Регулярни езици. Регулярни ирази. Крайни недетерминирани автомати. Конструиране на краен недетерминиран автомат по регулярен израз

Иво Стратев

9 юни 2020 г.

1 Регулярни езици

1.1 Алгебрична дефиниция

Нека Σ е крайна азбука. Дефинираме понятието регулярен език над Σ с рекурсия.

1.1.1 Базови случаи:

- $\emptyset$ е регулярен език над Σ;
- $\{\varepsilon\}$ е регулярен език над Σ;
- Ако $\mathfrak{u} \in \Sigma$, то $\{\mathfrak{u}\}$ е регулярен език над Σ.

1.1.2 Рекурсивни случаи:

- Ако R е регулярен език над Σ, то R^* е регулярен език над Σ;
- Ако R и S са регулярни езици над Σ, то $\mathsf{R} \cup \mathsf{S}$ е регулярен език над Σ;
- Ако R и S са регулярни езици над Σ, то $\mathsf{R} \cdot \mathsf{S}$ е регулярен език над Σ.

Регулярен език над Σ е всеки **език над** Σ, който може да бъде **получен след краен брой прилагания на горните правила**. Тоест само една част от езиците над Σ са регулярни.

Разбира се всеки краен език може да бъде получен чрез краен брой прилагания на правилата от дефиницията. Например

$$\{abc, cbaa\} = (\{a\} \cdot \{b\} \cdot \{c\}) \cup (\{c\} \cdot \{b\} \cdot \{a\} \cdot \{a\})$$

$$\{aa, aaab\} = (\{a\} \cdot \{a\}) \cdot (\{\varepsilon\} \cup (\{a\} \cdot \{b\}))$$

1.1.3 Забележки:

- Ако A и B са езици над Σ, то $A \cdot B = \{\alpha.\beta \mid \alpha \in A \ \& \ \beta \in B\}$ и $A \cdot B$ е език над Σ.
- Ако L е език над Σ, то $L^* = \bigcup_{n \in \mathbb{N}} L^n$ и L^* е език над Σ.
- Една от причините за да наричаме тази дефиниция алгебрична е, че когато разгледаме операциите * (звезда на Клини), $\cup$ (обединение) и $\cdot$ (конкатенация на езици) само върху езици над фиксирана азбука то можем да гледаме на тях като функции (операции с езици).

2 Регулярни изрази

2.1 Синтаксис

Нека Σ е крайна азбука. Нека $\Gamma = \Sigma \cup \{\emptyset, \varepsilon, (,), *, +, .\}$. **Регулярен израз** над Σ е всяка **дума** над азбуката Γ, която може да бъде **получена чрез креан брой прилагания** на следните правила:

2.1.1 Базови правила:

- думата $\emptyset$ е регулярен израз над Σ;
- думата ε е регулярен израз над Σ;
- Ако u е буква от Σ, то думата с единствена буква u е регулярен израз над Σ.

2.1.2 Рекурсивни правила:

- Ако $\mathtt{r}$ е регулярен израз над Σ, то думата $(\mathtt{r})*$ е регулярен израз над Σ;
- Ако $\mathtt{r}$ и $\mathtt{s}$ са регулярни изрази над Σ, то думата $(\mathtt{r}+\mathtt{s})$ е регулярен израз над Σ;
- Ако $\mathtt{r}$ и $\mathtt{s}$ са регулярни изрази над Σ, то думата $(\mathtt{r}.\mathtt{s})$ е регулярен израз над Σ.

2.1.3 Забележка:

Регулярните изрази над Σ са просто думи от един език над Γ.

2.2 Език на регулярен израз (Семантика)

Нека Σ е азбука. Нека $\mathtt{RegExp}(\Sigma)$ е множеството на думите, които са регулярни изрази над Σ.
Така $\mathtt{RegExp}(\Sigma)$ е език над азбуката $\Sigma \cup \{\emptyset, \varepsilon, (,), *, +, .\}$.
Нека $\mathcal{RL}_\Sigma : \mathtt{RegExp}(\Sigma) \to \mathcal{P}(\Sigma^*)$ дефинираме чрез рекурсия, така:

- $\mathcal{RL}_\Sigma(\emptyset) = \emptyset$;
- $\mathcal{RL}_\Sigma(\varepsilon) = \{\varepsilon\}$;
- Ако $\mathtt{u} \in \Sigma$, то $\mathcal{RL}_\Sigma(\mathtt{u}) = \{\mathtt{u}\}$;
- Ако $\mathtt{r} \in \mathtt{RegExp}(\Sigma)$, то $\mathcal{RL}_\Sigma((\mathtt{r})*) = \mathcal{RL}_\Sigma(\mathtt{r})^*$;
- Ако $\mathtt{r}, \mathtt{s} \in \mathtt{RegExp}(\Sigma)$, то $\mathcal{RL}_\Sigma((\mathtt{r}+\mathtt{s})) = \mathcal{RL}_\Sigma(\mathtt{r}) \cup \mathcal{RL}_\Sigma(\mathtt{s})$;
- Ако $\mathtt{r}, \mathtt{s} \in \mathtt{RegExp}(\Sigma)$, то $\mathcal{RL}_\Sigma((\mathtt{r}.\mathtt{s})) = \mathcal{RL}_\Sigma(\mathtt{r}) \cdot \mathcal{RL}_\Sigma(\mathtt{s})$.

Ако $\mathtt{r}$ е регулярен израз над Σ, то $\mathcal{RL}_\Sigma(\mathtt{r})$ е езикът на регулярния израз $\mathtt{r}$. Така $\mathcal{RL}_\Sigma$ на всеки регулярен израз съпоставя език над Σ, тоест $\mathcal{RL}_\Sigma$ играе роля на семантика за нас (придава значение на всяка дума от фиксиран език).

2.3 Пимери за пресмятане на език на регулярен израз

Нека фиксираме азбука $\Sigma = \{a, b, c\}$ и вместо $\mathcal{RL}_{\{a,b,c\}}$ да пишем само $\mathcal{R}_{\mathcal{L}}$. И нека пресметнем езикът на израза $(((b + c)) * .(c)*)$.

$$\begin{aligned}
&\mathcal{R}_{\mathcal{L}}((((a + b)) * .(c)*)) \\
&= \mathcal{R}_{\mathcal{L}}(((a + b))*) \cdot \mathcal{R}_{\mathcal{L}}((c)*) \\
&= \mathcal{R}_{\mathcal{L}}((a + b))^* \cdot \mathcal{R}_{\mathcal{L}}(c)^* \\
&= (\mathcal{R}_{\mathcal{L}}(a) \cup \mathcal{R}_{\mathcal{L}}(b))^* \cdot \{c\}^* \\
&= (\{a\} \cup \{b\})^* \cdot \{c\}^* \\
&= \{a, b\}^* \cdot \{c\}^*
\end{aligned}$$

Нека направим още две пресмятания.

$$\begin{aligned}
&\mathcal{R}_{\mathcal{L}}(((a.c) + (b.c)) \\
&= \mathcal{R}_{\mathcal{L}}((a.c)) \cup \mathcal{R}_{\mathcal{L}}((b.c)) \\
&= (\mathcal{R}_{\mathcal{L}}(a) \cdot \mathcal{R}_{\mathcal{L}}(c)) \cup (\mathcal{R}_{\mathcal{L}}(b) \cdot \mathcal{R}_{\mathcal{L}}(c)) \\
&= (\{a\} \cdot \{c\}) \cup (\{b\} \cdot \{c\}) \\
&= \{ac\} \cup \{bc\} \\
&= \{ac, bc\}
\end{aligned}$$

$$\begin{aligned}
&\mathcal{R}_{\mathcal{L}}(((a + b).c)) \\
&= \mathcal{R}_{\mathcal{L}}((a + b)) \cdot \mathcal{R}_{\mathcal{L}}(c) \\
&= (\mathcal{R}_{\mathcal{L}}(a) \cup \mathcal{R}_{\mathcal{L}}(b)) \cdot \{c\} \\
&= (\{a\} \cup \{b\}) \cdot \{c\} \\
&= \{a, b\} \cdot \{c\} \\
&= \{ac, bc\}
\end{aligned}$$

Видяхме, че $\mathcal{R}_{\mathcal{L}}(((a.c) + (b.c)) = \{ac, bc\} = \mathcal{R}_{\mathcal{L}}(((a + b).c))$. Следователно $\mathcal{R}_{\mathcal{L}}$ не е инекция! Подобен на разгледания контрапример върши работа за произволна азбука, дори когато тя е само от една буква. Също така предстои да видим, че не е и сюрекция. Тоест не всеки език над Σ

може да бъде получен като образ на някой регулярен език над Σ. Вярно е обаче, че

$$\mathtt{Range}(\mathcal{RL}_\Sigma) = \{L \in \mathcal{P}(\Sigma^*) \mid (\exists r \in \mathtt{RegExp}(\Sigma))(L = \mathcal{RL}_\Sigma(r)\} =$$
$$\{L \in \mathcal{P}(\Sigma^*) \mid \text{"има КТДА } \mathcal{A} \text{ такъв, че } \mathcal{A} \text{ разпознава } L\text{"}\} =$$
$$\{L \in \mathcal{P}(\Sigma^*) \mid (\exists n \in \mathbb{N})(\exists \delta : \Sigma \times S_n \to S_n)(\exists F \subseteq S_n))(L = \mathcal{L}(\langle \Sigma, S_n, \delta, 0, F\rangle))\}$$

Където

- $S_0 = \{0\}$;
- Ако $n \in \mathbb{N}$, то $S_{n+1} = S_n \cup \{n\}$.

Ще го видим чрез примери. Формулирано на човешки език, то звучи така "Автоматните и регулярните езици над фиксирана азбука съвпадат".

3 Крайни недетерминирани автомати

3.1 Дефиниция:

Нека

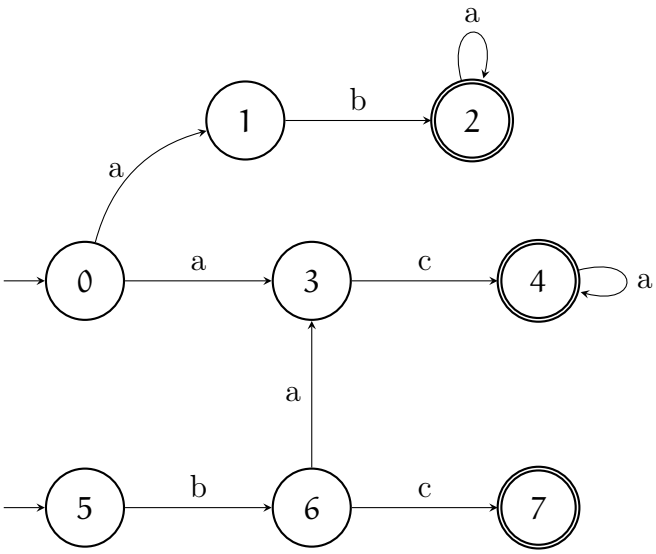
1. Σ е крайна азбука;
2. Q е непразно крайно множество (има поне едно начално състояние);
3. $\Delta \subseteq Q \times \Sigma \times Q$ (тук нямаме тоталност, възможно е за някое състояние q да няма буква x и състояние p, така че да е в сила $\langle q, x, p\rangle \in \Delta$, но пък имаме недетерминираност понеже за някое състояние q и някоя буква x е възможно да има състояния t и p, такива че $\langle q, x, t\rangle \in \Delta$ и $\langle q, x, p\rangle \in \Delta$);
4. $S \subseteq Q$ и $S \neq \emptyset$ (има поне едно начално състояие);
5. $F \subseteq Q$.

Тогава $\langle \Sigma, Q, \Delta, S, F\rangle$ е краен недетерминиран автомат (КНА) над Σ.

3.2 Всеки тотален КДА може да бъде разглеждан като КНА.

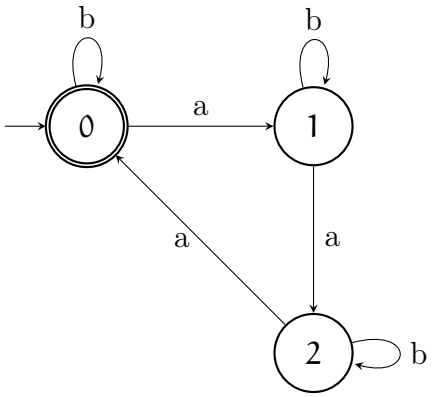
Нека $\mathcal{A} = \langle \Sigma, Q, \delta, s, F \rangle$ е тотален КДА.
Нека $\Delta = \{\langle q, x, \delta(x, q)\rangle \mid q \in Q \ \& \ x \in \Sigma\}$.
Тогава $\langle \Sigma, Q, \Delta, \{s\}, F\rangle$ е $\mathcal{A}$ гледан като КНА.

3.3 Език на КНА



Ако използваме опита ни натрупан покрай работата с тоталните крайни детерминирани автомати би следвало да кажем, че автомата с показаната диграма разпознава думите ab, abaaa, aca, bc и baca, понеже за всяка от тях има маршрут от някое начално състояние до някое финално състояние.

Нека си припомним как формулизирахме разпознаване на дума от един тотален КДА. Нека $\mathcal{A} = \langle \{a, b\}, \{0, 1, 2\}, \delta, 0, \{0\}\rangle$ е автомата представен на диаграмата:



Тогава $\mathcal{A}$ разпознава една дума $\omega \in \{a, b\}^*$ ТСТК $\delta^*(0,\ \omega) \in \{0\}$.
Където δ^*, дефинирахме така:

- Ако $q \in \{0, 1, 2\}$, то $\delta^*(q,\ \varepsilon) = q$;
- Ако $q \in \{0, 1, 2\}$ и $x \in \{a, b\}$ и $\alpha \in \{a, b\}^*$, то $\delta^*(q,\ x\alpha) = \delta^*(\delta(q, x),\ \alpha)$.

Чрез рекурсивните пресмятания на δ^* по автомата $\mathcal{A}$ реално се извършва едно напълно детерминистичо изчисление, което остава може би леко скрито. Така за един тотален КДА можем да мислим като хардуерно реализирана програма. Тоест едно устройство с крайна памет, на което подаваме дума и то отговаря с да или не. Някакъв вид подобрен **if statement**. Ще използваме идеята за изчисление за да дефинираме напълно формално какво значин един КНА да разпознава даден език.

3.4 Език на КНА

Нека $\mathcal{N} = \langle \Sigma, Q, \Delta, S, F \rangle$ е КНА. Експлицитно ще кажем какво значи за нас изчисление по автомата $\mathcal{N}$. Целта ни е по състояние и дума да кажем какво значи успещно изчисление, тоест изчисление, което ни води до финално състояние. Не се ограничаваме до изчисление започващо от начално състояние, защото в хода на изчислението моментното състоянието вероятно ще се промения.

Изчисление започващо от съсояние t с дума ω по автомата $\mathcal{N}$, до финално състояние p ще бележим с $\langle t, \omega \rangle \vdash_{\mathcal{N}} p$. Въвеждаме следните изчислителни правила:

$$\frac{q \in F}{\langle q, \varepsilon \rangle \vdash_{\mathcal{N}} q}$$

$$\frac{x \in \Sigma \quad \alpha \in \Sigma^* \quad q \in Q \quad t \in Q \quad f \in F \quad \langle q, x, t\rangle \in \Delta \quad \langle t, \alpha\rangle \vdash_{\mathcal{N}} f}{\langle q, x\alpha\rangle \vdash_{\mathcal{N}} f}$$

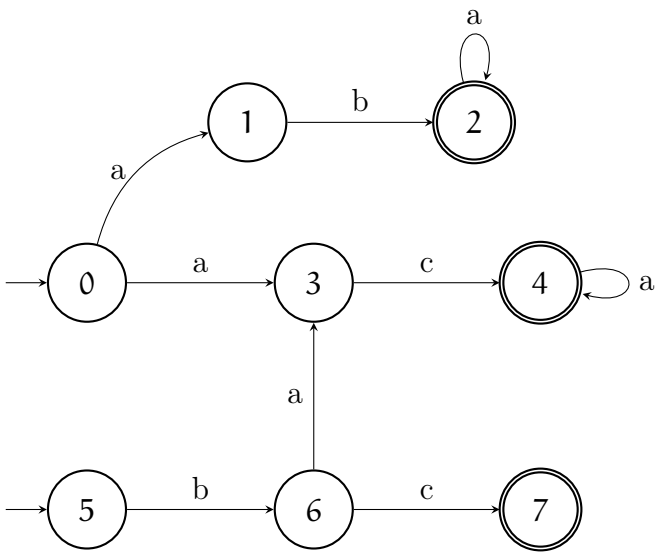
Тогава езикът разпознават от КНА $\mathcal{N}$ е

$$\mathcal{L}_{\mathcal{ND}}(\mathcal{N}) = \left\{ \omega \in \Sigma^* \mid \underbrace{(\exists s \in S)(\exists f \in F)\ \langle s,\ \omega\rangle \vdash_{\mathcal{N}} f}_{\mathcal{N} \text{ разпознава думата } \omega} \right\}$$

Тоест езикът разпознаван от $\mathcal{N}$ е множеството на точно онези думи, за които има успешно изчисление започващо от някое начално състояние.

3.5 Два примера за успешни изчисления.

Отново разглеждаме КНА с диаграма



Автоматът разпознава думата bc, защото

$$\langle 7, \varepsilon\rangle \vdash_{\mathcal{N}} 7$$
$$\langle 6, c\varepsilon\rangle \vdash_{\mathcal{N}} 7$$
$$\langle 5, bc\varepsilon\rangle \vdash_{\mathcal{N}} 7$$

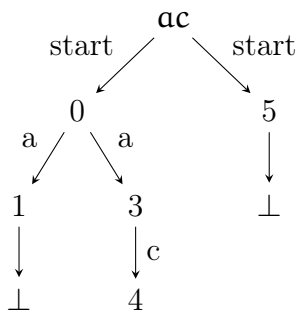
Тоест $\langle 5, bc\rangle \vdash_{\mathcal{N}} 7$.

Също така автоматът разпознава думата $\mathtt{acaa}$, защото

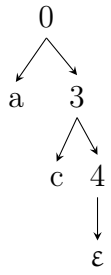
$$\begin{aligned}\langle 4, \varepsilon\rangle &\vdash_{\mathcal{N}} 4\\ \langle 4, \mathtt{a}\varepsilon\rangle &\vdash_{\mathcal{N}} 4\\ \langle 4, \mathtt{aa}\varepsilon\rangle &\vdash_{\mathcal{N}} 4\\ \langle 3, \mathtt{caa}\varepsilon\rangle &\vdash_{\mathcal{N}} 4\\ \langle 0, \mathtt{acaa}\varepsilon\rangle &\vdash_{\mathcal{N}} 4\end{aligned}$$

и $\mathtt{acaa} = \mathtt{acaa}\varepsilon$.

Задачата за разпознаване на дума от един КНА реално е задача за търсене, защото може да бъде моделирана като такава. За целта се използват дървета на търсене и на извод. Сега ще дадем примери за няколко примера за двата вида дървета за различни думи.



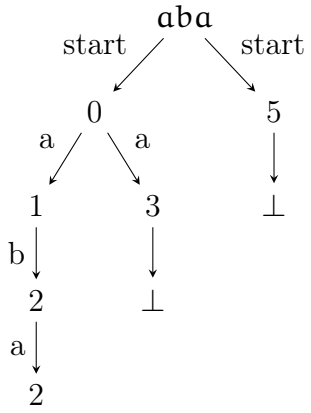
Фигура 1: Дърво на търсене за думата $\mathtt{ac}$



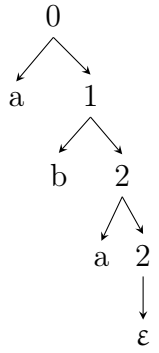
Фигура 2: Дърво на извод за думата $\mathtt{ac}$

Идеята на дървото на търсене по дадена дума е да визуализира всевъзможните изчислени по дадена дума. Тоест започвайки от всяко начално състояние на автомата правейки всевъзможните последователни преходи от състояние в състояние по буквите на думата да видим достиганията на финални състояния. Използваме $\perp$ за изчисления, които пропадат, тоест не е прочетена цялата дума и от текущото състояние не е възможен преход с текущата буква.

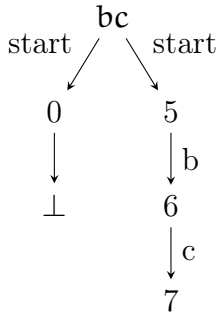
Едно дърво на извод на една дума показва един възможен начин за получаване на думата. Такива дървета ще използваме при контекстно свободните граматики, за това сега само ги показваме с идеята да се навържат нещата по-късно ("to see the bigger picture later on").



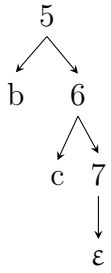
Фигура 3: Дърво на търсене за думата aba



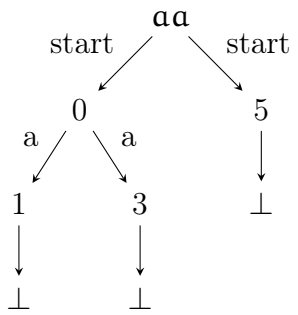
Фигура 4: Дърво на извод за думата $\mathtt{baa}$



Фигура 5: Дърво на търсене за думата $\mathtt{bc}$

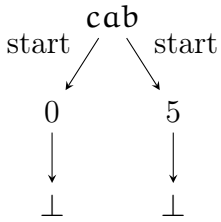


Фигура 6: Дърво на извод за думата $\mathtt{bc}$



Фигура 7: Дърво на търсене за думата aa

Това, което виждаме е, че всяко изчисление с думата aa по разглежданият автомат е неуспешно / пропадащо .



Фигура 8: Дърво на търсене за думата cab

Това, което е важно да се запомни е, че за да приеме автомата една дума е достатъчно едно успешно изчисление по дадената дума. Ако всяко изчисление не е успешно, то думата не е разпозната от автомата и не е от неговия език.

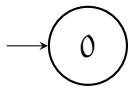
4 Конструиране на краен недетерминиран автомат по регулярен израз

Конструирането следва дефиницията за регулярен израз. Първо показваме базови конструкции.

4.1 Базови конструкции

4.1.1 Празният език

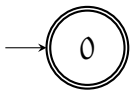
В празният език няма думи, така че не трябва да има финално състояние в автомат, който ще го разпознава. От друга страна трябва да имаме поне едно начално състояние. Така, че един автомат за празният език е



Формално $\langle \Sigma, \{0\}, \emptyset, \{0\}, \emptyset \rangle$.

4.1.2 Езикът на празната дума

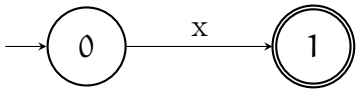
Езикът на прзната дума съдържа само ε, която няма букви. Трябват ни начално и финално състояние. Хубаво е автоматът да е с минимален брой състояния. Така, че само едно състояние, което е и начално и финално е достатъчно. Ако имаме преход с буква ще разпознаваме повече думи от празната. Така, че няма да имаме никакви преходи и така автомат за езикът на празната дума е



Формално $\langle \Sigma, \{0\}, \emptyset, \{0\}, \{0\} \rangle$.

4.1.3 За буква

Нека $x \in \Sigma$. Искаме автомат, който разпознава думата с единствена буква x и никоя друга дума. Така достигаме до следния автомат



Формално $\langle \Sigma, \{0, 1\}, \{\langle 0, x, 1 \rangle\}, \{0\}, \{1\} \rangle$.

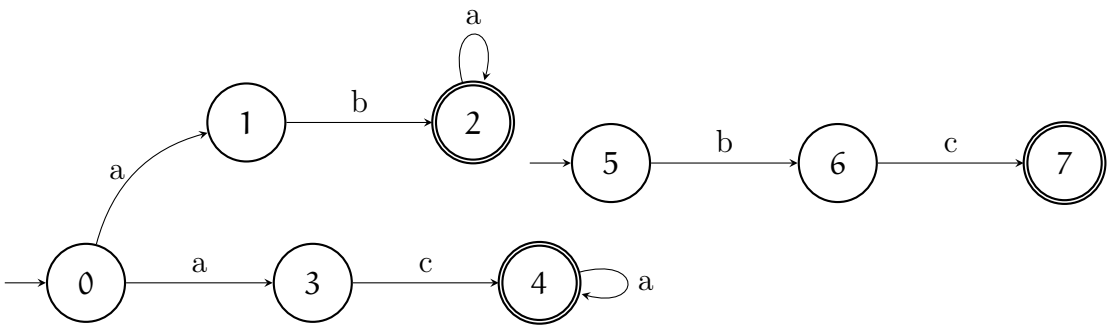
4.2 Конструкции съотвестващи на рекурсивните правила

Фиксираме азбука Σ.

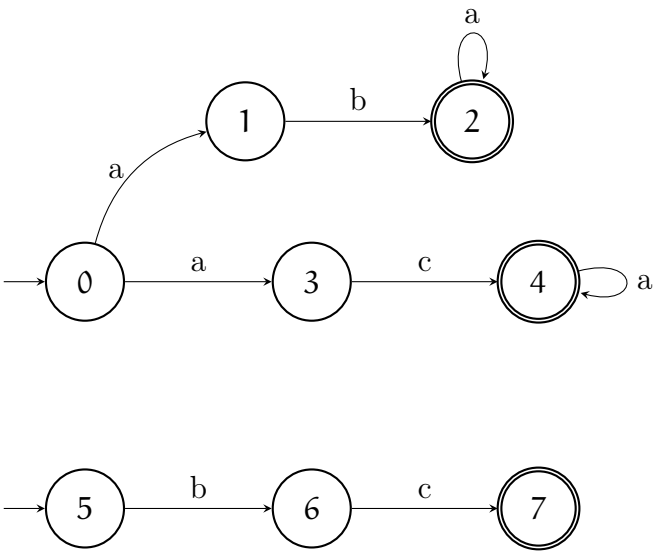
4.2.1 За +

Нека $r, s \in \mathtt{RegExp}(\Sigma)$. Тогава $\mathcal{RL}_\Sigma((r+s)) = \mathcal{RL}_\Sigma(r) \cup \mathcal{RL}_\Sigma(s)$. Значи искаме КНА, който разпознава обединието на двата езика. Но ако състоянията на автоматите разпознаващи двата езика са непресичащи се множества, то ако някак си направим обединие на двата автомата ще сме получили автомат, който ни върши работа.

Нека разгледаме следните два автомата зададени чрез свойте диаграми:



За да ги обединим интуитивно е достатъчно да направим обща диграма на двата автомата.



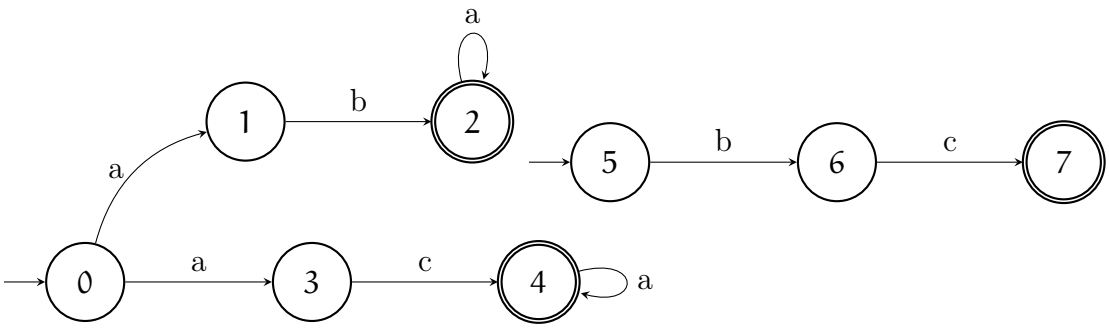
Нека сега да формализираме. Нека

$$\mathcal{N}_1 = \langle \Sigma, Q_1, \Delta_1, S_1, F_1 \rangle$$
$$\mathcal{N}_2 = \langle \Sigma, Q_2, \Delta_2, S_2, F_2 \rangle$$

са два НКА съответно за регулярните изрази r_1 и r_2.
Тоест $\mathcal{L}_{\mathcal{ND}}(\mathcal{N}_1) = \mathcal{RL}(r_1)$ и $\mathcal{L}_{\mathcal{ND}}(\mathcal{N}_2) = \mathcal{RL}(r_2)$. Тогава
$\mathcal{N} = \langle \Sigma,\ Q_1 \cup Q_2,\ \Delta_1 \cup \Delta_2,\ S_1 \cup S_2,\ F_1 \cup F_2 \rangle$ е такъв, че
$\mathcal{L}_{\mathcal{ND}}(\mathcal{N}) = \mathcal{RL}((r_1{+}r_2)) = \mathcal{RL}(r_1) \cup \mathcal{RL}(r_2)$.

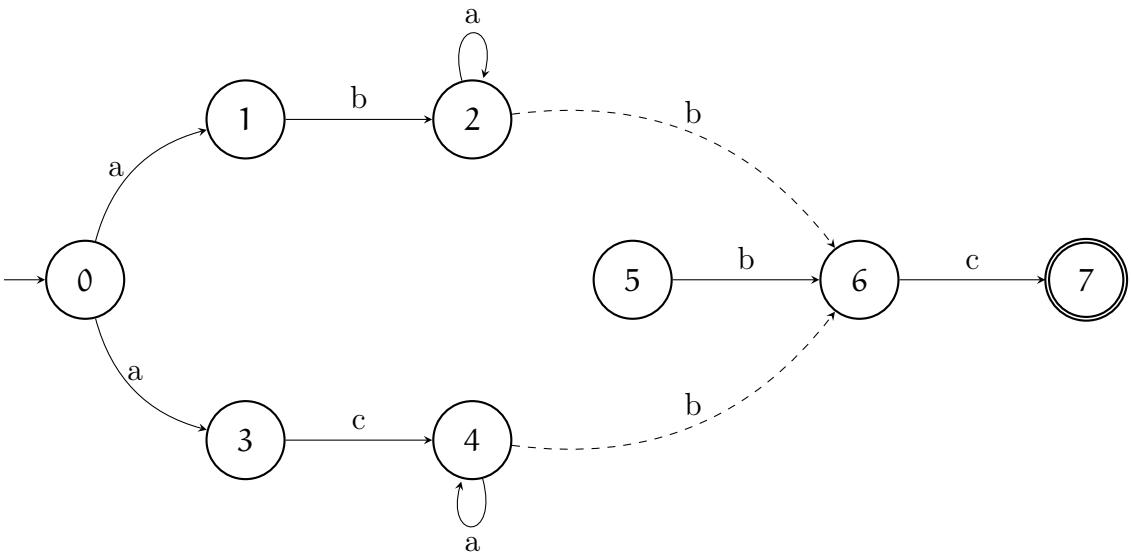
4.2.2 За ·

Нека $r, s \in \mathsf{RegExp}(\Sigma)$. Тогава $\mathcal{RL}_\Sigma((r{\cdot}s)) = \mathcal{RL}_\Sigma(r) \cup \mathcal{RL}_\Sigma(s)$. Искаме да направим автомат разпознаващ конкатенацията на двата езика. Нека разгледаме диаграмите на същите два автомата като в примера за +.



Интиуитивно за конкатенацията трябва да навържем последователно двата автомата. Така вървейки по този от ляво ще разпознаем дума от първият език, след това ще се прехвърлим на десният и ще продължим да четем дума от вторият. Така общо ще сме прочели дума, която е конкатенация на дума от левия с дума от десният. Значи трябва да добавим преходи от финалните на този от ляво, към състония на този от дясно. Обаче ако напрвим преходи към началното състояние на този от дясно, то ще прочетем допълнително още една буква. Значи от финлно на този от ляво трябва да се прехвърлим в сътояние на вторият съответстващо на преход от началното състояние. Тоест да започнем да четем дума от езикът на този в дясно, прочитайки първата буква и после да продължим нейното допрочитане. Например за думата $\mathtt{ababc}$ ще искаме да направим преходите $0 \to 1 \to 2 \to 2 \to 6 \to 7$. Тоест вместо преходът $5 \to 6$ ще искаме да можем да направим преход $2 \to 6$. На ниво диаграма

трябва слеем двата автомата в един и да добавим преходите отбелязани с пунктир.



Добре ще слеем двата автомата, но трябва да кажем, кои ще са началните и кои ще са финалните състояния. Невежо ще си кажем начални ще са началните състояния на този от ляво, а финални ще са финалните на тези от дясно. Проблем би настъпил ако в езикът на десният беше и празната дума, защото тогава всяка дума от езикът на левият ще бъде в конкатенацията, значи в такава ситуация финални сътояния трябва да са финалните и на двата автомата.

Нека сега разгледаме ситуацията когато в езикът на левият автомат беше празната дума, дали тогава началните състояния на десния не трябва да са начални и в този за конкатенацията ? Отговорът на този въпрос е отрицателен, защото тогава левит ще има начално състояние, което е финално и тогава ще сме копирали и от него началните преходи в десният автомат, тоест ще има от къде да започнем директно да правим изчисление по десният автомат, ако думата е непразна разбира се.

Нека да формализираме горните наблюдения. Нека

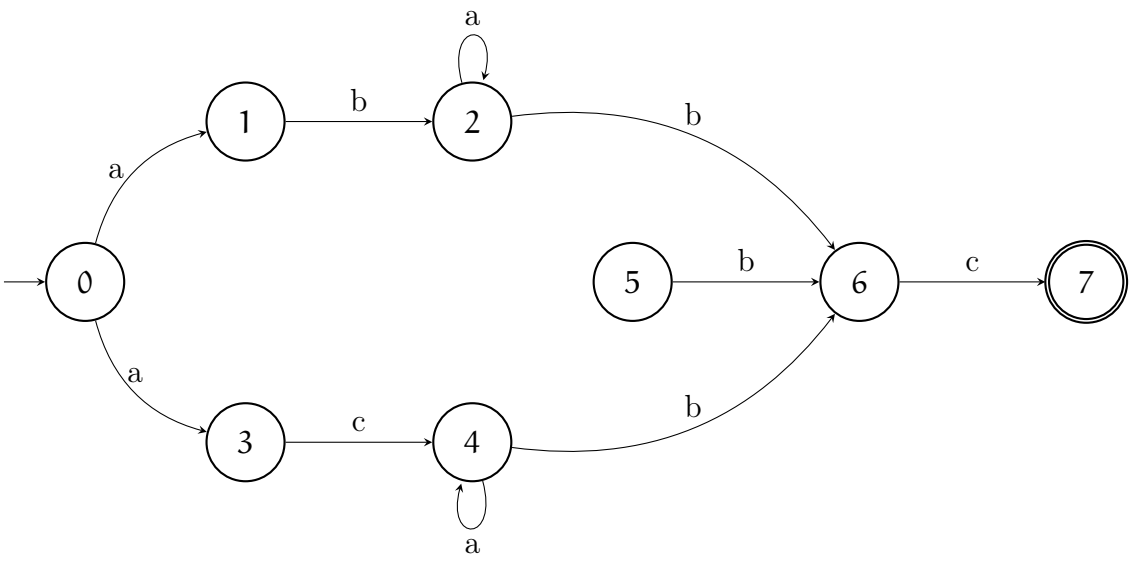
$$\mathcal{N}_1 = \langle \Sigma, Q_1, \Delta_1, S_1, F_1 \rangle$$
$$\mathcal{N}_2 = \langle \Sigma, Q_2, \Delta_2, S_2, F_2 \rangle$$

са два НКА съответно за регулярните изрази r_1 и r_2.
Тоест $\mathcal{L}_{\mathcal{ND}}(\mathcal{N}_1) = \mathcal{RL}(r_1)$ и $\mathcal{L}_{\mathcal{ND}}(\mathcal{N}_2) = \mathcal{RL}(r_2)$.
Тогава $\mathcal{N} = \langle \Sigma,\ Q_1 \cup Q_2,\ \Delta_1 \cup \Delta_2 \cup \Delta_{\text{bridge}},\ S_1,\ F \rangle$, където

- $\Delta_{\text{bridge}} = \{\langle f, x, q\rangle \in F_1 \times \Sigma \times Q_2 \mid (\exists s \in S_2)(\langle s, x, q\rangle \in \Delta_2)\}$
- $F = \begin{cases} F_1 \cup F_2 & S_2 \cap F_2 \neq \emptyset \\ F_2 & S_2 \cap F_2 = \emptyset \end{cases}$

е такъв, че $\mathcal{L}_{\mathcal{ND}}(\mathcal{N}) = \mathcal{RL}((r_1.r_2)) = \mathcal{RL}(r_1) \cdot \mathcal{RL}(r_2)$.

Ако следваме точно този формализъм, то за разгледаният пример получаваме



Това, което можем да забележим е, че състояние 5 е "висящо по-точно е недостижимо състояние. Такива състояния са излишни и могат да бъдат премахнати без проблем. Нека променим въведената конструкция, така че всички привидно (само стартови) недостижи състояния са премахнати.

Нека

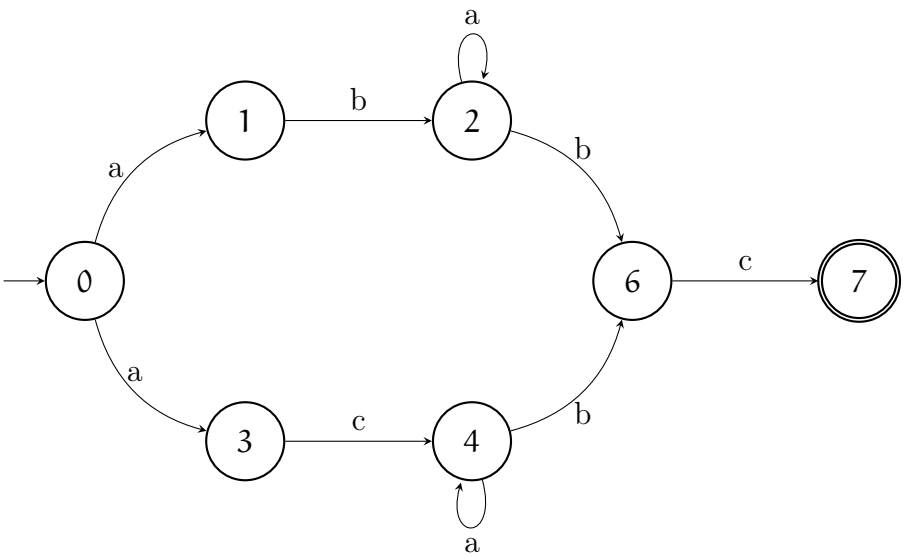
$$\mathcal{N}_1 = \langle \Sigma, Q_1, \Delta_1, S_1, F_1 \rangle$$
$$\mathcal{N}_2 = \langle \Sigma, Q_2, \Delta_2, S_2, F_2 \rangle$$

са два НКА съответно за регулярните изрази r_1 и r_2.
Тоест $\mathcal{L}_{\mathcal{ND}}(\mathcal{N}_1) = \mathcal{RL}(r_1)$ и $\mathcal{L}_{\mathcal{ND}}(\mathcal{N}_2) = \mathcal{RL}(r_2)$.
Тогава $\mathcal{N} = \langle \Sigma,\ Q_1 \cup (Q_2 \setminus U),\ \Delta_1 \cup (\Delta_2 \setminus \Delta_u) \cup \Delta_{bridge},\ S_1,\ F \rangle$, където

- $U = \{q \in S_2 \mid \neg(\exists x \in \Sigma)(\exists p \in Q)(\langle p, x, q \rangle \in \Delta_2)\}$
- $\Delta_u = \Delta_2 \cap (U \times \Sigma \times Q_2)$
- $\Delta_{bridge} = \{\langle f, x, q \rangle \in F_1 \times \Sigma \times Q_2 \mid (\exists s \in S_2)(\langle s, x, q \rangle \in \Delta_2)\}$
- $F = \begin{cases} F_1 \cup (F_2 \setminus U) & S_2 \cap F_2 \neq \emptyset \\ F_2 \setminus U & S_2 \cap F_2 = \emptyset \end{cases}$

е такъв, че $\mathcal{L}_{\mathcal{ND}}(\mathcal{N}) = \mathcal{RL}((r_1.r_2)) = \mathcal{RL}(r_1) \cdot \mathcal{RL}(r_2)$.

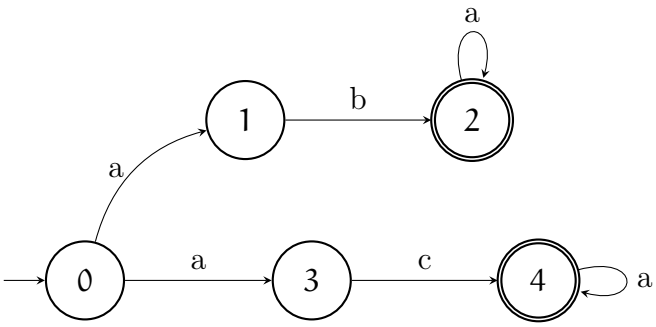
Така автоматът за конкатенация от разгледаният пример е:



Разбира се оптимизираната конструкция вероятно се помни по-трудно, важна е идеята зад нея. За тази цел човек може да следва по-простата (първоначалната) конструкция и след това поглеждайки резултатният автомат лесно може да съобрази, кои бивши начални състояния на автомата от дясно са излишни и безпроблемно могат да бъдат премахнати.

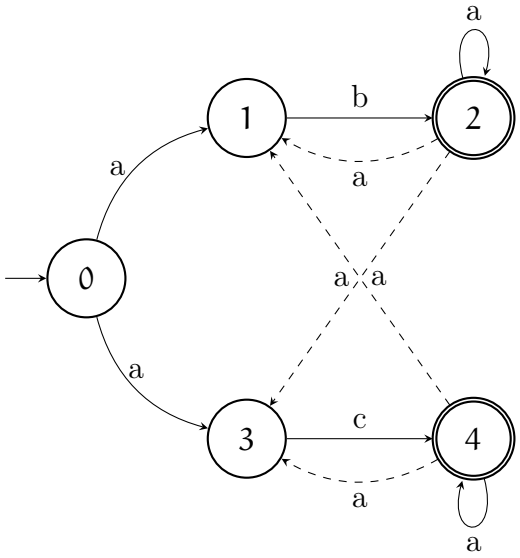
4.2.3 За $*$:

Нека се опитаме да обясним идеята зад тази конструкция като разгледаме следният автомат:

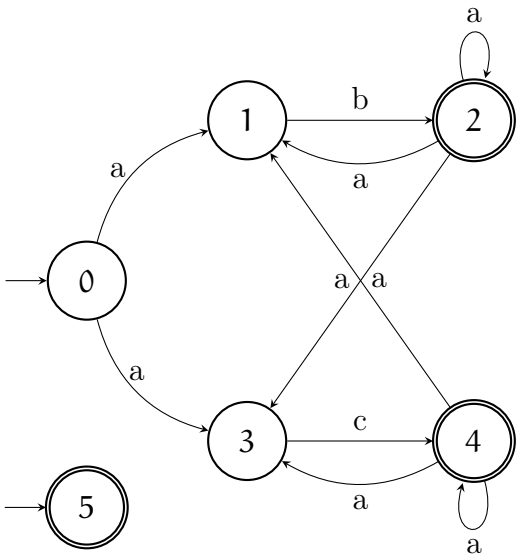


Нека езикът разпознаван от разглежданият автомат е L. Както знаем $L^* = \bigcup_{n \in \mathbb{N}} L^n$, където $L^0 = \{\varepsilon\}$ и ако $k \in \mathbb{N}$, то $L^{k+1} = L^k \cdot L$.

Така за да разпознаем дума от L^* интуитивно трябва да направим следното: след като прочетем дума от L да имаме възможост да приключим с четенето или да започем да четем отново дума от L. Това можем да го постигнем като навържем автомата със себе си. Тоест да симулираме преходите от началните състояния ($\{0\}$) от всяко финално състояние ($\{2, 4\}$). Или да добавим преходите отбелязани с пунктир от картинката:



С този автомат обаче не разпознаваме празната дума, която твърдо е в езика L^*. За това добавяме ново начално състояние, което да е и финално и получаваме:



Така ако трябва да опишем с думи конструкцията тя звучи така: Навързваме автомата сам съм себе си като копираме преходите от началните състояния във финалните и добавяме ново начално състояние,

което е и финално, ако не е имало начално състояние, което да е и финално (празната дума не е в L) в автомата, на който правим $*$.

Сега да формализираме:
Нека $\mathcal{N} = \langle \Sigma, Q, \Delta, S, F \rangle$ такъв, че $\mathcal{L}_{\mathcal{ND}}(\mathcal{N}) = \mathcal{RL}(r)$.
Тогава $\mathcal{N}' = \langle \Sigma,\ Q',\ \Delta \cup \Delta_{copy},\ S',\ F' \rangle$, където

$$t \notin Q$$

$$Q' = \begin{cases} Q & , S \cap F \neq \emptyset \\ Q \cup \{t\} & , S \cap F = \emptyset \end{cases}$$

$$\Delta_{copy} = \{\langle f, x, q \rangle \mid f \in F \ \&\ x \in \Sigma \ \&\ q \in Q \ \&\ (\exists s \in S)(\langle s, x, q \rangle \in \Delta)\}$$

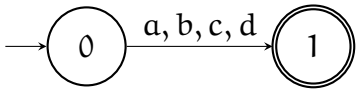
$$S' = \begin{cases} S & , S \cap F \neq \emptyset \\ S \cup \{t\} & , S \cap F = \emptyset \end{cases}$$

$$F' = \begin{cases} F & , S \cap F \neq \emptyset \\ F \cup \{t\} & , S \cap F = \emptyset \end{cases}$$

5 Разрешени оптимизации

5.1 + на букви

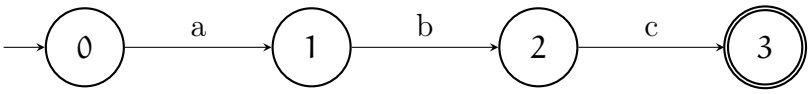
Когато например имаме регулярен израз, който представлява + на букви. Например $(a + (b + (c + d)))$, то можем директно да направим автомат разпознаващ същият език като езикът на регулярният израз. Като имаме две състояния - начално и финално и с всяка буква имаме преход от началното към финалното. За примера чрез диаграма:



5.2 . на букви

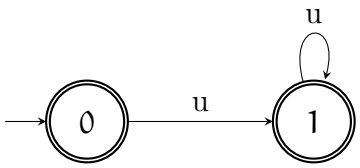
Когато например имаме регулярен израз, който представлява . на букви. Например $(a \cdot (b \cdot c))$, то можем директно да направим автомат разпознаващ същият език като езикът на регулярният израз. Като ако буквите са n на брой в израза, то правим автомат с $n + 1$ състояния и преходи

$\langle q_i, x_i, q_{i+1} \rangle$ като първото състояние е началното, а последното финално. За примера $(a \cdot (b \cdot c))$ чрез диаграма:



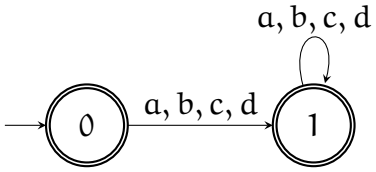
5.3 За израз, който е $*$ на буква или $+$ на букви

Ако имаме израз $(u)*$, то директно можем да направим автомат:



Защото езикът на изразът $(u)*$ е $\{u\}^*$ или $\{u^n \mid n \in \mathbb{N}\}$.

Аналогично за израз $((a + (b + (c + d)))*)$ можем директно да направим автомат:



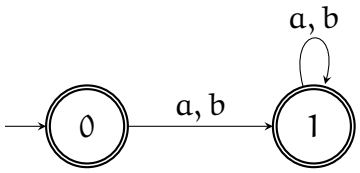
Защото езикът на изразът $((a + (b + (c + d)))*)$ е $\{a, b, c, d\}^*$ или множеството на всички думи над азбуката $\{a, b, c, d\}$.

5.4 Примери за конструиране на КНА по регулярен израз

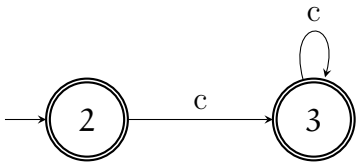
5.5 За изразът $(((a + b)) * .(c)*)$

Започваме да стройм автомата от долу на горе, използвайки споменатите оптимизации.

Първо за изразът $((a + b))*$:

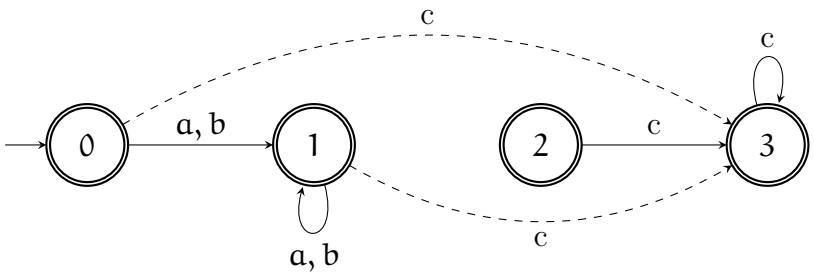


След това за изразът $(c)*$:

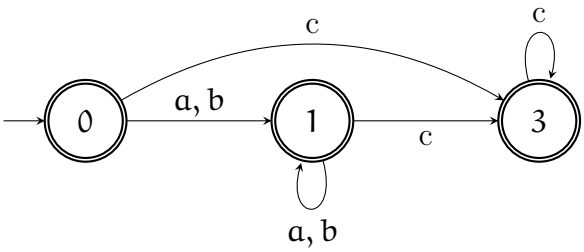


За да направим автомат за конкатенацията, удобо е да наредим двата автомата един до друг като запазим наредбата.

Сега копираме преходите начално състояние, буква, състояние от десният от всяка финално състояние на левият.



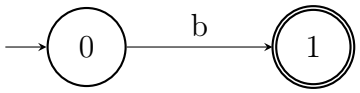
Забелязваме, че състояние 2 е недостижимо и може да бъде премахнато. Така получаваме КНА:



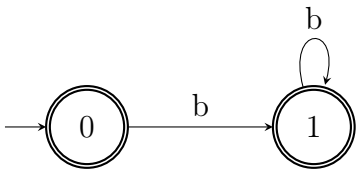
5.6 За изразът $((\mathtt{a}*).(\mathtt{b}.(\mathtt{b}*)))$

Като използваме въведените конвенции да пропускаме скоби и точки изразът може да бъде записан като $\mathtt{a}^*\mathtt{b}\mathtt{b}^*$. При последователни конкатенации, може би уробдно е да се кара от дясно на ляво.

Авомат за $\mathtt{b}$

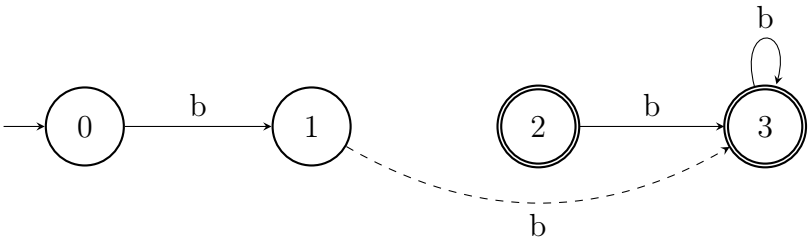


За $(\mathtt{b}*)$

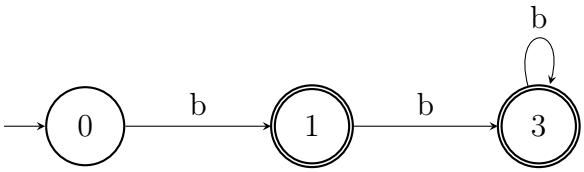


Отново рисуваме двата автомата един до друг като запазваме реда, но променя състоянията на този от дясно, така че двете множества да са непресичащи.

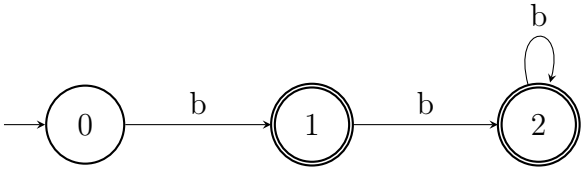
Копираме прехода $\langle 2, \mathtt{b}, 3\rangle$ в състояние 1.



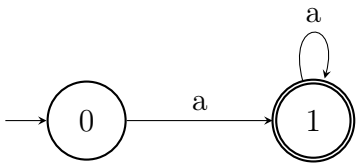
Състояние 2 е недостижимо за това го премахваме.



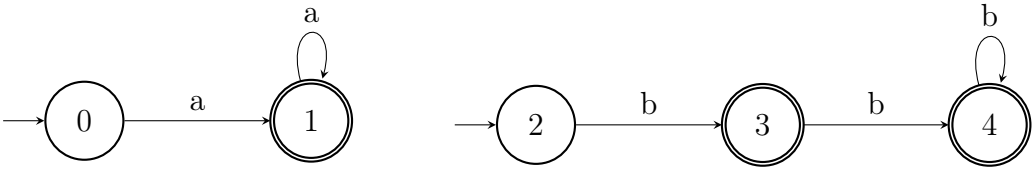
С цел удобство променяме индексацията на състоянията.



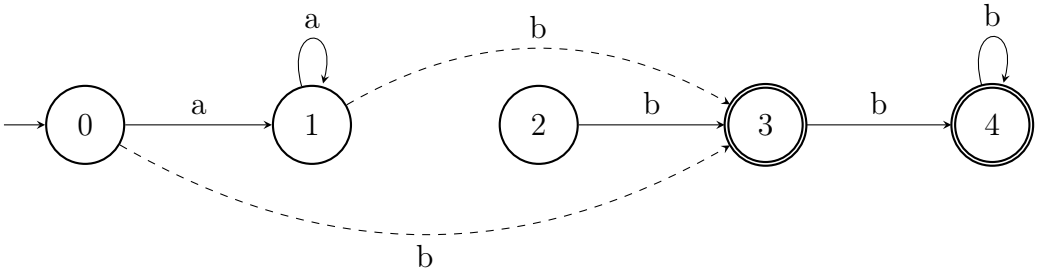
Автомат за $(\mathtt{a}*)$.



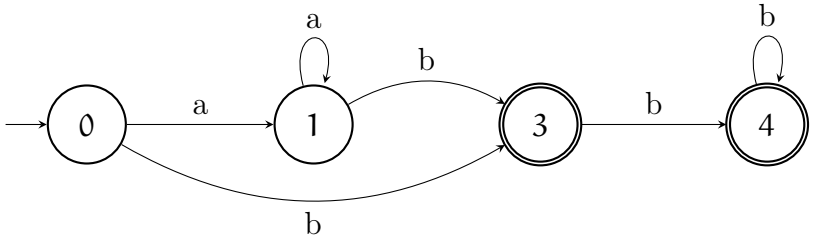
Правим за $((\mathtt{a}*).(\mathtt{b}.(\mathtt{b}*)))$. Слагаме двата автомата един до друг като правим състоянията последователни.



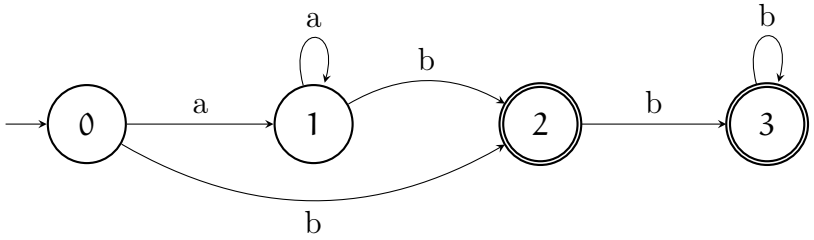
Копираме прехода $\langle 2, \mathtt{b}, 3\rangle$ от състояние 1.



Състояние 2 е недостижимо за тов го премахваме.

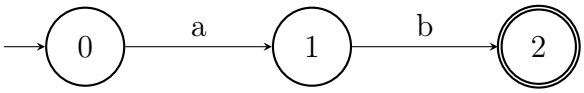


Така автомат разпознаващ езикът на регулярния израз $((a*).(b.(b*)))$.

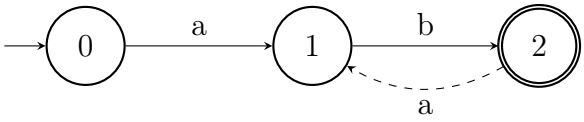


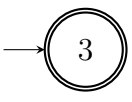
5.7 За израза $((a + (((a.b)*).b))*)$

За $(a.b)$.

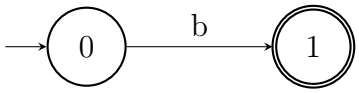


За $((a.b)*)$. Копираме преходът $\langle 0, a, 1\rangle$ от състояние 2 и добавяме ново финално състояние, което е и финално.

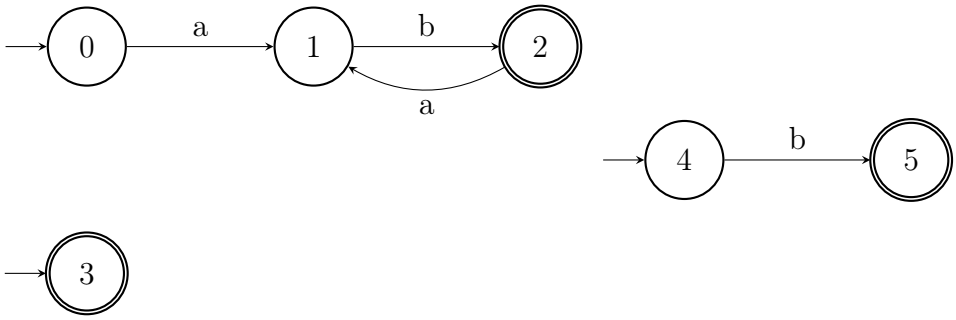




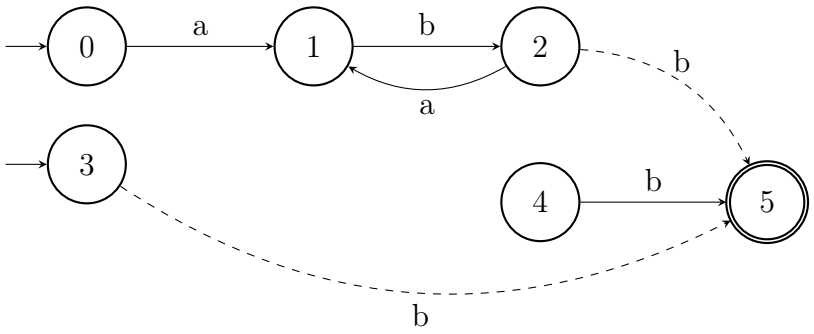
За $\mathtt{b}$.



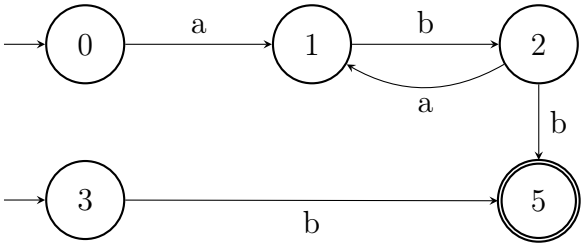
За $(((\mathtt{a}.\mathtt{b})*).\mathtt{b})$ слагаме двата автомата един до друг и ги правим с полседователни състояния.



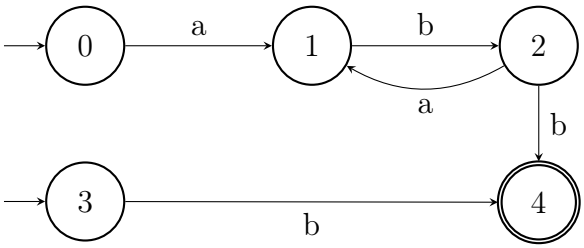
Свързваме ги.



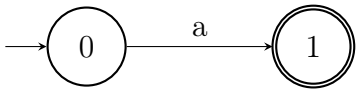
Премахваме състояние 4.



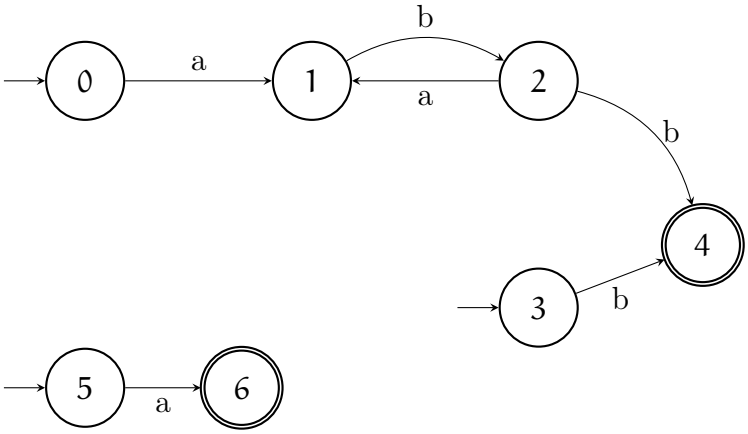
Правим състоянията последователни.



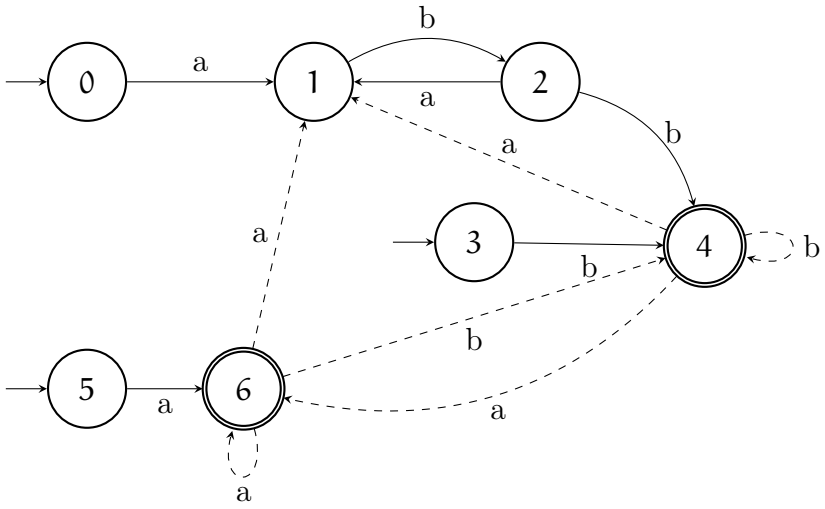
Автомат за регулярния израз a.



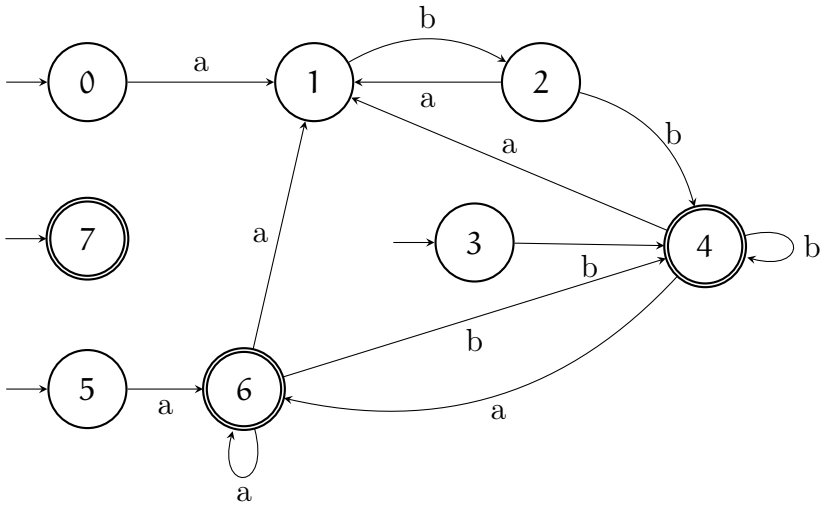
За $(a + (((a.b)*).b))$ обединяваме двата автотмата като правим множествата от състояния непресичащи се.



За $((a + (((a.b)*).b))*)$ първо свързваме циклично предният автомат. Тоест копираме преходите от началните състояния във всяко финално.

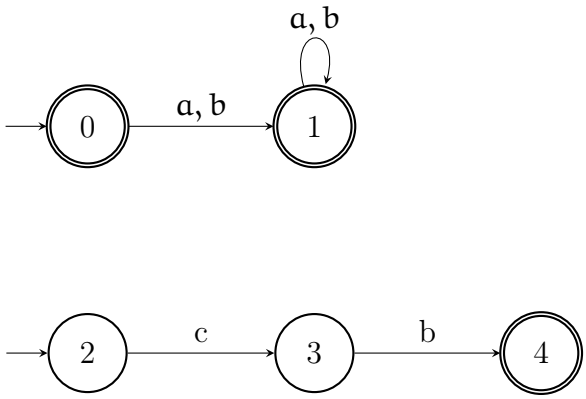


В автоматът за $(a + (((a.b)*).b))$ няма начално състояние, което е финално. За това добавя ново начално състояние 7, което е и финално състояние, за да разпознаваме и празната дума.

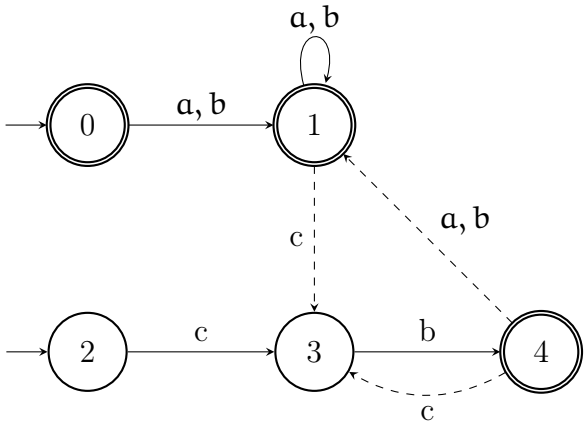


5.8 За изразът (((((a + b)*) + (c.b))*.a)*)

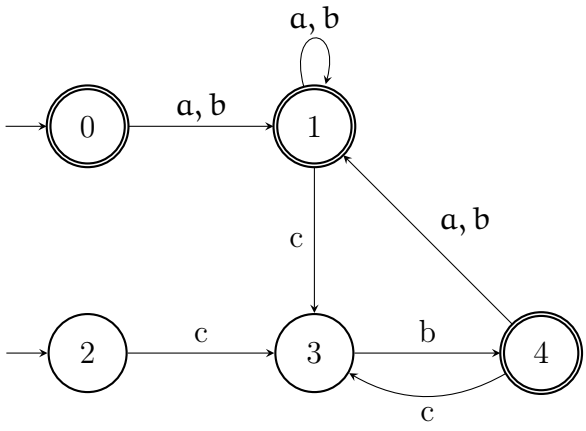
Директно можем да направим автомат за $(((a + b)*) + (c.b))$ следвайки конструкциите, но пропускайки няколко стъпки.



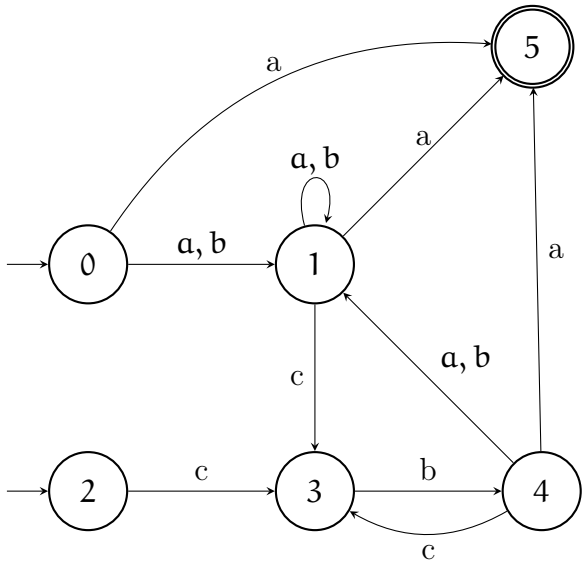
За $((((a + b)*) + (c.b))*)$ навързваме автомата циклично.



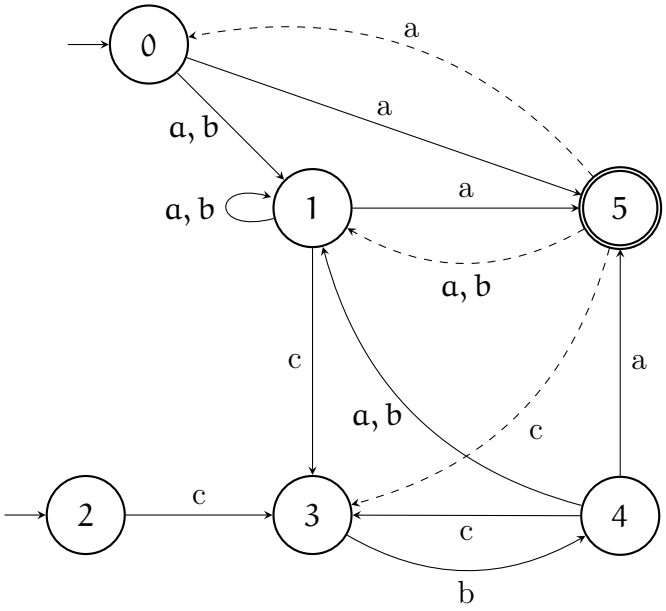
Състояние 0 е и начално и финално, за това не добавяме ново състояние. Така автоматът за $((((a + b)*) + (c.b))*)$ е



За $(((((a + b)*) + (c.b))*).a)$ добавяме ново състояние, което става единственото финално и добавяме преходи от всяко старо финално, към това ново състояние с буквата a.



За $((((((a + b)*) + (c.b))*).a)*)$ навързваме автомата циклично.



Съобразяваме, че не разпознаваме празната дума за това добавяме ново начално състояние, което е и финално.