Об одном равномерном свойстве суперкомпилятора SCP4

(построение специализированных версий алгоритма Матиясевича
посредством
специализации наивного поиска образца в строке)

Андрей П. Немытых
Институт программных систем РАН

г. Переславль-Залесский

Совместное рабочее совещание ИПС РАН и МГТУ имени Н.Э. Баумана
по функциональному языку программирования Рефал
16 июня 2022 г., Москва

Пусть даны:

- **функциональный язык программирования $\mathcal{L}$;**
- **программная модель P на языке $\mathcal{L}$ наивного поиска образца в строке;**
- **метод автоматической специализации программ на языке $\mathcal{L}$.**

Может ли данный метод специализации преобразовать программу P к алгоритму Матиясевича для любого фиксированного образца и произвольной неизвестной строки?

Мы будем интересоваться:

- **функциональным языком программирования $\mathcal{L}$ = Рефал;**
- **программной моделью P в терминах хвостовой рекурсии наивного поиска образца в строке;**
- **методом автоматической специализации программ = суперкомпиляцией.**

Может ли данный метод специализации преобразовать программу P к специализированной версии алгоритма Матиясевича для любого фиксированного образца и произвольной неизвестной строки?

Определение. **(Частичная) функция $\mathrm{g(y) : M \rightharpoonup D}$ называется (частичной) подфункцией (частичной) функции $\mathrm{f(x, y) : N \times M \rightharpoonup D}$, если $\mathrm{\exists\, x_0 \in N}$ такое, что $\mathrm{\forall\, y \in M.\ g(y) = f(x_0, y)}$.**

Простейшая задача на специализацию:
Даны программа $\mathrm{P(x, y)}$, реализующая частичную функцию $\mathrm{f(x, y) : D \times D \rightharpoonup D}$, и $\mathrm{x_0 \in D}$.
Требуется автоматически построить программу $\mathrm{Q(y)}$ такую, что:

(1) $\mathrm{\forall\, y \in D}$, на которых $\mathrm{P(x_0, y)}$ определена, т.е. возвращает результат $\mathrm{[\![P(x_0, y)]\!]}$, $\mathrm{[\![Q(y)]\!] = [\![P(x_0, y)]\!]}$;

(2) и $\mathrm{T([\![Q(y)]\!]) \le T([\![P(x_0, y)]\!])}$,
где T - функция «времени» вычисления результата.

Пример

Образец: 'колокол'

Строка: 'колколоколоколколколоколкол'

Мы будем интересоваться предикатом:
Входит ли данный образец в данную строку?

'колколоколоколколколоколкол'
'колокол' 'колокол' 'колокол'

– программная модель P на Рефале наивного поиска образца в строке;

– P написана в терминах хвостовой рекурсии;

Может ли для любого фиксированного образца x_{π_0} и произвольной неизвестной строки $_{\#}\mathrm{y}_{\mathrm{str}}$ суперкомпилятор SCP4 преобразовать программу P к программе, кодирующей специализированную версию алгоритм Матиясевича?

Для каждого образца x_{π_0} своя задача на специализацию:

$$\mathfrak{T}(\mathrm{P}, \mathrm{x}_{\pi_0}) \triangleq \mathbf{SCP4}(\,\underline{\mathrm{P}}, \underline{\mathrm{S}(\mathrm{x}_{\pi_0}, {}_{\#}\mathrm{y}_{\mathrm{str}})}\,)$$

Здесь S - входная точка (функция) программы P, знак подчеркивания обозначает кодировку.

Для каждого образца x_{π_0} своя задача на специализацию:

$$\mathfrak{T}(\mathrm{P}, \mathrm{x}_{\pi_0}) \triangleq \mathbf{SCP4}(\underline{\mathrm{P}}, \underline{\mathrm{S}(\mathrm{x}_{\pi_0}, \#\mathrm{y}_{\mathrm{str}})})$$

Здесь S - входная точка (функция) программы P, знак подчеркивания обозначает кодировку.

Задача, равномерная по множеству образцов $\mathcal{A}^*$.

$$\forall \pi \in \mathcal{A}^* \,.\, \mathfrak{T}(\mathrm{P}, \pi)$$

Где $\mathcal{A}$ - алфавит символов.

Вариант Рефала для публикаций: s_{name} – s-переменные, p,q,x,y,z – e-переменные, ϵ – пустое выражение.

```
$ENTRY S { /* Search – предикат вхождения. */
  sa p, sa y = L(sa p, sa y, sa p, y);
  sa p, sb y = S(sa p, y);
     p,  ε   = F; }

* Переменные q, z используются для отката по строке y.
L { /* Look for – поиск образца p в строке y. */
  sa p, sa y, q, z = L(p, y, q, z);
  sa p, sb y, q, z = S(q, z);      /* откат по строке */
  sa p,  ε  , q, z = S(q, z);      /*                 */
   ε  ,  y  , q, z = T; }
```

Вариант Рефала для публикаций: s_{name} – s-переменные, p,q,x,y,z – e-переменные, ϵ – пустое выражение.

```
$ENTRY S { /* Search – предикат вхождения. */
  sa p, sa y = L(sa p, sa y, sa p, y);
  sa p, sb y = S(sa p, y);
     p,  ε   = F; }

* Переменные q, z используются для отката по строке y.
L { /* Look for – поиск образца p в строке y. */
  sa p, sa y, q, z = L(p, y, q, z);
  sa p, sb y, q, z = S(q, z);      /* откат по строке */
  sa p,  ε  , q, z = S(q, z);      /*                 */
   ε  ,  y  , q, z = T; }
```

Вариант Рефала для публикаций: s_{name} – s-переменные, p,q,x,y,z – e-переменные, ϵ – пустое выражение.

```
$ENTRY S { /* Search – предикат вхождения. */
  sa p, sa y = L(sa p, sa y, sa p, y);
  sa p, sb y = S(sa p, y);
     p,  ε   = F; }

* Переменные q, z используются для отката по строке y.
L { /* Look for – поиск образца p в строке y. */
  sa p, sa y, q, z = L(p, y, q, z);
  sa p, sb y, q, z = S(q, z);      /* откат по строке */
  sa p,  ε  , q, z = S(q, z);      /*                 */
   ε  ,  y  , q, z = T; }
```

Вариант Рефала для публикаций: s_{name} – s-переменные, p,q,x,y,z – e-переменные, ϵ – пустое выражение.

```
$ENTRY S { /* Search – предикат вхождения. */
  sa p, sa y = L(sa p, sa y, sa p, y);
  sa p, sb y = S(sa p, y);
     p,  ϵ   = F; }

* Переменные q, z используются для отката по строке y.
L { /* Look for – поиск образца p в строке y. */
  sa p, sa y, q, z = L(p, y, q, z);
  sa p, sb y, q, z = S(q, z);      /* откат по строке */
  sa p,  ϵ  , q, z = S(q, z);      /*                 */
   ϵ  ,  y  , q, z = T; }
```

Для каждого образца x_{π_0} своя задача на специализацию:

$$\mathfrak{T}(P, x_{\pi_0}) \triangleq \mathbf{SCP4}(\underline{P}, \underline{S(x_{\pi_0}, {}_{\#}y_{str})})$$

Хвостовая рекурсия.

Сложность модели P в худшем случае: $\mathcal{O}(|{}_{\#}x_{\pi}| \times |{}_{\#}y_{str}|)$.

```
$ENTRY S { /* Search – предикат вхождения. */
  s_a p, s_a y = L(s_a p, s_a y, s_a p, y);
  s_a p, s_b y = S(s_a p, y);
     p,  ϵ     = F; }
```

*** Переменные q, z используются для отката по строке y.**

```
L { /* Look for – поиск образца p в строке y. */
  s_a p, s_a y, q, z = L(p, y, q, z);
  s_a p, s_b y, q, z = S(q, z);      /* откат по строке */
  s_a p,  ϵ   , q, z = S(q, z);      /*                 */
   ϵ  ,  y    , q, z = T; }
```

История:

– доклад Ю.В. Матиясевича 15-ого мая 1969 г. в Ленинграде на семинаре по конструктивной математике;

– публикация на русском языке в 1971 г. (Ю.В. Матиясевич);

– публикация на английском языке в 1973 (Ю.В. Матиясевич), перевод;

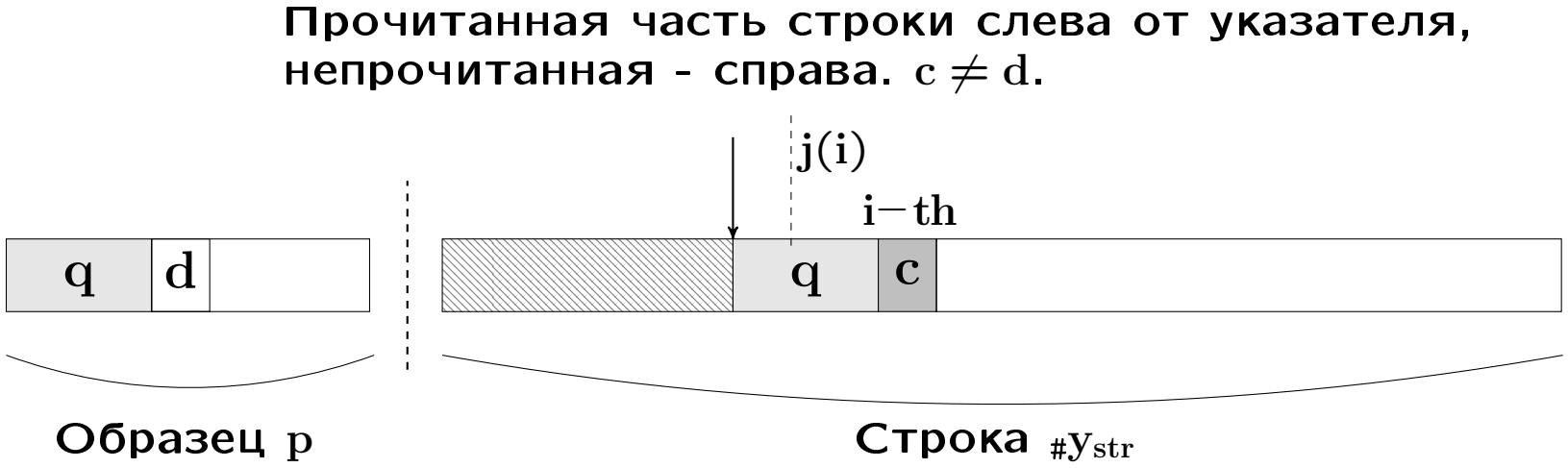
– препринт, 1970 г. (J. H. Morris и V. R. Pratt);

– статья, 1977 г. (D. Knuth, J. Morris и V. Pratt).

Алгоритм M-KMP - I

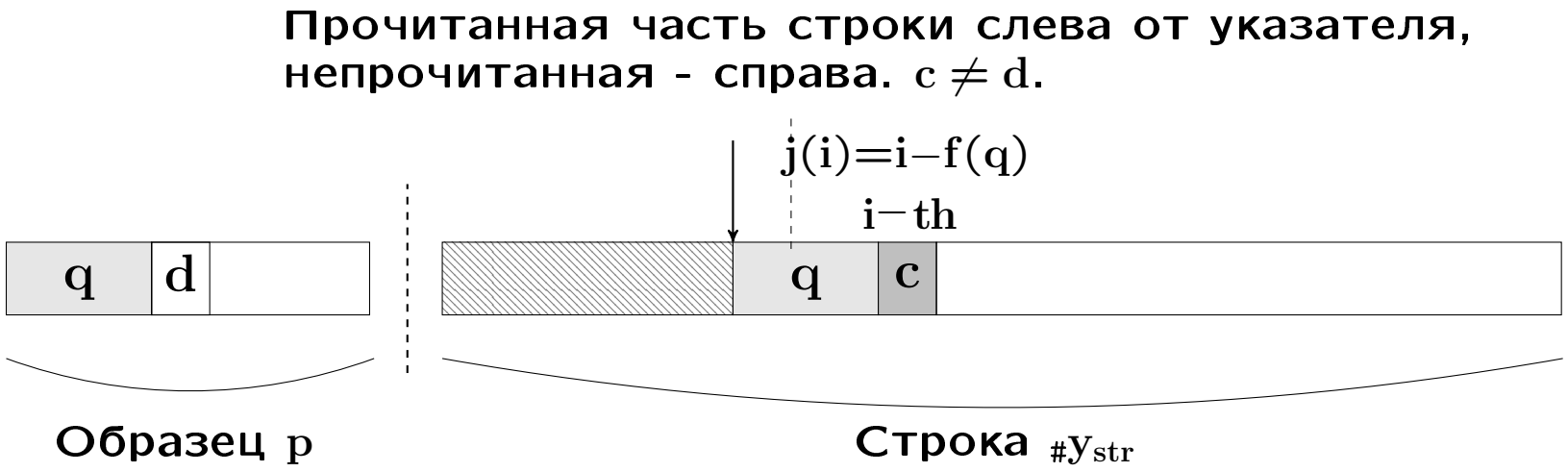
– q - прочитанное начало образца p, i - индекс наблюдаемого вхождения символа $\mathrm{c} \neq \mathrm{d}$.

– M-KMP за «время» $\mathcal{O}(|\mathrm{p}|)$ строит функцию $\mathrm{f(q)}$ такую, что этот этот индекс можно сдвинуть к индексу $\mathrm{j(i)} = \mathrm{i} - \mathrm{f(q)}$, перед которым p появиться не может.



Алгоритм M-KMP - II

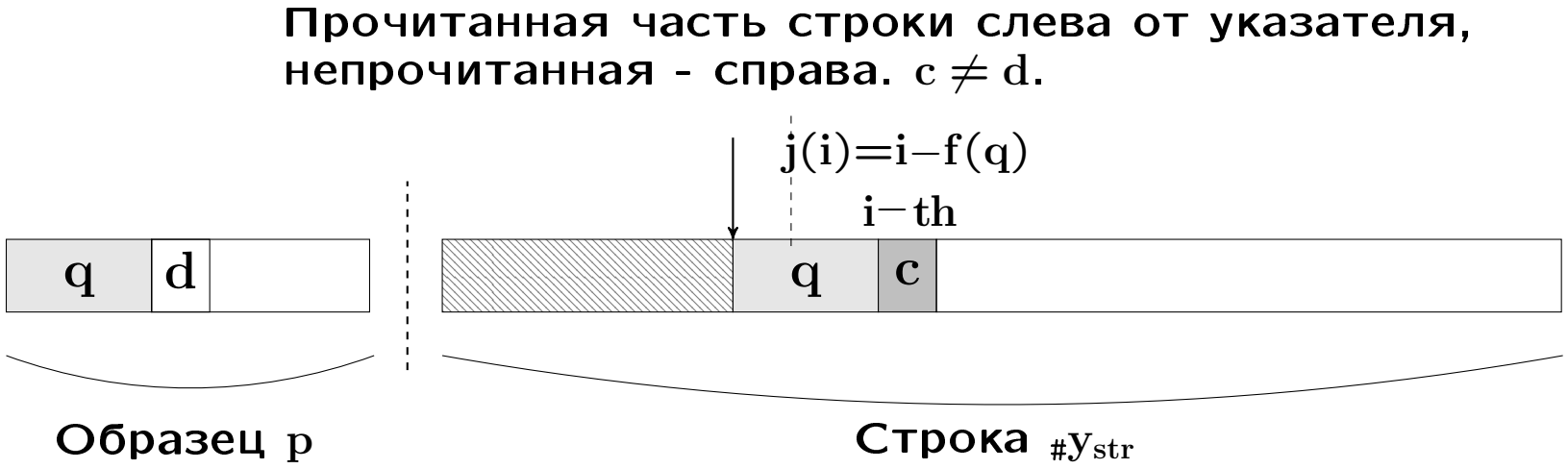
- Затем начинается непосредственный поиск, использующий функцию $\mathrm{f(q)}$.
- Сложность в худшем случае алгоритма M-KMP есть $\mathcal{O}(|\mathrm{p}| + |_{\#}\mathrm{y_{str}}|)$.



Алгоритм M-KMP - III

– **Затем начинается непосредственный поиск, использующий функцию** $\mathrm{f(q)}$.

– **Пусть** $\mathrm{l(q)}$ **есть начало и конец слова** q, **тогда** $\mathrm{f}(\epsilon) = 0$; **и если** $\mathrm{q} \neq \epsilon$: $\mathrm{f(q)} = \max\left\{|\mathrm{l(q)}| \;\middle|\; \mathrm{l(q)} \neq \mathrm{q}\right\}$, **где** ϵ – **пустое слово.**



Алгоритм M-KMP - IV

– **Затем начинается непосредственный поиск, использующий функцию** $\mathrm{f}(\mathrm{q})$.

– **Пусть** $\mathrm{l}(\mathrm{q})$ **есть начало и конец слова** q, **тогда** $\mathrm{f}(\epsilon) = 0$; **и если** $\mathrm{q} \neq \epsilon$: $\mathrm{f}(\mathrm{q}) = \max\left\{|\mathrm{l}(\mathrm{q})| \;\middle|\; \mathrm{l}(\mathrm{q}) \neq \mathrm{q}\right\}$, **где** ϵ – **пустое слово.**

Пример

– 'aab': $(\mathrm{f}(\epsilon) = 0, \mathrm{j}(\mathrm{i}) = \mathrm{i})$, $(\mathrm{f}(\text{'a'}) = 0, \mathrm{j}(\mathrm{i}) = \mathrm{i})$, $(\mathrm{f}(\text{'aa'}) = 1, \mathrm{j}(\mathrm{i}) = \mathrm{i} - 1)$;

– 'ababa': $(\mathrm{f}(\epsilon) = 0, \mathrm{j}(\mathrm{i}) = \mathrm{i})$, $(\mathrm{f}(\text{'a'}) = 0, \mathrm{j}(\mathrm{i}) = \mathrm{i})$, $(\mathrm{f}(\text{'ab'}) = 0, \mathrm{j}(\mathrm{i}) = \mathrm{i})$, $(\mathrm{f}(\text{'aba'}) = 1, \mathrm{j}(\mathrm{i}) = \mathrm{i} - 1)$, $(\mathrm{f}(\text{'abab'}) = 2, \mathrm{j}(\mathrm{i}) = \mathrm{i} - 2)$.

Специализация программных моделей наивного поиска образца в строке

Публикации о преобразованиях программных моделей наивного поиска при нескольких фиксированных входных образцах к оптимизированным версиям алгоритма M-KMP:

- generalized partial computation (Futamura, Nogi, 1988)
  - результаты существенно опираются на использовании «отрицательной информации» о конфигурациях программных моделей;
- partial evaluation (Consel, Danvy, 1989)
  - для получения желаемых результатов авторы существенно отступили от наивности модели поиска;
- среди многих других авторов
  - суперкомпиляция, partial deduction, extended partial deduction, недетерминированные программные модели, и др.;

Преобразования программных моделей наивного поиска

Ручные эксперименты преобразований моделей наивного поиска:

- вычисление оптимизированной версии наивного в общем положении поиска образца в строке, где образец неизвестный – произвольный – параметризованный;
  результат вычисления – алгоритм M-KMP в общем положении (R.S. Bird, J. Gibbons, G. Jones, 1989)
  - метод преобразования основан на алгебраической технике и использует не автоматизированные трюки, включая отношения высшего порядка, понижающие сложность вычисления в худшем случае преобразуемой модели;
- в терминах constraint logic (D. Smith, 1991)
  - построение нескольких специализированных версий алгоритма M-KMP, действующих на некоторых множествах конечных деревьев;

Две задачи

$$\forall \pi_0 \in \mathcal{A}^* \,.\, \mathbf{SCP4}(\,\underline{\mathrm{P}}, \underline{\mathrm{S}(\pi_0, {}_{\#}\mathrm{y}_{\mathrm{str}})}\,)$$

Суперкомпилятор SCP4 строит специализированную версию алгоритма Матиясевича.

vs.

Ручной просчёт (R.S. Bird и др.) оптимизированной версии наивного в общем положении поиска образца в строке, основанный на алгебраической технике и использующий не автоматизированные трюки, включая отношения высшего порядка, понижающие сложность вычисления в худшем случае преобразуемой модели. Результат – алгоритм Матиясевича в общем положении.

$$\mathbf{Opt}(\,\underline{\mathrm{P}}, \underline{\mathrm{S}({}_{\#}\mathrm{x}_{\pi}, {}_{\#}\mathrm{y}_{\mathrm{str}})}\,)$$

Где ${}_{\#}\mathrm{x}_{\pi}$ **и** ${}_{\#}\mathrm{y}_{\mathrm{str}}$ **суть параметры, значения которых пробегают** $\mathcal{A}^*$.

Язык ограничений на множество значений s-параметров.

Определение.

constraint ::= R(t,. . . ,t)

R(t,. . . ,t) ::= | Q(t,t) $\wedge$ R(t,. . . ,t)

Q(t,t) ::= (t $\neq$ t)

t ::= $'\nu'$ | s-parameter

где $\nu \in \mathcal{A}$, ($t_1^0 \neq t_1^0$) интерпретируются как False,

а и ($'\nu_1^0' \neq '\nu_2^0'$), если ν_1^0 не равно ν_2^0, интерпретируются как True.

Пример:

($'a' \neq 'b'$) = True,

($'a' \neq 'a'$) = ($_{\#}s_a \neq {}_{\#}s_a$) = False.

Параметризованные конфигурации.

Определение.
Параметризованная программная конфигурация есть пара вида

$$\langle \mathbf{pexpr}, \mathrm{R}(\mathrm{p_1}, \ldots, \mathrm{p_n}) \rangle$$

, где $\mathrm{p_i}$ суть s-параметры из параметризованного выражения pexpr.

Здесь R есть ограничение (предикат) описывающий «отрицательную информацию», ограничивающую области определения параметров $\mathrm{p_1}, \ldots, \mathrm{p_n}$.

Наилучшая свертка пути в дереве развертки программы.

Пусть даны программа P и её начальная параметризованная конфигурация $\mathrm{F(pexpr)}$.

Деревом полной развёртки пары $\langle \mathrm{P}, \mathrm{F(pexpr)} \rangle$ назовём бесконечную параллельную развёртку этой пары. Обозначим его $\hat{\mathrm{P}}_{\mathrm{F(pexpr)}}$. Индекс будем опускать, когда он ясен из контекста.

Определение. Пусть даны $\hat{\mathrm{P}}$ и параметризованные конфигурации $[\mathrm{C_1}] \triangleq \langle \mathrm{pexpr_1}, \mathrm{R_1(p_1, \ldots, p_n)} \rangle$, $[\mathrm{C_2}] \triangleq \langle \mathrm{pexpr_2}, \mathrm{R_2(q_1, \ldots, q_k)} \rangle$, помечающие узлы в $\hat{\mathrm{P}}$. Скажем, что $[\mathrm{C_1}]$ накрывает $[\mathrm{C_2}]$, если $\exists$ переименовка σ параметров из $[\mathrm{C_1}]$ такая, что $\sigma(\mathrm{pexpr_1}) = \mathrm{pexpr_2}$ и предикат $\mathrm{R_2(q_1, \ldots, q_k)} \Rightarrow \mathrm{R_1}(\sigma(\mathrm{p_1}), \ldots, \sigma(\mathrm{p_n}))$ тождественно истинен.

Отрезки путей в дереве развертки программы, накрывающие поддеревья.

Пусть даны дерево полной развёртки $\hat{\mathrm{P}}$, путь t, начинающийся в корне $\hat{\mathrm{P}}$, и параметризованные конфигурации $[\mathrm{C}_1]$, $[\mathrm{C}_2]$ на t.

Определение. Скажем, что отрезок Δ_1 пути t накрывает $[\mathrm{C}_1]$, если на Δ_1 $\exists$ конфигурация $[\mathrm{C}_0]$, которая накрывает $[\mathrm{C}_2]$.

Определение. Пусть T есть поддерево дерева $\hat{\mathrm{P}}$ и $[\mathrm{C}_2]$ – корень T. Скажем, что отрезок Δ_2 пути t, состоящий из предков $[\mathrm{C}_2]$, накрывает поддерево T, если на любом бесконечном пути, начинающийся в $[\mathrm{C}_2]$ $\exists$ конфигурация $[\mathrm{K}_\mathrm{i}]$, накрытая отрезком Δ_2.

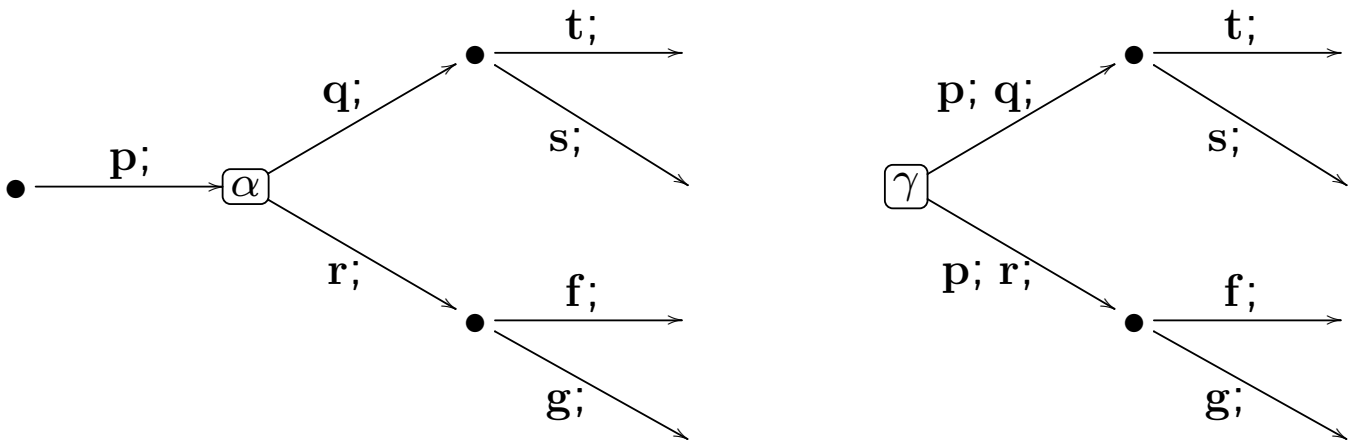
Опорные и транзитивные узлы.

Узел в дереве называется транзитивным, если его выходная степень равна 1. Транзитивные узлы удаляются из дерева полной развёртки программы.

Первый узел выходной степени больше 1 на пути из корня дерева, порожденного одношаговой развёрткой конфигурации, назовём опорным.

Аналогично называются конфигурации, помечающие такие узлы.

Два дерева одношаговой развёртки.



α и γ опорные узлы, первый узел $\bullet$ первого дерева является транзитивным узлом.

Хвостовая рекурсия.

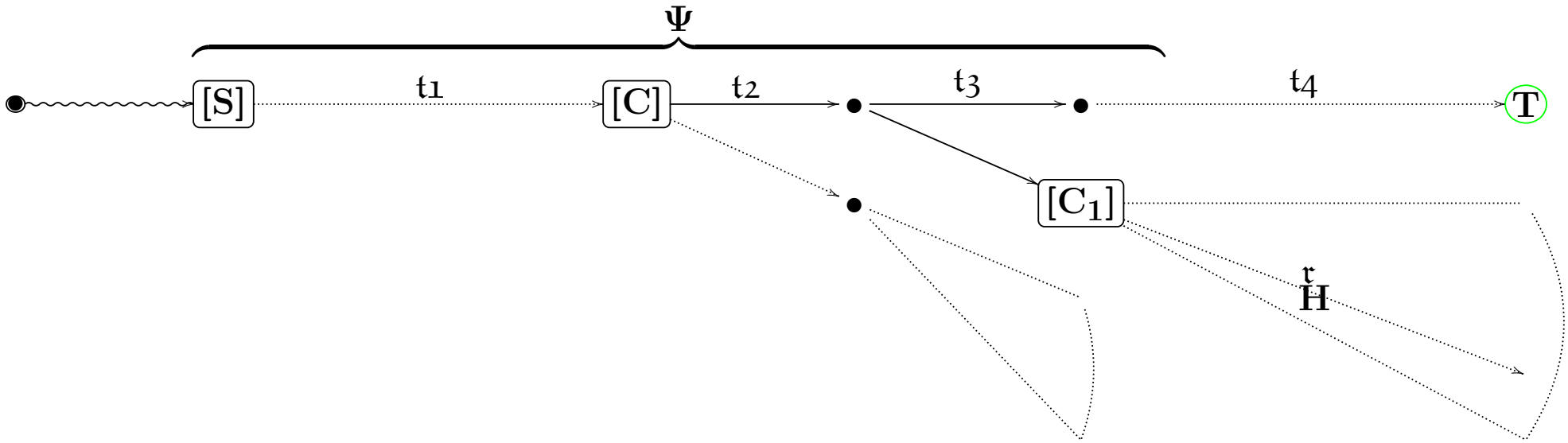
Сложность модели P в худшем случае: $\mathcal{O}(|{}_{\#}\mathrm{x}_{\pi}| \times |{}_{\#}\mathrm{y}_{\mathrm{str}}|)$.

```
$ENTRY S { /* Search – предикат вхождения. */
  s_a p, s_a y = L(s_a p, s_a y, s_a p, y);
  s_a p, s_b y = S(s_a p, y);
     p,  ε     = F; }
```

*** Переменные q, z используются для отката по строке y.**

```
L { /* Look for – поиск образца p в строке y. */
  s_a p, s_a y, q, z = L(p, y, q, z);
  s_a p, s_b y, q, z = S(q, z);      /* откат по строке */
  s_a p,  ε   , q, z = S(q, z);      /*                 */
   ε  ,  y    , q, z = T; }
```

Теорема о накрытии. **Пусть дано слово $\pi \in \mathcal{A}^+$. Пусть t есть первый, кратчайший, путь из корня дерева $\hat{\mathrm{P}}_{\mathrm{S}(\pi,\#\mathrm{y}_{\mathrm{str}})}$ к листу, помеченному конфигурацией T, Ψ – непустое начало пути t, $[\mathrm{C}]$ – последняя опорная конфигурация на Ψ.**
Тогда для любого поддерева H дерева $\hat{\mathrm{P}}_{\mathrm{S}(\pi,\#\mathrm{y}_{\mathrm{str}})}$, с корнем в листе дерева одношаговой развёртки конфигурации $[\mathrm{C}]$ и не принадлежащего пути t, Ψ накрывает H.



$\mathrm{t} ::= \mathrm{t}_1, \mathrm{t}_2, \mathrm{t}_3, \mathrm{t}_4$

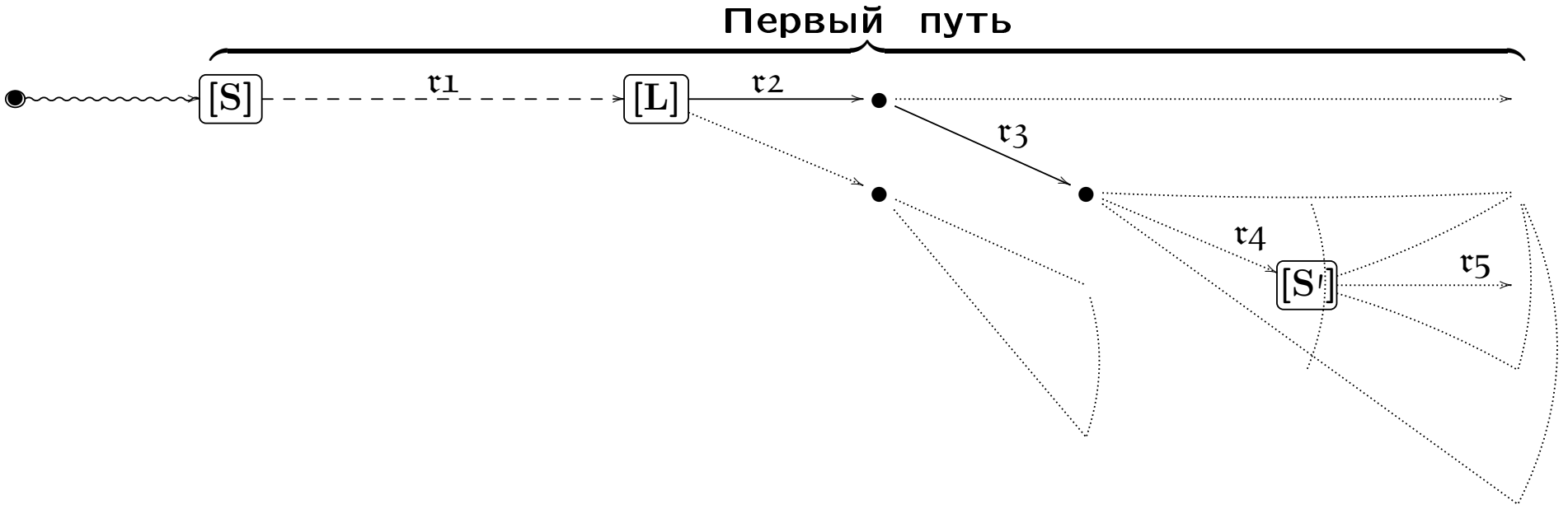
Лемма 1. $\forall\, \pi \in \mathcal{A}^{+}$ **первый путь из корня дерева** $\hat{\mathrm{P}}_{\mathrm{S}(\pi,\#\mathrm{y_{str}})}$ **заканчивается пассивной конфигурацией** T **и все опорные конфигурации на этом пути, порождаемые суперкомпилятором SCP4, образуют конечную последовательность**

$$[\mathrm{S}], [\mathrm{L_1}], \ldots, [\mathrm{L_{(n-1)}}],$$

где $\mathrm{n} = |\pi|$, $[\mathrm{S}]$ **– начальная конфигурация,**
$[\mathrm{L_i}] \triangleq \langle \mathrm{L}(\pi_{\mathrm{i}}, \#\mathrm{y_{str}}, \pi, \omega_{\mathrm{i}}\ \#\mathrm{y_{str}}), \epsilon \rangle$, **где** π_i **есть** i**-й непустой суффикс образца** π **(**$\pi_0 = \pi$**, для** $0 < i < |\pi|$, $|\pi_i| = |\pi| - i$**),** ω_i **есть** i**-й префикс слова** π_1 **такой, что** $|\omega_i| = i - 1$.
Транзитивная конфигурация $[\mathrm{L_n}]$ **следует за этой последовательностью активных конфигурация вдоль этого пути;**
и $\forall\, \mathrm{i}, \mathrm{j} \in \mathbb{N}.\ 0 \leq \mathrm{i} < \mathrm{j} < \mathrm{n}$ **верно неравенство** $|\pi_i| > |\pi_j|$.

Нижеследующие две леммы верны для дерева полной развёртки, представленного в префиксном форме.

Лемма 2. $\forall \pi \in \mathcal{A}^+$ **и любого бесконечного пути** $\mathfrak{r}$ **из корня дерева** $\hat{\mathrm{P}}_{\mathrm{S}(\pi,\#\mathrm{y_{str}})}$, **содержащего хотя бы одну конфиг. с вызовом функции** L, $\exists$ **конфиг.** $[\mathrm{S}']$ **вида** $\langle \mathrm{S}(\pi, \#\mathrm{s_a}\, \#\mathrm{y_{str}}), \mathrm{R}(\#\mathrm{s_a})\rangle$ **такая, что корень** $[\mathrm{S}]$ **дерева** $\hat{\mathrm{P}}_{\mathrm{S}(\pi,\#\mathrm{y_{str}})}$ **является единственным предком конфиг.** $[\mathrm{S}']$ **с вызовом функции** S. **Отрезок** $[\![[\mathrm{S}];\, [\mathrm{S}']]\!]$ **не содержит других опорных конфиг. кроме:** $[\mathrm{S}]$,$[\mathrm{S}']$ **и нескольких опорных конфиг., принадлежащих первому пути дерева** $\hat{\mathrm{P}}_{\mathrm{S}(\pi,\#\mathrm{y_{str}})}$.

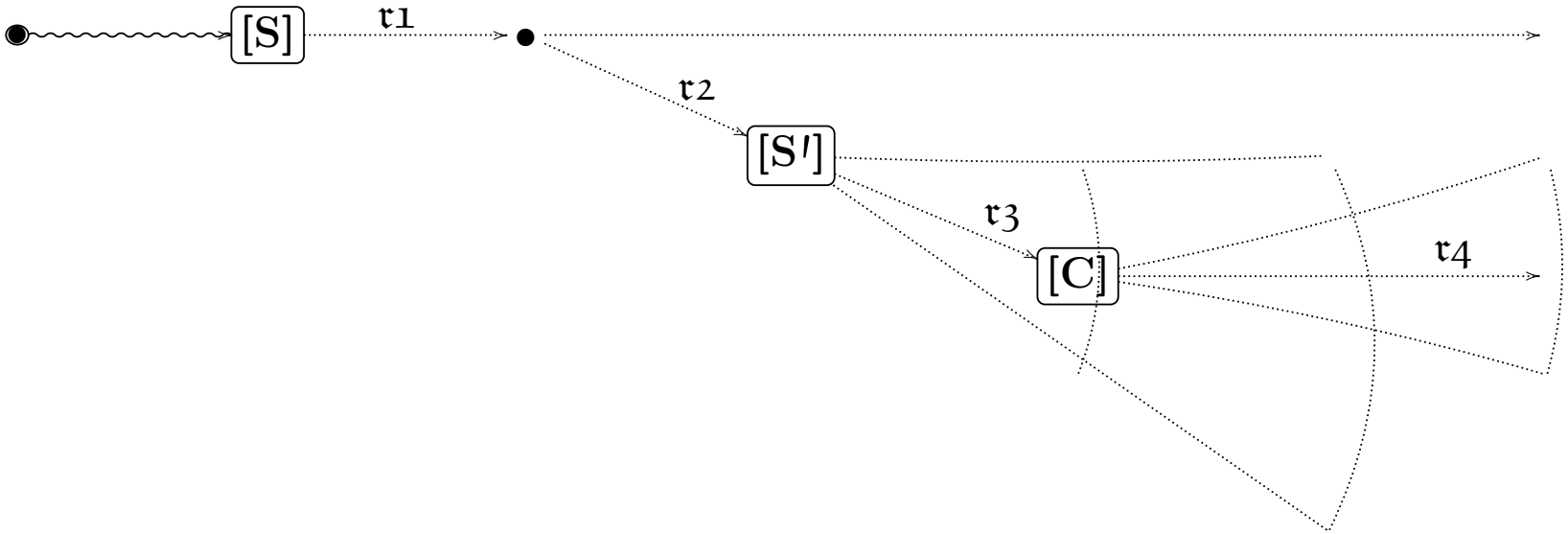


$\mathfrak{r} ::= \mathfrak{r}_1, \mathfrak{r}_2, \mathfrak{r}_3, \mathfrak{r}_4, \mathfrak{r}_5$

Лемма 3. $\forall\,\pi \in \mathcal{A}^{+}$ **и любого бесконечного пути** $\mathfrak{r}$ **из корня дерева** $\hat{\mathrm{P}}_{\mathrm{S}(\pi,\#\mathrm{y_{str}})}$, **содержащего хотя бы одну конфигурацию** $\langle \mathrm{S}(\pi, \#\mathrm{s_a}\,\#\mathrm{y_{str}}), \mathrm{R}(\#\mathrm{s_a})\rangle$, **пусть** $[\mathrm{S}']$ **есть первое вхождение такой конфиг. в** $\mathfrak{r}$. **Тогда первая опорная конфиг. после** $[\mathrm{S}']$ **на продолжении пути** $\mathfrak{r}$ **есть конфиг.** $[\mathrm{C}]$ **одного и двух видов:**

(1) **если предикат** $\mathrm{R}(\xi_1)$ **выполним, тогда** $[\mathrm{C}]$ **имеет вид** $\mathrm{L}(\pi_1, \#\mathrm{y_{str}}, \pi, \#\mathrm{y_{str}})$ **и это вхожд. накрыв. одним из её предков;**

(2) **если предикат** $\mathrm{R}(\#\mathrm{s_a}) \wedge (\#\mathrm{s_a} \neq \xi_1)$ **выполним, тогда** $[\mathrm{C}]$ **есть потомок конфиг.** $[\mathrm{S}']$ **вида** $\mathrm{S}(\pi, \#\mathrm{y_{str}})$.



$\mathfrak{r} ::= \mathfrak{r}_1, \mathfrak{r}_2, \mathfrak{r}_3, \mathfrak{r}_4$

Задача: SCP4(P, S('abcabcaca', #y$_{str}$))

Остаточная программа. Функция F7.

```
$ENTRY F7 { /* Входная точка. */
  'a'  y = F13_a(y);
  s_b  y = F7(y);
   ε   = F; }
```

Остаточная программа. Функция $F13_a$.

$$
\begin{aligned}
&F13_a\{\\
&\quad 'bcab'\ y \ = F33_{abcab}(y);\\
&\quad 'bcaa'\ y \ = F13_a(y);\\
&\quad 'bca'\ s_b\ y \ = F7(y);\\
&\quad 'bca' \ = F;\\
&\quad 'bc'\ s_b\ y \ = F7(y);\\
&\quad 'bc' \ = F;\\
&\quad 'ba'\ y \ = F13_a(y);\\
&\quad 'b'\ s_b\ y \ = F7(y);\\
&\quad 'b' \ = F;\\
&\quad 'a'\ y \ = F13_a(y);\\
&\quad s_b\ y \ = F7(y);\\
&\quad\ \ \epsilon \ = F; \}
\end{aligned}
$$

Остаточная программа. Функция $\mathsf{F33}_{\mathrm{abcab}}$.

$$
\begin{aligned}
&\mathsf{F33}_{\mathrm{abcab}}\{\\
&\quad \text{'caca'}\ \mathrm{y} \ = \mathrm{T};\\
&\quad \text{'cac'}\ \mathrm{s_{a1}}\ \mathrm{y} \ = \mathsf{F7}(\mathrm{y});\\
&\quad \text{'cac'} \ = \mathrm{F};\\
&\quad \text{'cab'}\ \mathrm{y} \ = \mathsf{F33}_{\mathrm{abcab}}(\mathrm{y});\\
&\quad \text{'caa'}\ \mathrm{y} \ = \mathsf{F13}_{\mathrm{a}}(\mathrm{y});\\
&\quad \text{'ca'}\ \mathrm{s_{c1}}\ \mathrm{y} \ = \mathsf{F7}(\mathrm{y});\\
&\quad \text{'ca'} \ = \mathrm{F};\\
&\quad \text{'c'}\ \mathrm{s_a}\ \mathrm{y} \ = \mathsf{F7}(\mathrm{y});\\
&\quad \text{'c'} \ = \mathrm{F};\\
&\quad \text{'a'}\ \mathrm{y} \ = \mathsf{F13}_{\mathrm{a}}(\mathrm{y});\\
&\quad \mathrm{s_c}\ \mathrm{y} \ = \mathsf{F7}(\mathrm{y});\\
&\qquad \epsilon \ = \mathrm{F};\ \}
\end{aligned}
$$

Префиксное дерево отождествления остаточной функции.