
4. For any other path p' fulfilling 1–3 the *first* difference between p and p' has p' take a *lower priority state* than p in the *same* δ_2 transition

Examples: pNFA semantics and Java behavior

The pNFA for Javas view of $(a*)*$ is $A = (\{q_2, q_3\}, \{q_0, q_1\}, \{a\}, q_0, \{(q_2, a, q_1)\}, \{[q_0, (q_1, q_3)], [q_1, (q_2, q_0)]\}, q_3)$.



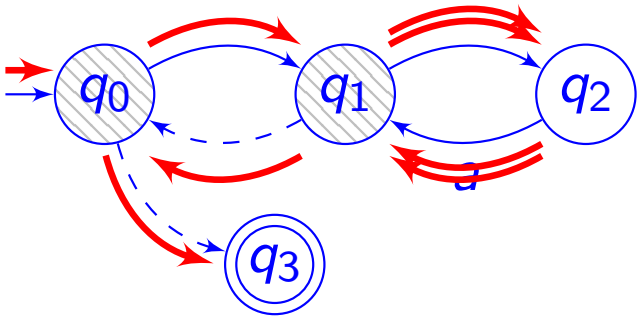
Matching the string aa.

An accepting pNFA path p is the *highest priority* computation path if:

1. It is an accepting path in the usual finite automata sense
2. It cannot use the same δ_2 transition twice in the same all-δ_2 sub-path,
3. No path p' fulfilling 1 and 2 is a prefix of p
4. For any other path p' fulfilling 1–3 the *first* difference between p and p' has p' take a *lower priority state* than p in the *same* δ_2 transition

Examples: pNFA semantics and Java behavior

The pNFA for Javas view of $(a*)*$ is $A = (\{q_2, q_3\}, \{q_0, q_1\}, \{a\}, q_0, \{(q_2, a, q_1)\}, \{[q_0, (q_1, q_3)], [q_1, (q_2, q_0)]\}, q_3)$.



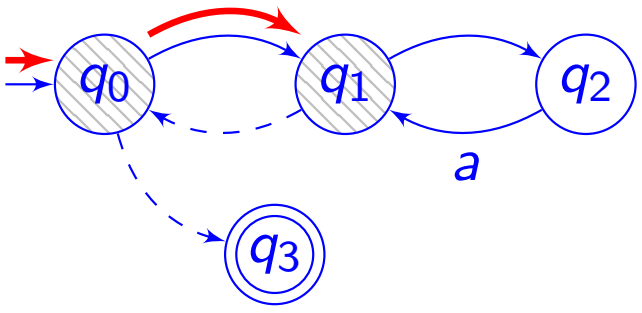
Matching the string aa.

Java effectively finds the pNFA path:

1. Actual representation is the quirky object graph: we give a Thomson-style construction which creates pNFA (pTr) mimicking Java
2. Java follows the most common approach: greedy depth-first search until a path is found
3. The non-looping is handled in each looping construct object in Java, rejecting a path rather than looping on an empty string

Examples: pNFA semantics and Java behavior

The pNFA for Javas view of $(a*)*$ is $A = (\{q_2, q_3\}, \{q_0, q_1\}, \{a\}, q_0, \{(q_2, a, q_1)\}, \{[q_0, (q_1, q_3)], [q_1, (q_2, q_0)]\}, q_3)$.



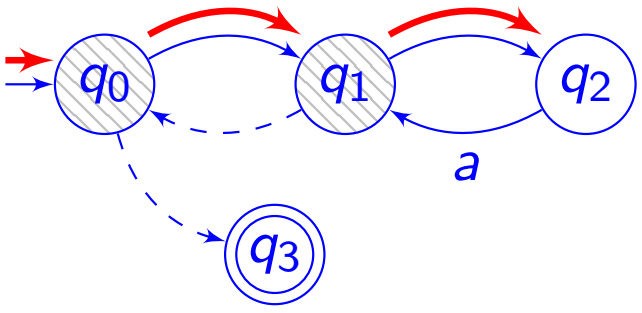
Matching the string aab.

The pure NFA/pNFA perspective: simple, no accepting paths.

Java attempts *exponentially many* candidate paths before rejecting however.

Examples: pNFA semantics and Java behavior

The pNFA for Javas view of $(a*)*$ is $A = (\{q_2, q_3\}, \{q_0, q_1\}, \{a\}, q_0, \{(q_2, a, q_1)\}, \{[q_0, (q_1, q_3)], [q_1, (q_2, q_0)]\}, q_3)$.



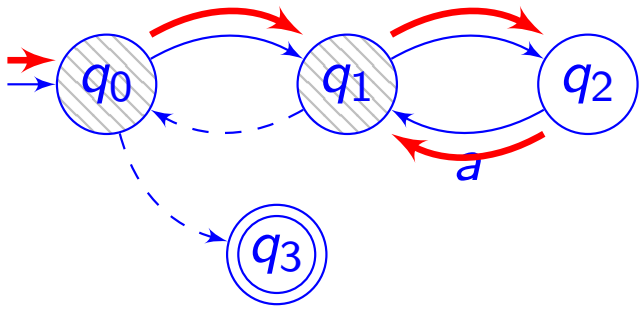
Matching the string aab.

The pure NFA/pNFA perspective: simple, no accepting paths.

Java attempts *exponentially many* candidate paths before rejecting however.

Examples: pNFA semantics and Java behavior

The pNFA for Javas view of $(a*)*$ is $A = (\{q_2, q_3\}, \{q_0, q_1\}, \{a\}, q_0, \{(q_2, a, q_1)\}, \{[q_0, (q_1, q_3)], [q_1, (q_2, q_0)]\}, q_3)$.



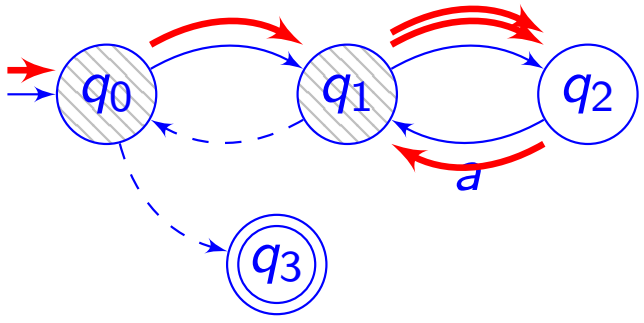
Matching the string aab.

The pure NFA/pNFA perspective: simple, no accepting paths.

Java attempts *exponentially many* candidate paths before rejecting however.

Examples: pNFA semantics and Java behavior

The pNFA for Javas view of $(a*)*$ is $A = (\{q_2, q_3\}, \{q_0, q_1\}, \{a\}, q_0, \{(q_2, a, q_1)\}, \{[q_0, (q_1, q_3)], [q_1, (q_2, q_0)]\}, q_3)$.



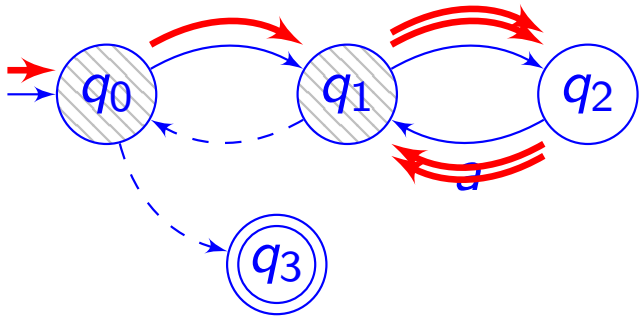
Matching the string aab.

The pure NFA/pNFA perspective: simple, no accepting paths.

Java attempts *exponentially many* candidate paths before rejecting however.

Examples: pNFA semantics and Java behavior

The pNFA for Javas view of $(a*)*$ is $A = (\{q_2, q_3\}, \{q_0, q_1\}, \{a\}, q_0, \{(q_2, a, q_1)\}, \{[q_0, (q_1, q_3)], [q_1, (q_2, q_0)]\}, q_3)$.



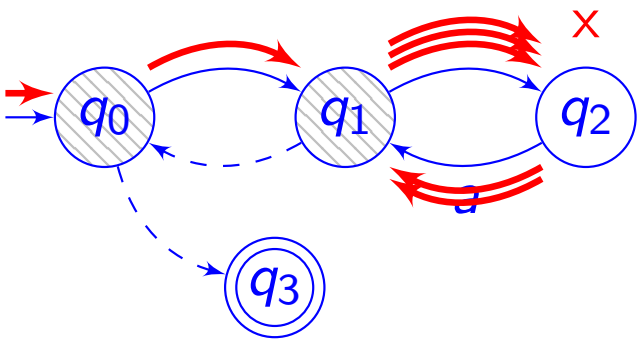
Matching the string aab.

The pure NFA/pNFA perspective: simple, no accepting paths.

Java attempts *exponentially many* candidate paths before rejecting however.

Examples: pNFA semantics and Java behavior

The pNFA for Javas view of $(a*)*$ is $A = (\{q_2, q_3\}, \{q_0, q_1\}, \{a\}, q_0, \{(q_2, a, q_1)\}, \{[q_0, (q_1, q_3)], [q_1, (q_2, q_0)]\}, q_3)$.



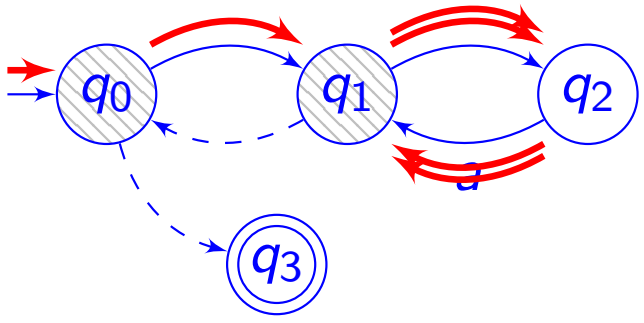
Matching the string aab.

The pure NFA/pNFA perspective: simple, no accepting paths.

Java attempts *exponentially many* candidate paths before rejecting however.

Examples: pNFA semantics and Java behavior

The pNFA for Javas view of $(a*)*$ is $A = (\{q_2, q_3\}, \{q_0, q_1\}, \{a\}, q_0, \{(q_2, a, q_1)\}, \{[q_0, (q_1, q_3)], [q_1, (q_2, q_0)]\}, q_3)$.



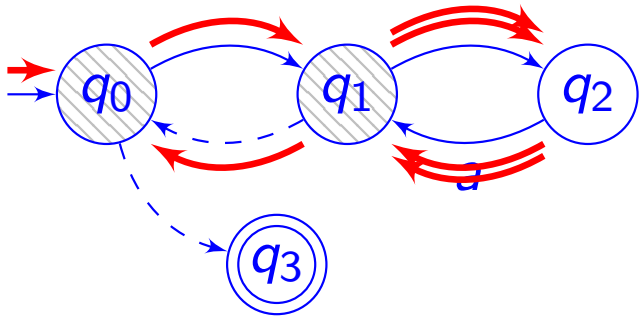
Matching the string aab.

The pure NFA/pNFA perspective: simple, no accepting paths.

Java attempts *exponentially many* candidate paths before rejecting however.

Examples: pNFA semantics and Java behavior

The pNFA for Javas view of $(a*)*$ is $A = (\{q_2, q_3\}, \{q_0, q_1\}, \{a\}, q_0, \{(q_2, a, q_1)\}, \{[q_0, (q_1, q_3)], [q_1, (q_2, q_0)]\}, q_3)$.



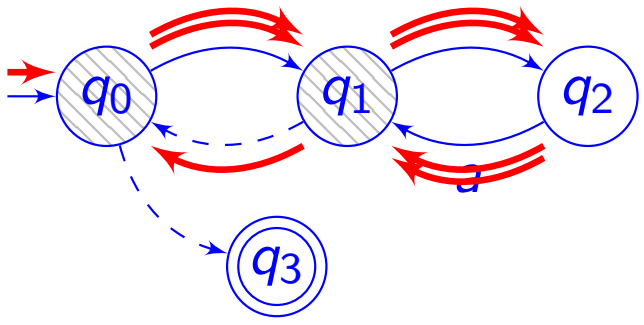
Matching the string aab.

The pure NFA/pNFA perspective: simple, no accepting paths.

Java attempts *exponentially many* candidate paths before rejecting however.

Examples: pNFA semantics and Java behavior

The pNFA for Javas view of $(a*)*$ is $A = (\{q_2, q_3\}, \{q_0, q_1\}, \{a\}, q_0, \{(q_2, a, q_1)\}, \{[q_0, (q_1, q_3)], [q_1, (q_2, q_0)]\}, q_3)$.



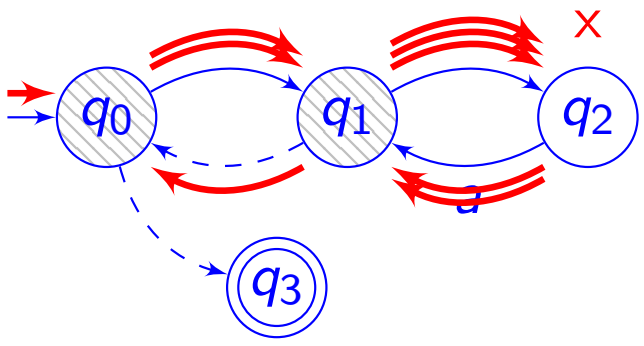
Matching the string aab.

The pure NFA/pNFA perspective: simple, no accepting paths.

Java attempts *exponentially many* candidate paths before rejecting however.

Examples: pNFA semantics and Java behavior

The pNFA for Javas view of $(a*)*$ is $A = (\{q_2, q_3\}, \{q_0, q_1\}, \{a\}, q_0, \{(q_2, a, q_1)\}, \{[q_0, (q_1, q_3)], [q_1, (q_2, q_0)]\}, q_3)$.



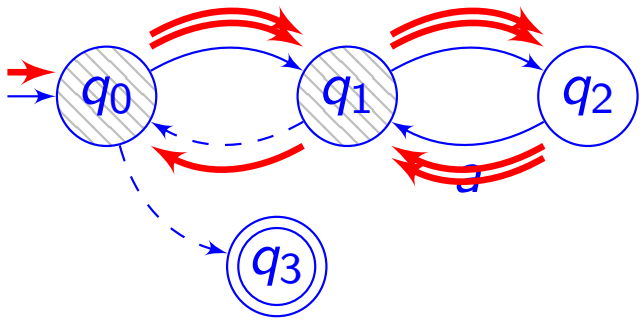
Matching the string aab.

The pure NFA/pNFA perspective: simple, no accepting paths.

Java attempts *exponentially many* candidate paths before rejecting however.

Examples: pNFA semantics and Java behavior

The pNFA for Javas view of $(a*)*$ is $A = (\{q_2, q_3\}, \{q_0, q_1\}, \{a\}, q_0, \{(q_2, a, q_1)\}, \{[q_0, (q_1, q_3)], [q_1, (q_2, q_0)]\}, q_3)$.



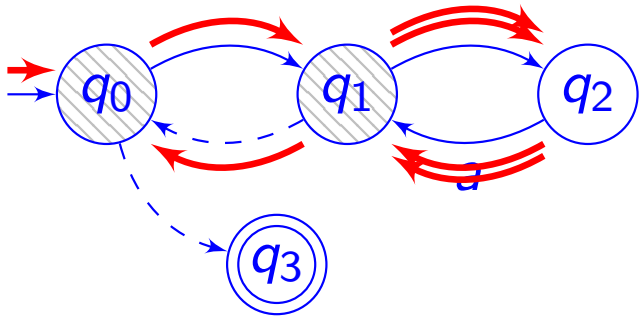
Matching the string aab.

The pure NFA/pNFA perspective: simple, no accepting paths.

Java attempts *exponentially many* candidate paths before rejecting however.

Examples: pNFA semantics and Java behavior

The pNFA for Javas view of $(a*)*$ is $A = (\{q_2, q_3\}, \{q_0, q_1\}, \{a\}, q_0, \{(q_2, a, q_1)\}, \{[q_0, (q_1, q_3)], [q_1, (q_2, q_0)]\}, q_3)$.



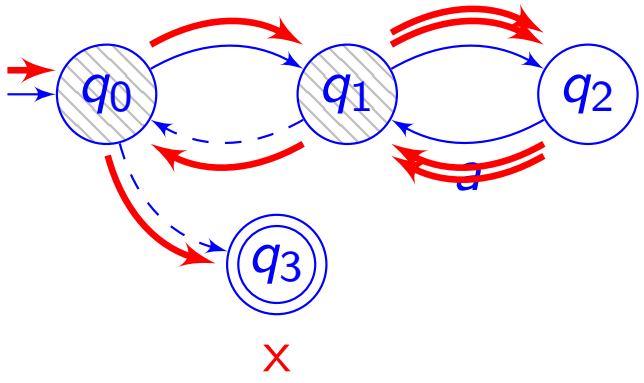
Matching the string aab.

The pure NFA/pNFA perspective: simple, no accepting paths.

Java attempts *exponentially many* candidate paths before rejecting however.

Examples: pNFA semantics and Java behavior

The pNFA for Javas view of $(a*)*$ is $A = (\{q_2, q_3\}, \{q_0, q_1\}, \{a\}, q_0, \{(q_2, a, q_1)\}, \{[q_0, (q_1, q_3)], [q_1, (q_2, q_0)]\}, q_3)$.



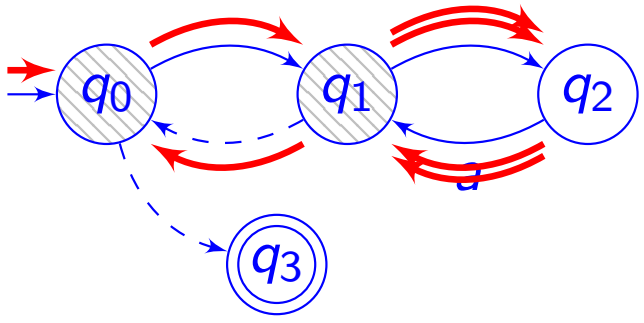
Matching the string aab.

The pure NFA/pNFA perspective: simple, no accepting paths.

Java attempts *exponentially many* candidate paths before rejecting however.

Examples: pNFA semantics and Java behavior

The pNFA for Javas view of $(a*)*$ is $A = (\{q_2, q_3\}, \{q_0, q_1\}, \{a\}, q_0, \{(q_2, a, q_1)\}, \{[q_0, (q_1, q_3)], [q_1, (q_2, q_0)]\}, q_3)$.



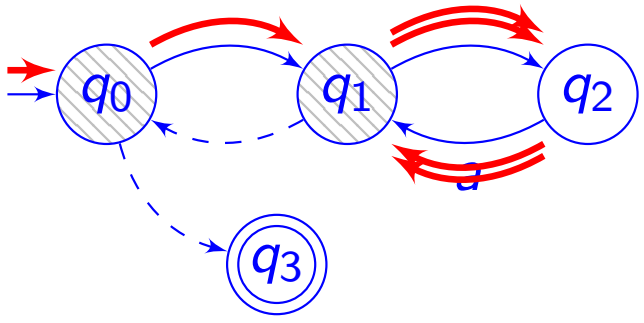
Matching the string aab.

The pure NFA/pNFA perspective: simple, no accepting paths.

Java attempts *exponentially many* candidate paths before rejecting however.

Examples: pNFA semantics and Java behavior

The pNFA for Javas view of $(a*)*$ is $A = (\{q_2, q_3\}, \{q_0, q_1\}, \{a\}, q_0, \{(q_2, a, q_1)\}, \{[q_0, (q_1, q_3)], [q_1, (q_2, q_0)]\}, q_3)$.



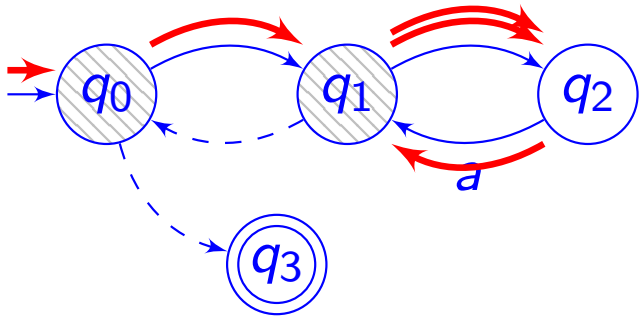
Matching the string aab.

The pure NFA/pNFA perspective: simple, no accepting paths.

Java attempts *exponentially many* candidate paths before rejecting however.

Examples: pNFA semantics and Java behavior

The pNFA for Javas view of $(a*)*$ is $A = (\{q_2, q_3\}, \{q_0, q_1\}, \{a\}, q_0, \{(q_2, a, q_1)\}, \{[q_0, (q_1, q_3)], [q_1, (q_2, q_0)]\}, q_3)$.



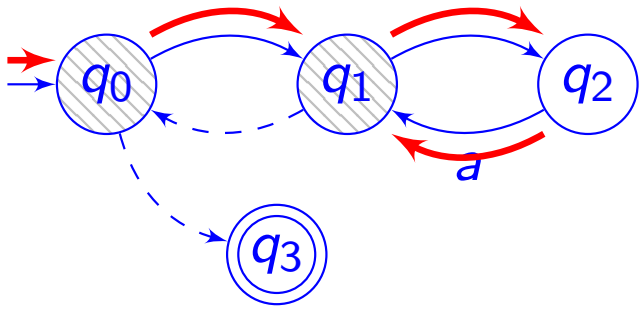
Matching the string aab.

The pure NFA/pNFA perspective: simple, no accepting paths.

Java attempts *exponentially many* candidate paths before rejecting however.

Examples: pNFA semantics and Java behavior

The pNFA for Javas view of $(a*)*$ is $A = (\{q_2, q_3\}, \{q_0, q_1\}, \{a\}, q_0, \{(q_2, a, q_1)\}, \{[q_0, (q_1, q_3)], [q_1, (q_2, q_0)]\}, q_3)$.



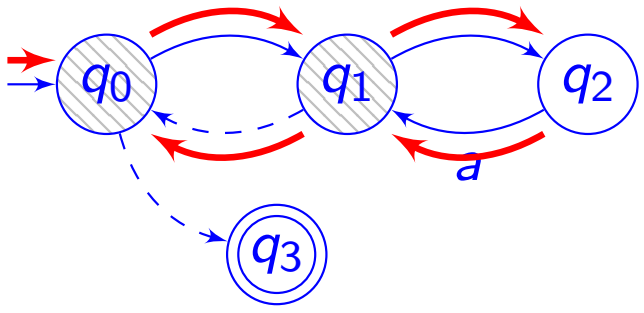
Matching the string aab.

The pure NFA/pNFA perspective: simple, no accepting paths.

Java attempts *exponentially many* candidate paths before rejecting however.

Examples: pNFA semantics and Java behavior

The pNFA for Javas view of $(a*)*$ is $A = (\{q_2, q_3\}, \{q_0, q_1\}, \{a\}, q_0, \{(q_2, a, q_1)\}, \{[q_0, (q_1, q_3)], [q_1, (q_2, q_0)]\}, q_3)$.



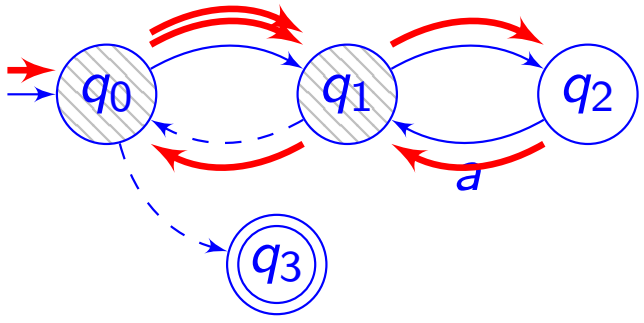
Matching the string aab.

The pure NFA/pNFA perspective: simple, no accepting paths.

Java attempts *exponentially many* candidate paths before rejecting however.

Examples: pNFA semantics and Java behavior

The pNFA for Javas view of $(a*)*$ is $A = (\{q_2, q_3\}, \{q_0, q_1\}, \{a\}, q_0, \{(q_2, a, q_1)\}, \{[q_0, (q_1, q_3)], [q_1, (q_2, q_0)]\}, q_3)$.



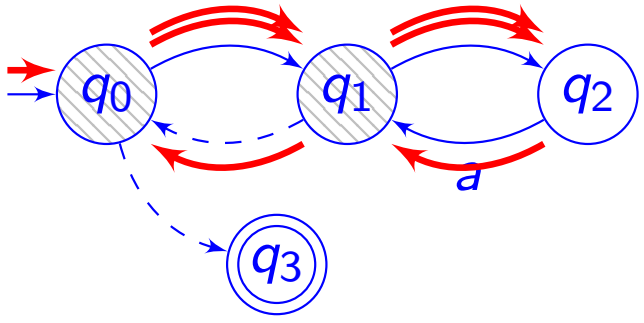
Matching the string aab.

The pure NFA/pNFA perspective: simple, no accepting paths.

Java attempts *exponentially many* candidate paths before rejecting however.

Examples: pNFA semantics and Java behavior

The pNFA for Javas view of $(a*)*$ is $A = (\{q_2, q_3\}, \{q_0, q_1\}, \{a\}, q_0, \{(q_2, a, q_1)\}, \{[q_0, (q_1, q_3)], [q_1, (q_2, q_0)]\}, q_3)$.



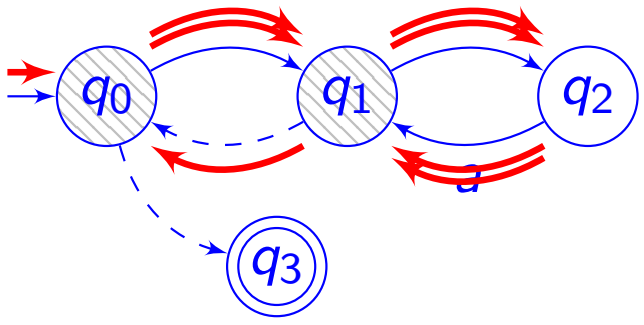
Matching the string aab.

The pure NFA/pNFA perspective: simple, no accepting paths.

Java attempts *exponentially many* candidate paths before rejecting however.

Examples: pNFA semantics and Java behavior

The pNFA for Javas view of $(a*)*$ is $A = (\{q_2, q_3\}, \{q_0, q_1\}, \{a\}, q_0, \{(q_2, a, q_1)\}, \{[q_0, (q_1, q_3)], [q_1, (q_2, q_0)]\}, q_3)$.



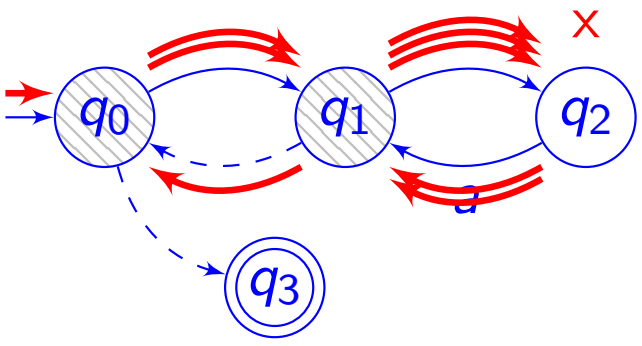
Matching the string aab.

The pure NFA/pNFA perspective: simple, no accepting paths.

Java attempts *exponentially many* candidate paths before rejecting however.

Examples: pNFA semantics and Java behavior

The pNFA for Javas view of $(a*)*$ is $A = (\{q_2, q_3\}, \{q_0, q_1\}, \{a\}, q_0, \{(q_2, a, q_1)\}, \{[q_0, (q_1, q_3)], [q_1, (q_2, q_0)]\}, q_3)$.



Matching the string aab.

The pure NFA/pNFA perspective: simple, no accepting paths.

Java attempts *exponentially many* candidate paths before rejecting however.

Already illustrates that pNFA help us describe the *semantics*, and *algorithmic behavior* of implementations.

It misses some of the *purpose* of the implementation however

The Capturing Groups Parse: pTr

pTr follow in the mold of pNFA:

$$T = (Q_1, Q_2, \Sigma_1, \Sigma_2, q_0, \delta_1, \delta_2, F)$$

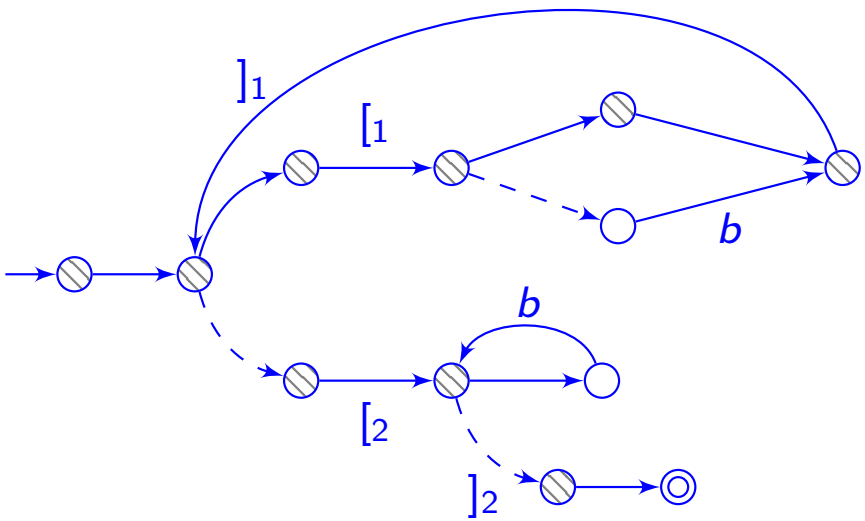
- Q_1 and Q_2 are *finite sets of states*, let $Q := Q_1 \cup Q_2$,
- Σ_1 is the *input alphabet*,
- Σ_2 is the *output alphabet*,
- $q_0 \in Q$ is the *initial state*,
- $\delta_1 : Q_1 \times \Sigma \to Q$ is the *deterministic transition* function,
- $\delta_2 : Q_2 \to (\Sigma_2^* \times Q)^*$ is the *prioritized non-det. transition* function,
- $F \subseteq Q_1$ are the final states.

Computation paths analogous to pNFA, just adds output. Each transition adds its label to the output, *both* δ_1 *and* δ_2

The output is a *decoration* of the input. With $v = T(w)$ we have $v \in (\Sigma_1 \cup \Sigma_2)^*$ and removing all Σ_2 symbols from v produces w

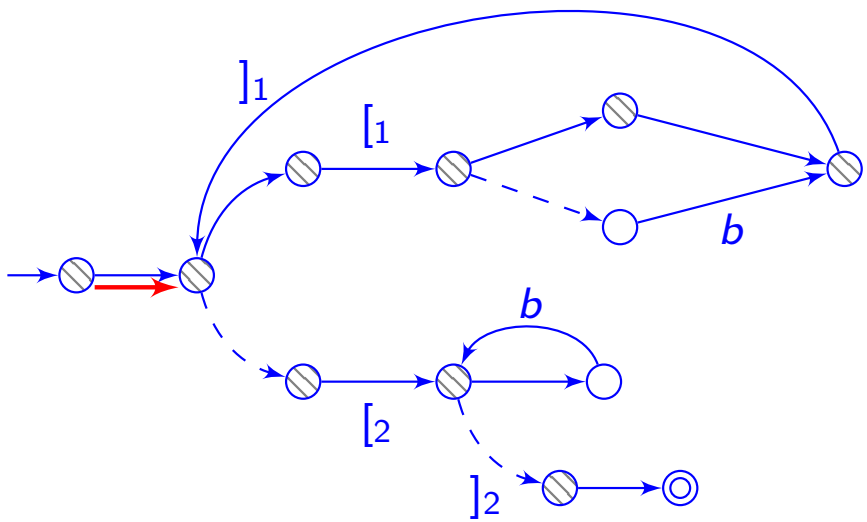
pTr Computation Example

For example, $T(bb) = [_1]_1[_2bb]_2$ for T as below, where we have $\Sigma_1 = \{b\}$ and $\Sigma_2 = \{[_1,]_1, [_2,]_2\}$:



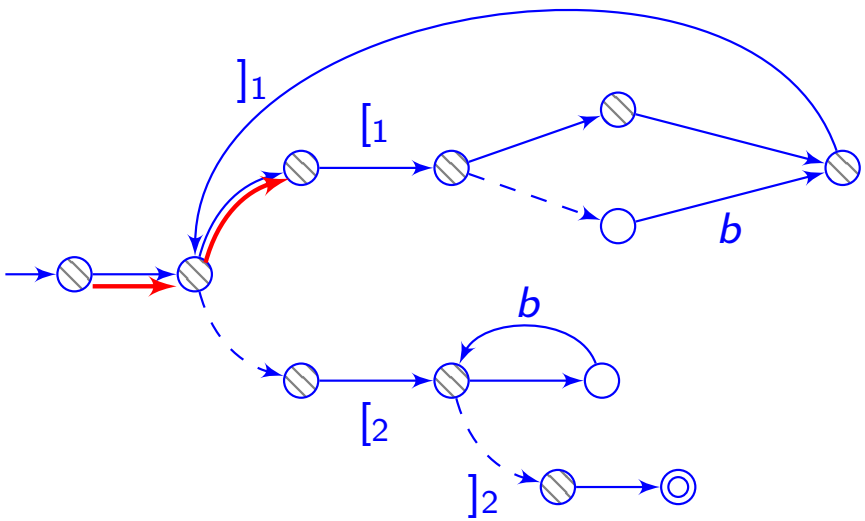
pTr Computation Example

For example, $T(bb) = [_1]_1[_2bb]_2$ for T as below, where we have $\Sigma_1 = \{b\}$ and $\Sigma_2 = \{[_1,]_1, [_2,]_2\}$:



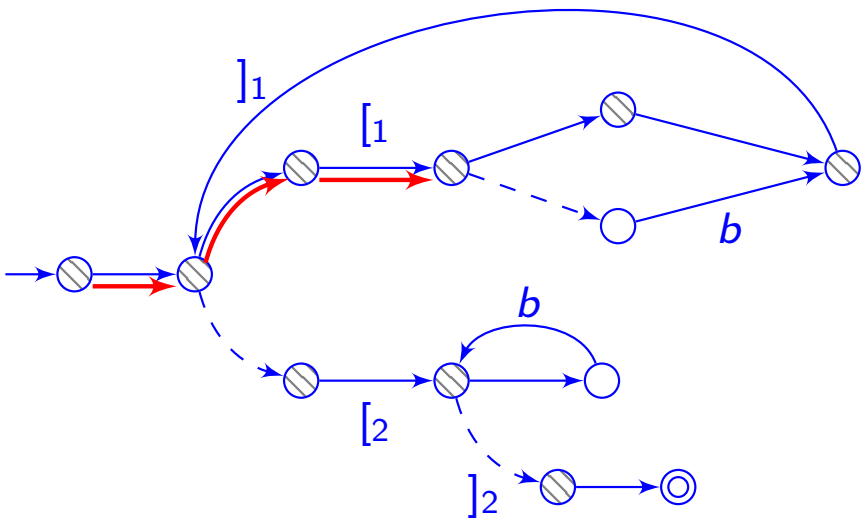
pTr Computation Example

For example, $T(bb) = [_1]_1[_2bb]_2$ for T as below, where we have $\Sigma_1 = \{b\}$ and $\Sigma_2 = \{[_1,]_1, [_2,]_2\}$:



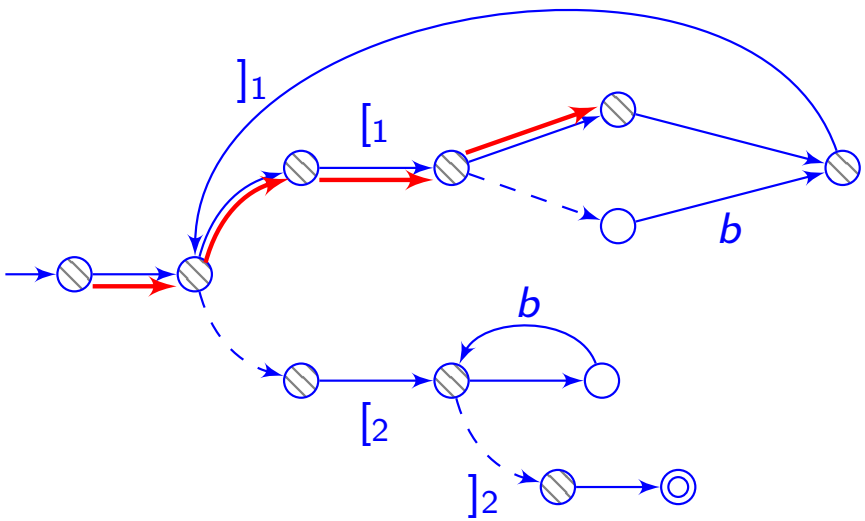
pTr Computation Example

For example, $T(bb) = [_1]_1[_2bb]_2$ for T as below, where we have $\Sigma_1 = \{b\}$ and $\Sigma_2 = \{[_1,]_1, [_2,]_2\}$:



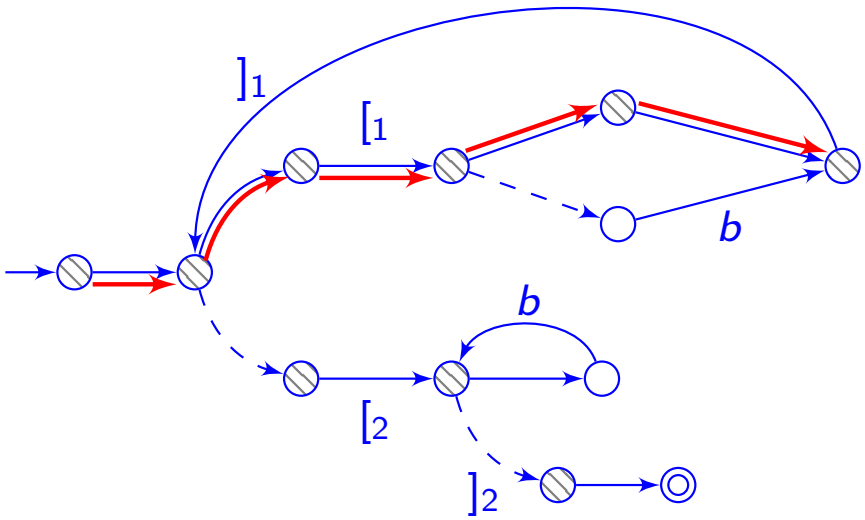
pTr Computation Example

For example, $T(bb) = [_1]_1[_2bb]_2$ for T as below, where we have $\Sigma_1 = \{b\}$ and $\Sigma_2 = \{[_1,]_1, [_2,]_2\}$:



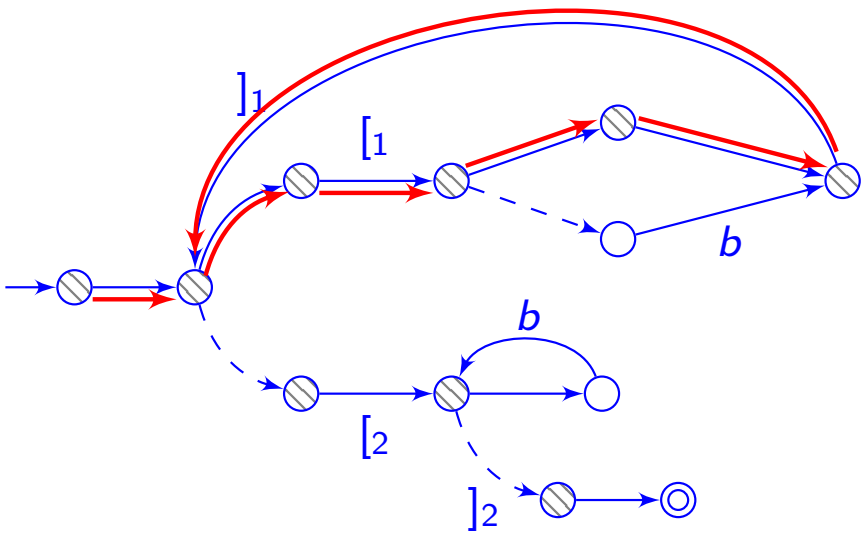
pTr Computation Example

For example, $T(bb) = [_1]_1[_2bb]_2$ for T as below, where we have $\Sigma_1 = \{b\}$ and $\Sigma_2 = \{[_1,]_1, [_2,]_2\}$:



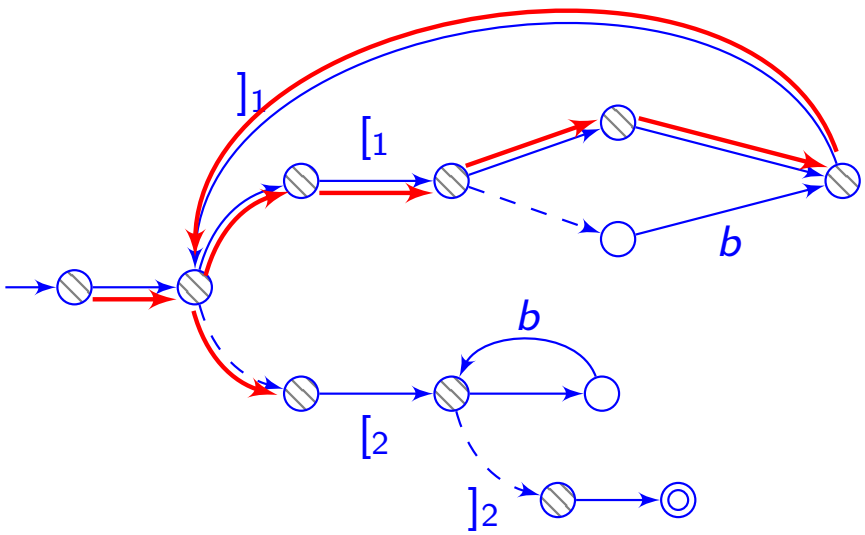
pTr Computation Example

For example, $T(bb) = [_1]_1[_2bb]_2$ for T as below, where we have $\Sigma_1 = \{b\}$ and $\Sigma_2 = \{[_1,]_1, [_2,]_2\}$:



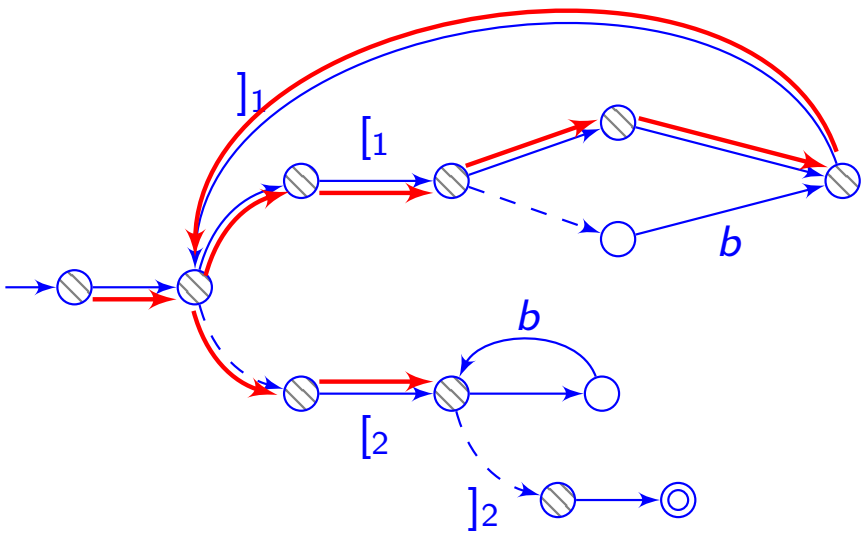
pTr Computation Example

For example, $T(bb) = [_1]_1[_2bb]_2$ for T as below, where we have $\Sigma_1 = \{b\}$ and $\Sigma_2 = \{[_1,]_1, [_2,]_2\}$:



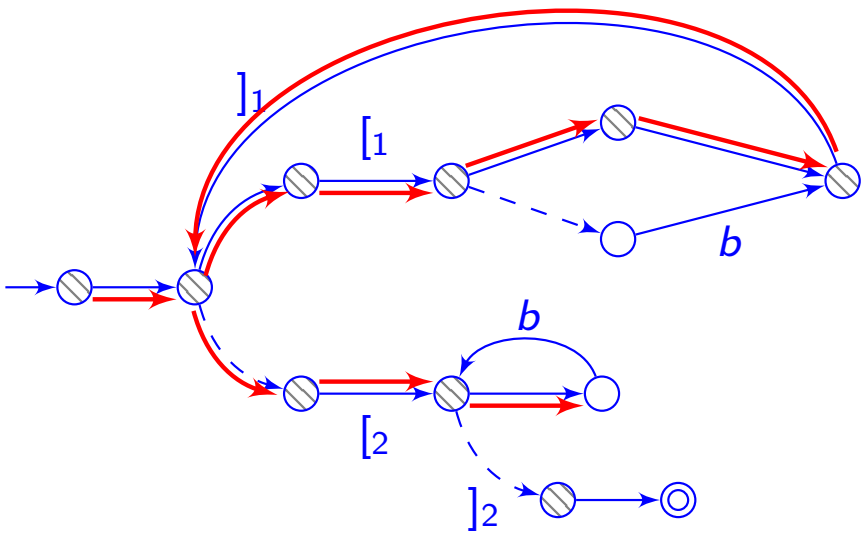
pTr Computation Example

For example, $T(bb) = [_1]_1[_2bb]_2$ for T as below, where we have $\Sigma_1 = \{b\}$ and $\Sigma_2 = \{[_1,]_1, [_2,]_2\}$:



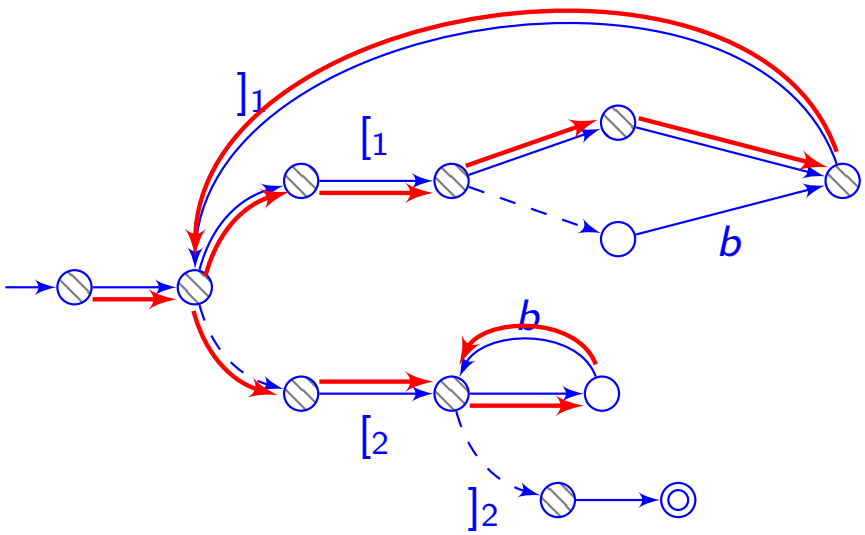
pTr Computation Example

For example, $T(bb) = [_1]_1[_2bb]_2$ for T as below, where we have $\Sigma_1 = \{b\}$ and $\Sigma_2 = \{[_1,]_1, [_2,]_2\}$:



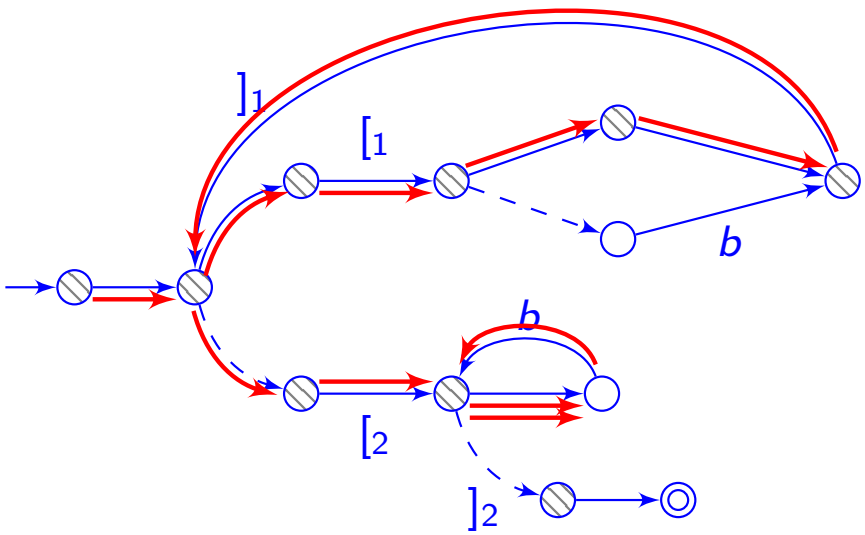
pTr Computation Example

For example, $T(bb) = [_1]_1[_2bb]_2$ for T as below, where we have $\Sigma_1 = \{b\}$ and $\Sigma_2 = \{[_1,]_1, [_2,]_2\}$:



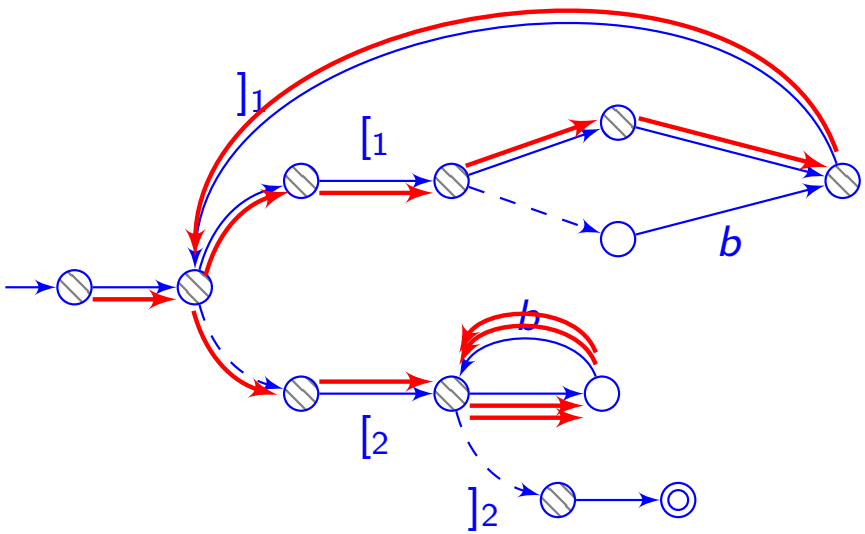
pTr Computation Example

For example, $T(bb) = [_1]_1[_2bb]_2$ for T as below, where we have $\Sigma_1 = \{b\}$ and $\Sigma_2 = \{[_1,]_1, [_2,]_2\}$:



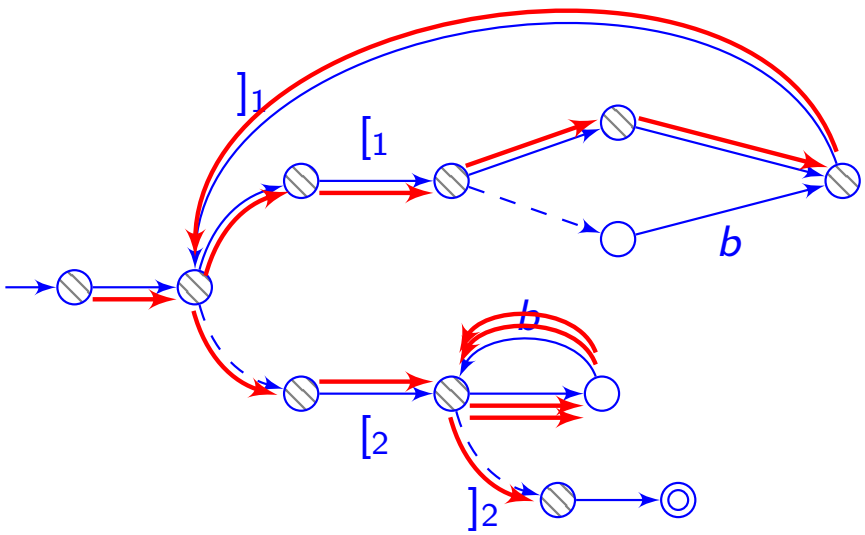
pTr Computation Example

For example, $T(bb) = [_1]_1[_2bb]_2$ for T as below, where we have $\Sigma_1 = \{b\}$ and $\Sigma_2 = \{[_1,]_1, [_2,]_2\}$:



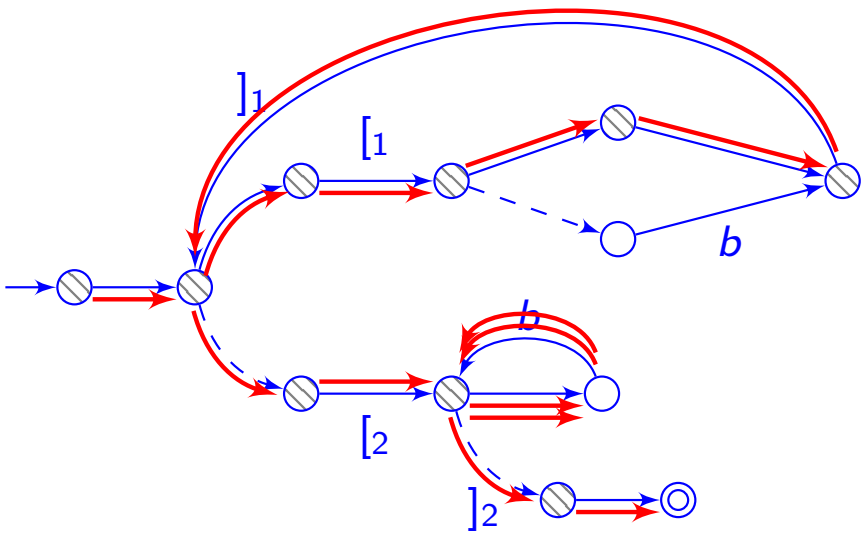
pTr Computation Example

For example, $T(bb) = [_1]_1[_2bb]_2$ for T as below, where we have $\Sigma_1 = \{b\}$ and $\Sigma_2 = \{[_1,]_1, [_2,]_2\}$:



pTr Computation Example

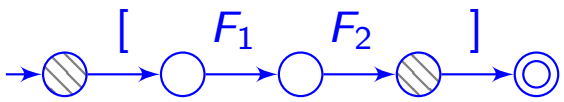
For example, $T(bb) = [_1]_1[_2bb]_2$ for T as below, where we have $\Sigma_1 = \{b\}$ and $\Sigma_2 = \{[_1,]_1, [_2,]_2\}$:



Java pTr Constructions

Some of the building blocks for Java regular expressions as pTr:

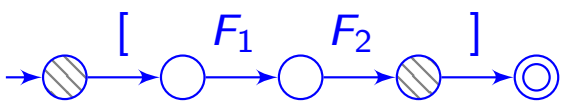
$(F_1 \cdot F_2)$



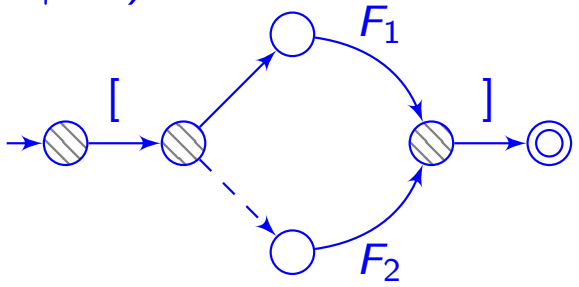
Java pTr Constructions

Some of the building blocks for Java regular expressions as pTr:

$(F_1 \cdot F_2)$



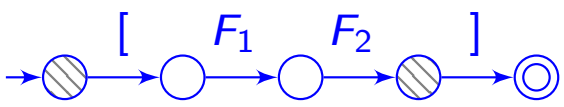
$(F_1 \mid F_2)$



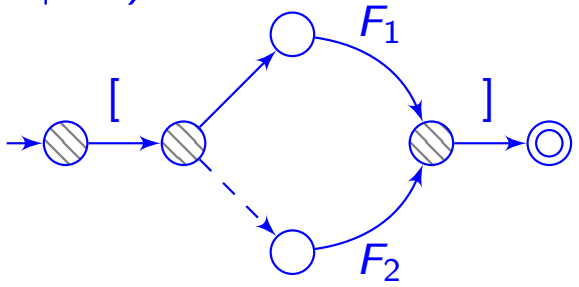
Java pTr Constructions

Some of the building blocks for Java regular expressions as pTr:

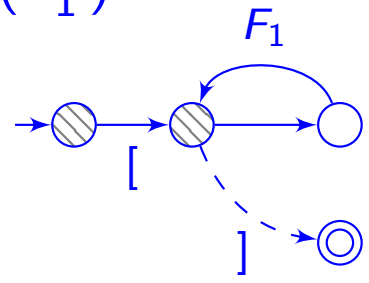
$(F_1 \cdot F_2)$



$(F_1 \mid F_2)$



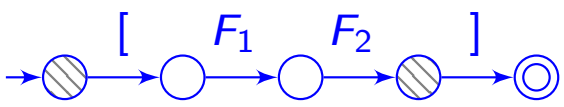
(F_1^*)



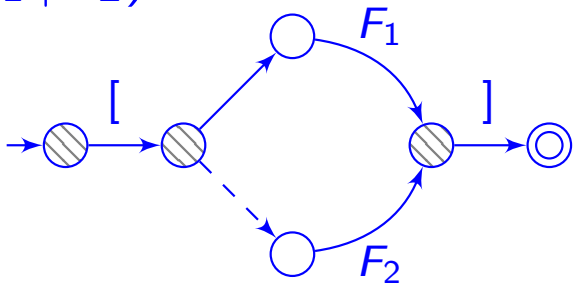
Java pTr Constructions

Some of the building blocks for Java regular expressions as pTr:

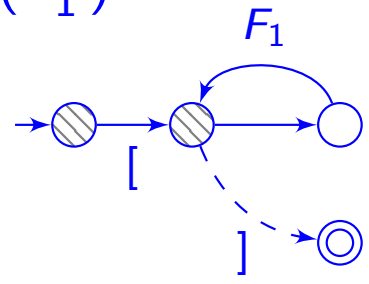
$(F_1 \cdot F_2)$



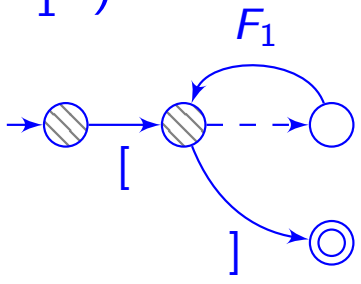
$(F_1 \mid F_2)$



(F_1^*)

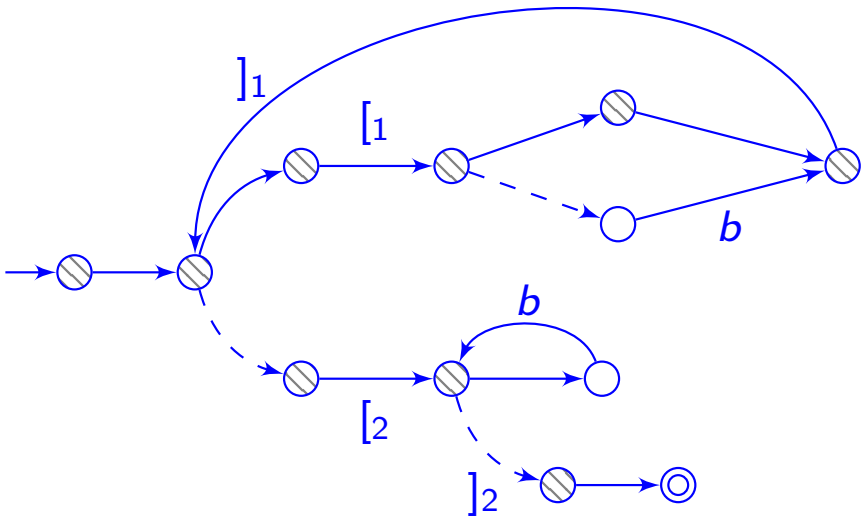


$(F_1^{*?})$



A Java Regular Expression pTr

This is, in fact, the Java pTr for $(\varepsilon \mid b)^*(b^*)$:



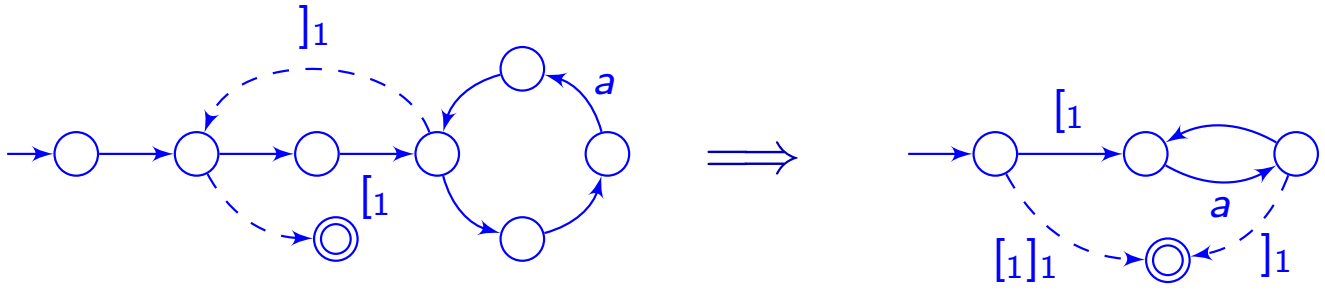
Paper Results 1

A pTr is *flattened* if $\delta_2(q) \in (\Sigma_2^* \times Q_1)^*$ for all $q \in Q_2$

That is: there is no path spending two consecutive steps in Q_2. A very useful normal form

Can be computed in polynomial time, specifically in $\mathcal{O}(|Q_1||\Sigma_1| + |Q_2'||\delta_2|)$ (where Q_2' is a special subset of Q_2)

The flattened pTr is *no larger* than the original



Paper Results 2

An algorithm for checking pTr *equivalence* is given, yielding efficient *equivalence-checking of regexes with capturing groups*

PSPACE-hardness follows from classic results

Paper Results 2

An algorithm for checking pTr *equivalence* is given, yielding efficient *equivalence-checking of regexes with capturing groups*

PSPACE-hardness follows from classic results

For pTr: $\mathcal{O}(|T_1|2^{|Q|} + |T_2|2^{|Q|})$, by checking domain equivalence,converting them to normal *functional transducers*, which we can then quickly equivalence check

This is reapplied to regexes: $2^{\mathcal{O}(|R_1|+|R_2|)}$

Improves existing results; still a gap to be considered!

Paper Results 3

An algorithm is given which given a pTr T and a string w finds the *slim parse output* of running T on w in *linear time*

Slim output is an real-world implementation detail: only the *last occurrence of each* Σ_2 *"bracket"* is output

The overall strategy is to flatten the pTr and recast the depth first search as a *deterministic stack machine with output*, considering all options at once and excising duplicates

The point of comparison: RE2 by Russ Cox at Google, informally takes a similar approach.

Conclusion

Several angles on regular expression matching brought into one *formal* framework

- Can analyze real-world semantics, existing algorithms, and add to the body of algorithms applicable to the regular expressions *actually used*
- Many features of "standard" regular expression library still need to be considered! E.g. reintegrating pruning
- Other practical parsing tools have similar deviations from standard theory: fertile ground for study!

Conclusion

Several angles on regular expression matching brought into one *formal* framework

- Can analyze real-world semantics, existing algorithms, and add to the body of algorithms applicable to the regular expressions *actually used*
- Many features of "standard" regular expression library still need to be considered! E.g. reintegrating pruning
- Other practical parsing tools have similar deviations from standard theory: fertile ground for study!

Thanks for listening!