БУ ВО Ханты-Мансийского автономного округа – Югры
«Сургутский государственный университет»

Теория языков программирования и методы трансляции

Часть 3

- Грамматики Хомского
- Задача трансляции
- Структура транслятора

Гришмановский Павел Валерьевич
доцент кафедры автоматики и компьютерных систем, к.т.н., доцент

Сургут, 2019

Грамматики в форме Бэкуса–Наура

Форма Бэкуса–Наура и грамматики Хомского

Грамматика $G = (\mathbf{V}^T, \mathbf{V}^N, \mathbf{P}, S)$, где

- $\mathbf{V}^T$ – множество терминальных символов, соответствующее алфавиту порождаемого языка
- $\mathbf{V}^N$ – множество нетерминальных символов (металингвистических переменных), используемых в записи грамматики

  $$\mathbf{V}^T \cap \mathbf{V}^N = \varnothing,\ \mathbf{V}^T \cup \mathbf{V}^N = \mathbf{V}$$

- **P** – множество правил грамматики

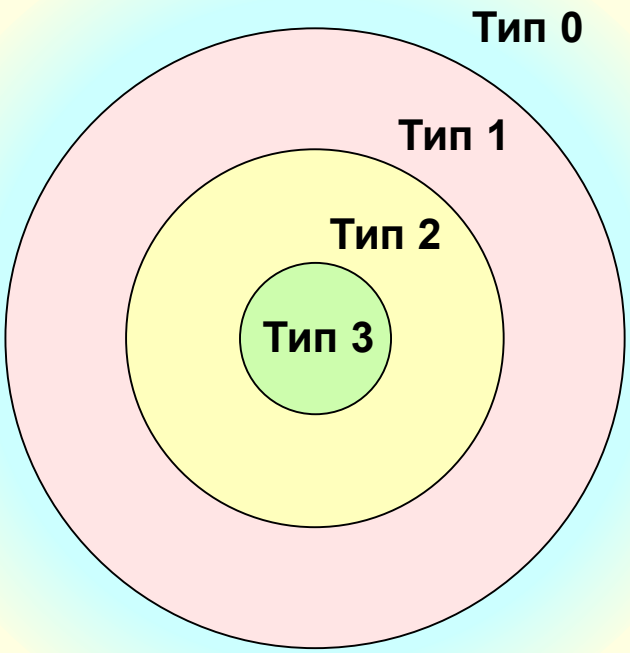
  $$\mathbf{P} : \alpha \to \beta,\ \alpha \in \mathbf{V}^+,\ \beta \in \mathbf{V}^*$$

- S – начальный (целевой) символ грамматики, $S \in \mathbf{V}^N$

- Наличие **рекурсивных правил** позволяет определять языки, порождающие бесконечное множество цепочек и цепочки сколь угодно большой длины при конечных алфавите и множестве правил
- Наличие **нерекурсивных правил** необходимо для порождения конечных цепочек языка
- Структура правил грамматики, порождающей язык, определяется сложностью этого языка

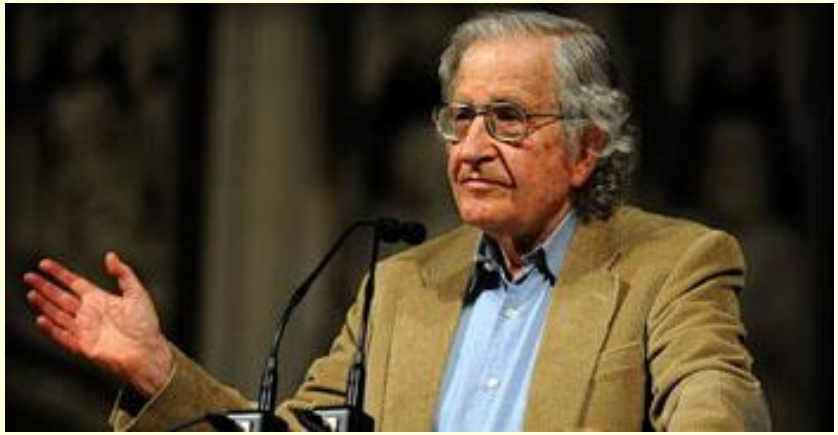
Иерархия грамматик Хомского

Все грамматики



Каждый следующий тип образуется за счет более жестких ограничений на структуру правил грамматики

Ноам Хомский
(Chomsky, Noam Avram)



07.12.1928 г., Филадельфия, США
лингвист, философ, общественный деятель

монография «Syntactic Structures», 1957 г., (рус. перевод «Синтаксические структуры», 1962 г.) – результат исследований в Гарвардском университете в 1951–1955 гг. – положила начало теории генеративизма (порождающих грамматик)

Иерархия грамматик Хомского

Тип	Структура правил	Название	Описание
Тип 0	$\alpha \to \beta$, $\alpha \in \mathbf{V}^+$, $\beta \in \mathbf{V}^*$	Неограниченные, с фразовой структурой	Описывают естественные языки Большинство свойств неразрешимо
Тип 1	$\gamma_1 A \gamma_2 \to \gamma_1 \beta \gamma_2$, $A \in \mathbf{V}^N$, $\beta \in \mathbf{V}^+$, $\gamma_1, \gamma_2 \in \mathbf{V}^*$	Контекстно-зависимые (КЗГ)	Описывают большинство ЯП Классы КЗГ и НУГ эквивалентны Значительная часть свойств неразрешена
	$\alpha \to \beta$, $\alpha, \beta \in \mathbf{V}^+$, $\lvert\alpha\rvert \le \lvert\beta\rvert$	Неукорачивающие (НУГ)	
Тип 2	$A \to \beta$, $A \in \mathbf{V}^N$, $\beta \in \mathbf{V}^*$	Контекстно-свободные (КСГ – УКСГ и НУКСГ)	Описывают отдельные конструкции ЯП, языки командных процессоров Хорошо изучены
Тип 3	$A \to B\beta \mid \beta$, $A, B \in \mathbf{V}^N$, $\beta \in \mathbf{V}^{T*}$	Регулярные леволинейные (ЛЛГ)	Описывают отдельные лексемы ЛЛГ и ПЛГ эквивалентны Хорошо изучены
	$A \to \beta B \mid \beta$, $A, B \in \mathbf{V}^N$, $\beta \in \mathbf{V}^{T*}$	Регулярные праволинейные (ПЛГ)	

Классификация языков

Классификация языков по типам выполняется по самой простой грамматике, его порождающей (с наибольшим номером типа) – если некоторый язык может быть описан грамматикой типа *n* и не может быть описан грамматикой типа *n*+1, то он является языком типа *n* (проверку грамматики целесообразно начинать с *n* = 3)

1) $S \to T \mid +T \mid -T$
   $T \to F \mid TF$
   $F \to 0 \mid 1 \mid \ldots \mid 9$
   Тип 2

2) $S \to NT$
   $N \to + \mid - \mid \lambda$
   $T \to F \mid TF$
   $F \to 0 \mid 1 \mid \ldots \mid 9$
   Тип 2

3) $S \to T \mid +T \mid -T$
   $T \to 0 \mid 1 \mid \ldots \mid 9 \mid$
   $\mid 0T \mid 1T \mid \ldots \mid 9T$
   Тип 3 (праволинейная)

4) $S \to N0 \mid N1 \mid \ldots \mid N9 \mid$
   $\mid S0 \mid S1 \mid \ldots \mid S9$
   $N \to + \mid - \mid \lambda$
   Тип 3 (леволинейная)

Тип 3
Регулярная грамматика

5) $S \to 0A1$
   $0A \to 00A1$
   $A \to \lambda$
   Тип 0

6) $S \to 0A1 \mid 01$
   $0A \to 00A1 \mid 001$
   Тип 1

7) $S \to 0S1 \mid 01$
   Тип 2

Тип 2
Контекстно-свободная грамматика

Вывод в грамматиках Хомского

Вывод – это процесс порождения предложения (подцепочки) языка на основе правил грамматики, определяющей этот язык

В грамматике $G = (\mathbf{V}^T, \mathbf{V}^N, \mathbf{P}, S)$:

$\alpha \Rightarrow \beta$	Цепочка β непосредственно выводима из цепочки α	Существует подходящее правило: $\alpha = \gamma_1 \upsilon \gamma_2,\ \beta = \gamma_1 \omega \gamma_2,\ \exists\ (\upsilon \rightarrow \omega) \in \mathbf{P}$
$\alpha \Rightarrow^* \beta$	Цепочка β выводима из цепочки α	Рекурсивное определение: $(\alpha = \beta) \vee \exists\ (\alpha \Rightarrow^* \delta,\ \delta \Rightarrow \beta)$
$\alpha \Rightarrow^+ \beta$	Цепочка β нетривиально выводима из цепочки α	Существует не менее одной промежуточной цепочки: $\exists\ ((\alpha \Rightarrow^+ \delta \vee \alpha \Rightarrow \delta),\ \delta \Rightarrow \beta)$
$\alpha \Rightarrow^n \beta$	Цепочка β выводима из цепочки α за n шагов	Существует ровно n-1 промежуточная цепочка: $\exists\ (\alpha \Rightarrow \delta_1 \Rightarrow \delta_2 \Rightarrow \ldots \Rightarrow \delta_n = \beta)$
$\alpha \Rightarrow_i \beta$	Цепочка β непосредственно выводима из цепочки α	Использовано правило № i множества **P** грамматики G

Вывод в грамматиках Хомского

Если в грамматике G $\exists\ \alpha \Rightarrow^* \beta$, то

- если $\beta \in \mathbf{V}^{T*}$, то вывод называется **законченным**, а β – **конечной** формой – цепочка β состоит только **из терминальных** символов
- если $\alpha = S$, то цепочка $\beta \in \mathbf{V}^*$ называется **сентенциальной** формой – цепочка выводима **из целевого** (начального) символа грамматики G
- если выполняются оба условия ($S \Rightarrow^* \beta \wedge \beta \in \mathbf{V}^{T*}$), то β является **конечной сентенциальной** формой – является **цепочкой языка**, порождаемого грамматикой G (т.е. программой на этом языке)

$S \rightarrow NT$	**1**
$N \rightarrow + \mid - \mid \lambda$	**2-4**
$T \rightarrow F \mid TF$	**5-6**
$F \rightarrow 0 \mid 1 \mid \ldots \mid 9$	**7-16**

$\underline{S} \Rightarrow_1 N\underline{T} \Rightarrow_6 \underline{N}TF \Rightarrow_4 \underline{T}F \Rightarrow_6 \underline{T}FF \Rightarrow_5 FF\underline{F} \Rightarrow$
$\Rightarrow_{10} FF3 \Rightarrow_9 \underline{F}23 \Rightarrow_8 123$

$\underline{S} \Rightarrow_1 N\underline{T} \Rightarrow_5 \underline{N}F \Rightarrow_3 -\underline{F} \Rightarrow_{14} -7$

$\underline{S} \Rightarrow_1 N\underline{T} \Rightarrow_6 N\underline{T}F \Rightarrow_6 N\underline{T}FF \Rightarrow_5 N\underline{F}FF \Rightarrow$
$\Rightarrow_{12} N5\underline{F}F \Rightarrow_{11} N54\underline{F} \Rightarrow_{10} \underline{N}543 \Rightarrow_2 +543$

$S \rightarrow T \mid +T \mid -T$	**1-3**
$T \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid \ldots \mid 9 \mid$	**4-13**
$\mid 0T \mid 1T \mid 2T \mid 3T \mid \ldots \mid 9T$	**14-23**

$\underline{S} \Rightarrow_1 \underline{T} \Rightarrow_{15} 1\underline{T} \Rightarrow_{16} 12\underline{T} \Rightarrow_7 123$

$\underline{S} \Rightarrow_3 -\underline{T} \Rightarrow_{11} -7$

$\underline{S} \Rightarrow_2 +\underline{T} \Rightarrow_{19} +5\underline{T} \Rightarrow_{18} +54\underline{T} \Rightarrow_7 +543$

Вывод в грамматиках Хомского

Цепочка вывода – это последовательность шагов вывода вида $\alpha = \delta_0 \Rightarrow \delta_1 \Rightarrow \delta_2 \Rightarrow \delta_3 \Rightarrow \ldots \Rightarrow \delta_n = \beta$

- Для $\forall \beta \in \mathbf{V}^* : S \Rightarrow^* \beta$ могут существовать более одной цепочки вывода, отличающиеся порядком применения правил

Левосторонним (**правосторонним**) называется такой вывод, при котором на каждом шаге вывода $\delta_{i-1} \Rightarrow \delta_i$ выбирается такое правило $(\alpha \rightarrow \beta) \in \mathbf{P}$, чтобы α занимала бы самую левую (правую) возможную позицию в цепочке $\delta_{i-1} = \gamma_1 \alpha \gamma_2$, $\gamma_1, \gamma_2 \in \mathbf{V}^*$

- Для грамматик **типа 3** цепочки лево- и правостороннего вывода совпадают – в любой выводимой цепочке присутствует только один нетерминальный символ и $\alpha = A$, $A \in \mathbf{V}^N$
- Для грамматик **типа 2** может быть однозначно построен как лево-, так и правосторонний вывод – на каждом шаге вывода применяется правило к самому левому (правому) нетерминальному символу, т.к. $\alpha = A$, $A \in \mathbf{V}^N$
- Для грамматик **типа 0 и 1** задача построения лево- и правостороннего вывода неоднозначна

Дерево вывода

Дерево вывода – это ориентированный граф, соответствующий цепочке вывода в грамматике $G = (\mathbf{V}^T, \mathbf{V}^N, \mathbf{P}, S)$

- Граф является деревом (без циклов и сросшихся ветвей)
- Листьями являются узлы, маркированные терминальными символами $a \in \mathbf{V}^T$
- Не листьевые узлы (имеющие дочерние узлы) маркированы нетерминальными символами алфавита $A \in \mathbf{V}^N$
- Корневой узел маркирован начальным (целевым) символом S
- Если дочерние для $A \in \mathbf{V}^N$ узлы маркированы $a_1, a_2, a_3, \ldots, a_n \in \mathbf{V}$ (слева направо), то $\exists (A \to a_1 a_2 a_3 \ldots a_n) \in \mathbf{P}$

Для построения дерева вывода достаточно иметь цепочку вывода
$\alpha = \delta_0 \Rightarrow \delta_1 \Rightarrow \delta_2 \Rightarrow \ldots \Rightarrow \delta_n = \beta$

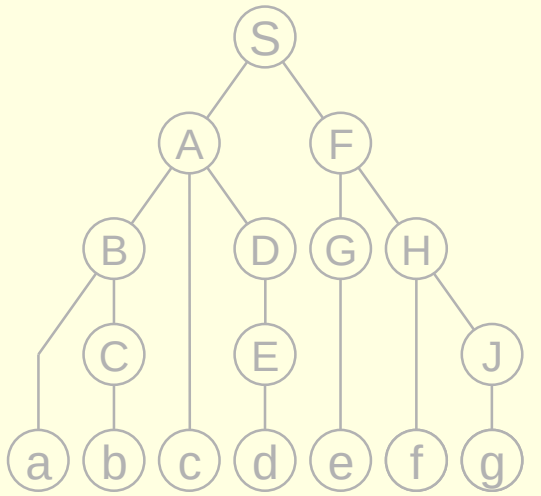
- Для грамматик **типов 3 и 2** для любой цепочки вывода может быть построено дерево вывода
- Для грамматик **типа 1** дерево вывода может быть построено в такой форме, но требует дополнительного отражения левого и правого контекста
- Для грамматик **типа 0** дерево вывода в такой форме построено быть не может

Дерево вывода (пример)

Генерация

Левосторонний вывод,
«сверху вниз»,
прямой порядок

Правило	$\alpha \in V_T^*$, $\beta \in V^*$
	λ S
S → AF	λ AF
A → BcD	λ BcDF
B → aC	a CcDF
C → b	abc DF
D → E	abc EF
E → d	abcd F
F → GH	abcd GH
G → e	abcde H
H → fJ	abcdef J
J → g	abcdefg λ



$\alpha\beta$

Разбор

Правосторонний вывод,
«снизу вверх»
обратный порядок

$\alpha \in V^*$, $\beta \in V_T^*$	Правило
S λ	S → AF
AF λ	F → GH
AGH λ	H → fJ
AGfJ λ	J → g
AG fg	G → e
A efg	A → BcD
BcD efg	D → E
BcE efg	E → d
B cdefg	B → aC
aC cdefg	C → b
λ abcdefg	

Однозначность грамматик

Грамматика однозначна, если для $\forall \beta \in \mathbf{V}^* : S \Rightarrow^* \beta$ деревья вывода, соответствующие всем возможным цепочкам вывода, совпадают

- Практический смысл с точки зрения генерации и разбора имеют лево- и правосторонний выводы, поэтому грамматика **считается** однозначной, если совпадают деревья **лево- и правостороннего вывода**

К неоднозначности контекстно-свободной грамматики приводит наличие в ней правил следующих видов:

$A \to AA$	AAA	или	AAA
$A \to A\alpha A$	$A\alpha A\alpha A$	или	$A\alpha A\alpha A$
$A \to \alpha A \mid A\beta$	$\alpha A\beta$	или	$\alpha A\beta$
$A \to \alpha A \mid \alpha A\beta$	$\alpha\alpha A\beta$	или	$\alpha\alpha A\beta$

Отсутствие подобных правил не означает, что грамматика однозначна

Пример неоднозначной грамматики:

$G = (\{a, b, c, +, -, *, /\}, \{S\}, \mathbf{P}, S)$,

$\mathbf{P} = \{ S \to S+S \mid S-S \mid S*S \mid S/S \mid a \mid b \mid c \}$

Однозначность грамматик (пример 1)

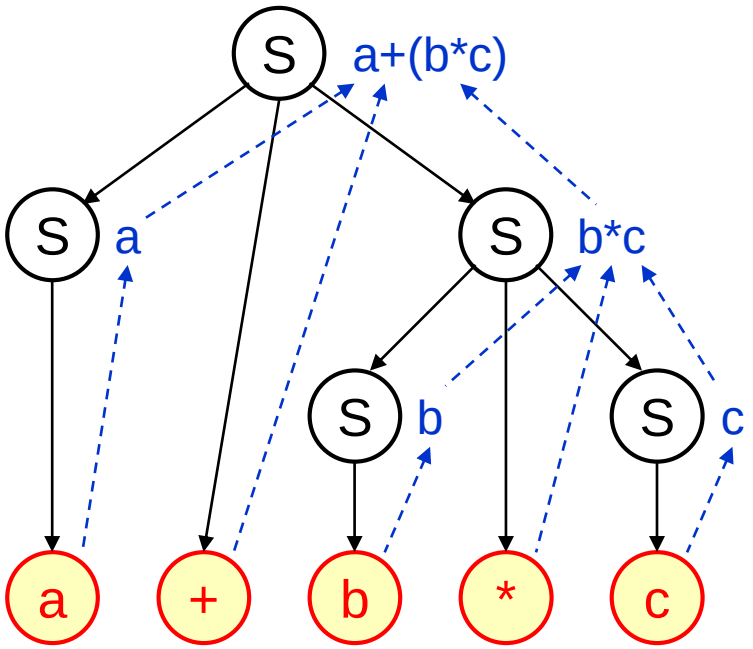
G = ({a, b, c, +, -, *, /}, {S}, **P**, S),

P = { S → S+S | S-S | S*S | S/S | a | b | c }

Левосторонний вывод:

S ⇒ S+S ⇒ a+S ⇒ a+S*S ⇒

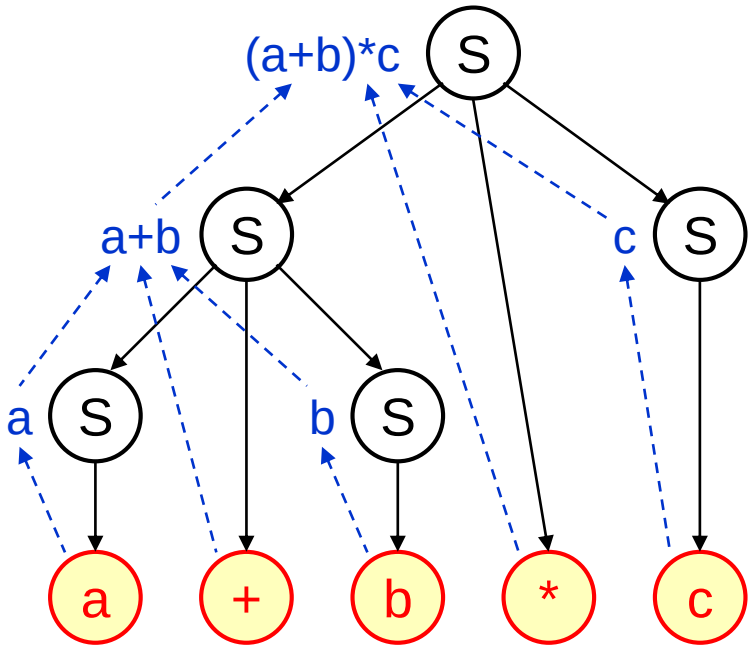
⇒ a+b*S ⇒ a+b*c



Правосторонний вывод:

S ⇒ S*S ⇒ S*c ⇒ S+S*c ⇒

⇒ S+b*c ⇒ a+b*c



Грамматика неоднозначна

Однозначность грамматик (пример 2)

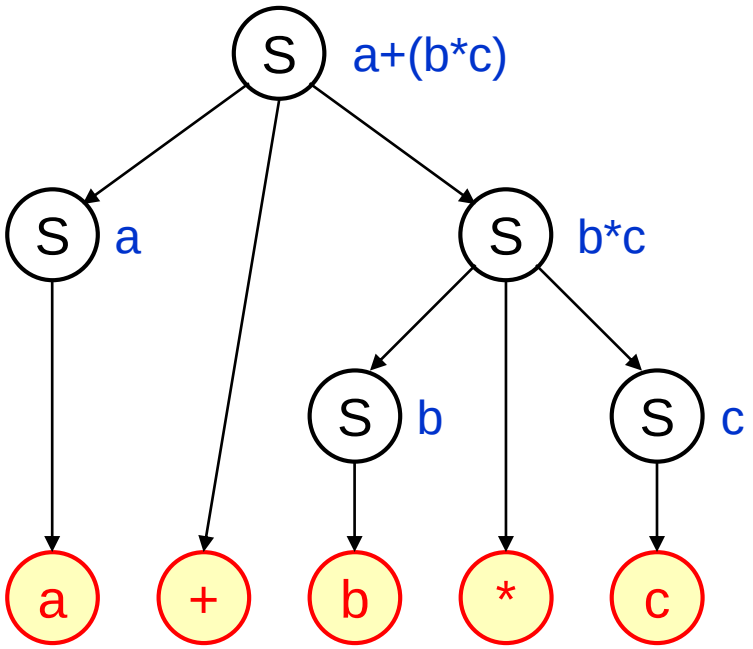
G = ({a, b, c, +, -, *, /}, {S}, **P**, S),

P = { S → S+S | S-S | S*S | S/S | a | b | c }

Левосторонний вывод:

$\underline{S} \Rightarrow \underline{S}+S \Rightarrow a+\underline{S} \Rightarrow a+\underline{S}*S \Rightarrow$

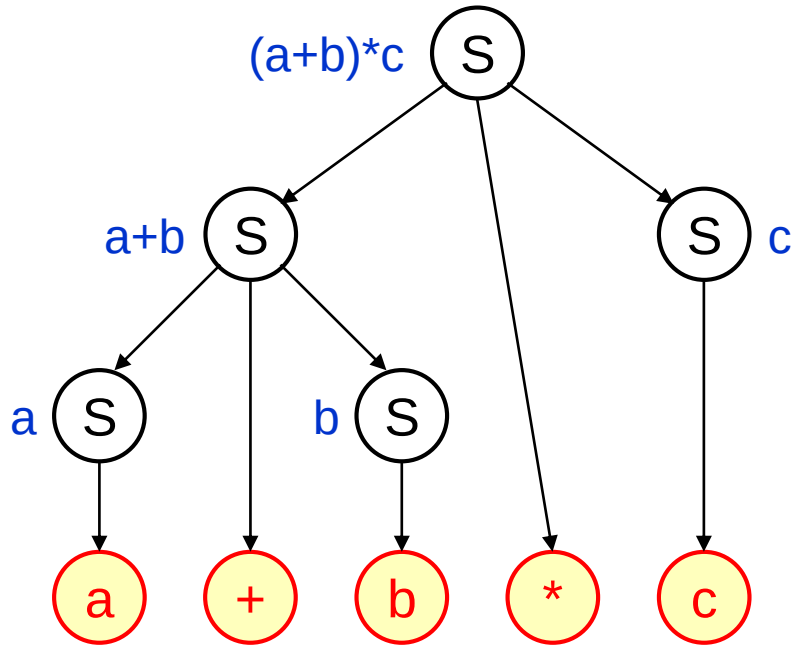
$\Rightarrow a+b*\underline{S} \Rightarrow a+b*c$



Левосторонний (другой) вывод:

$\underline{S} \Rightarrow \underline{S}*S \Rightarrow \underline{S}+S*S \Rightarrow a+\underline{S}*S \Rightarrow$

$\Rightarrow a+b*\underline{S} \Rightarrow a+b*c$



Грамматика неоднозначна

Однозначность грамматик (пример 3)

G = ({a, b, c, +, -, *, /}, {S, T, E}, **P**, S),

P = { S → S+T | S-T | T

T → T*E | T/E | E

E → a | b | c }

Левосторонний вывод:

$\underline{S} \Rightarrow \underline{S}+T \Rightarrow \underline{T}+T \Rightarrow \underline{E}+T \Rightarrow$

$\Rightarrow a+\underline{T} \Rightarrow a+\underline{T}*E \Rightarrow a+\underline{E}*E \Rightarrow$

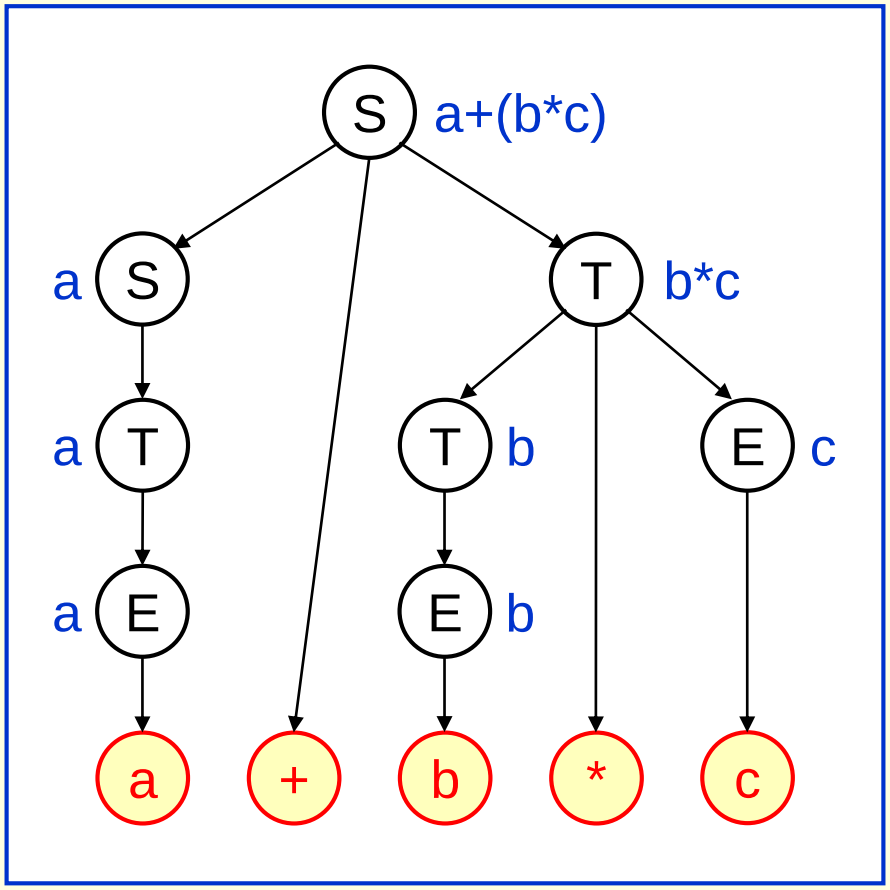
$\Rightarrow a+b*\underline{E} \Rightarrow a+b*c$

Правосторонний вывод:

$\underline{S} \Rightarrow S+\underline{T} \Rightarrow S+T*\underline{E} \Rightarrow S+\underline{T}*c \Rightarrow$

$\Rightarrow S+\underline{E}*c \Rightarrow \underline{S}+b*c \Rightarrow \underline{T}+b*c \Rightarrow$

$\Rightarrow \underline{E}+b*c \Rightarrow a+b*c$



Грамматика однозначна

Однозначность грамматик (пример 4)

G = ({a, b, c, +, -, *, /}, {S, T, E}, **P**, S),

P = { S → S+T | S-T | T

T → T*E | T/E | E

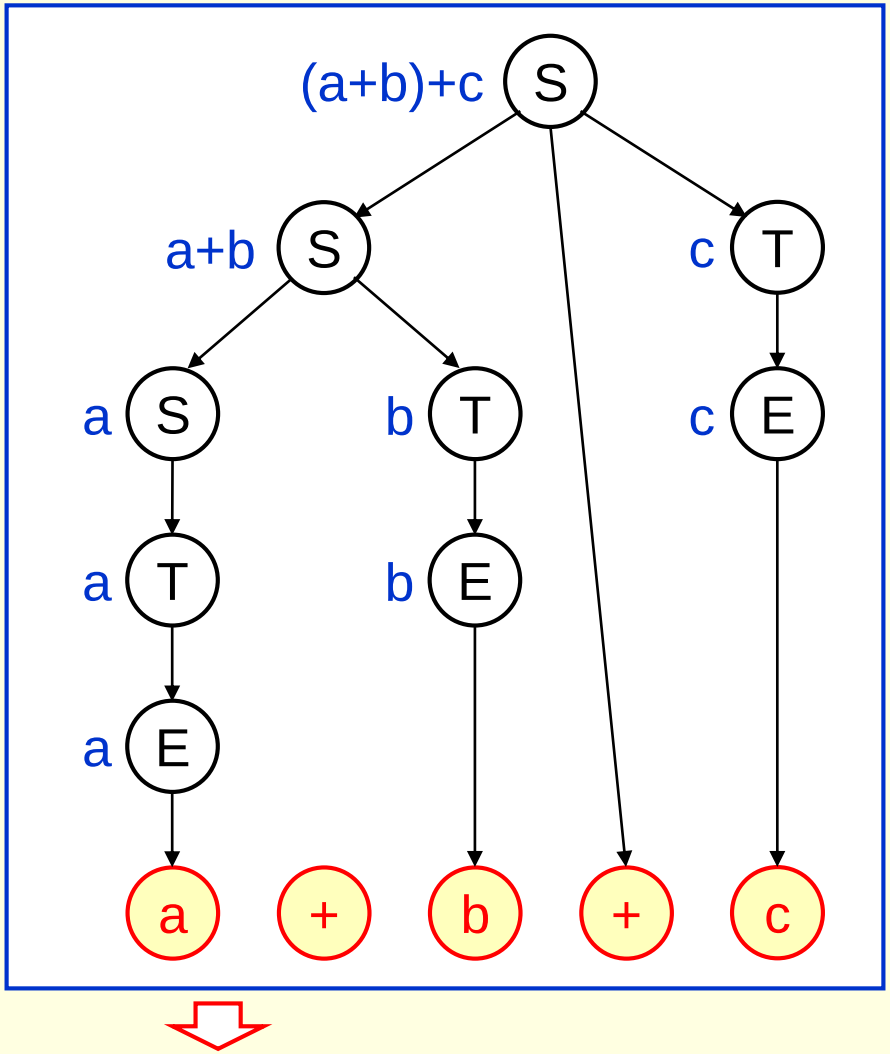
E → a | b | c }

Левосторонний вывод:

$\underline{S} \Rightarrow \underline{S}+T \Rightarrow \underline{S}+T+T \Rightarrow \underline{T}+T+T \Rightarrow$
$\Rightarrow \underline{E}+T+T \Rightarrow a+\underline{T}+T \Rightarrow a+\underline{E}+T \Rightarrow$
$\Rightarrow a+b+\underline{T} \Rightarrow a+b+\underline{E} \Rightarrow a+b+c$

Правосторонний вывод:

$\underline{S} \Rightarrow S+\underline{T} \Rightarrow S+\underline{E} \Rightarrow \underline{S}+c \Rightarrow \underline{S}+T+c \Rightarrow$
$\Rightarrow S+\underline{E}+c \Rightarrow \underline{S}+b+c \Rightarrow \underline{T}+b+c \Rightarrow$
$\Rightarrow \underline{E}+b+c \Rightarrow a+b+c$



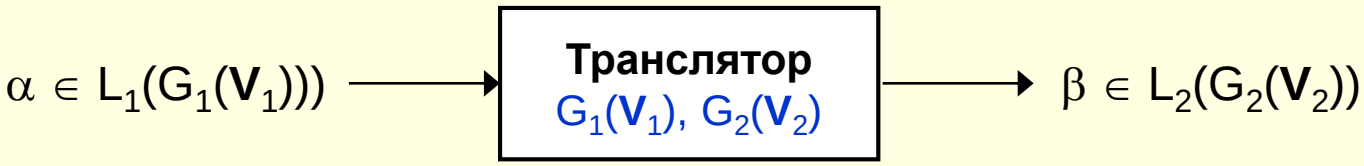
Грамматика однозначна

Задача трансляции

Задача трансляции – построение программы (выходной цепочки) на машинном языке некоторого компьютера, эквивалентной по смыслу исходной программе (входной цепочке)

- **Входная цепочка** – это программа для виртуального компьютера, которым является транслятор
- **Выходная цепочка** – это программа для хост-компьютера, на котором построен виртуальный компьютер, являющийся транслятором

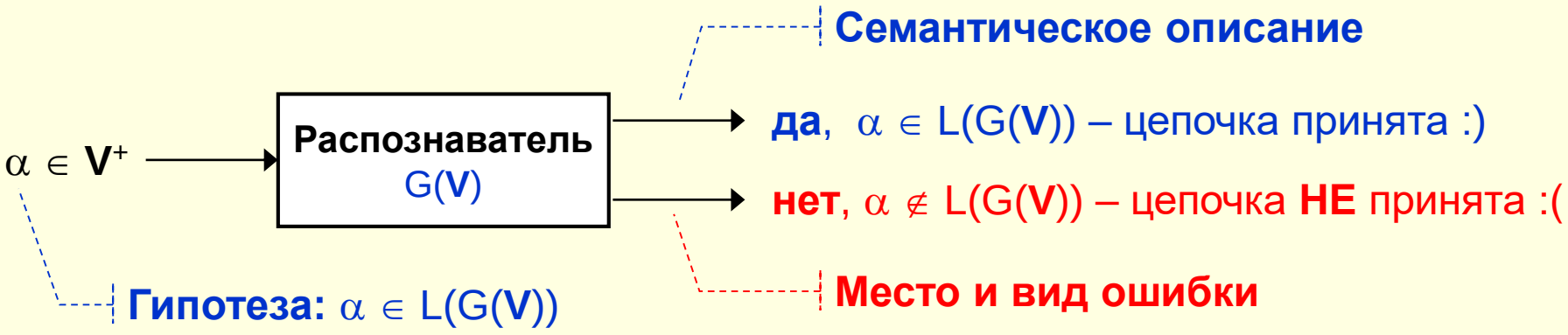
Трансляция – это перевод с одного языка на другой



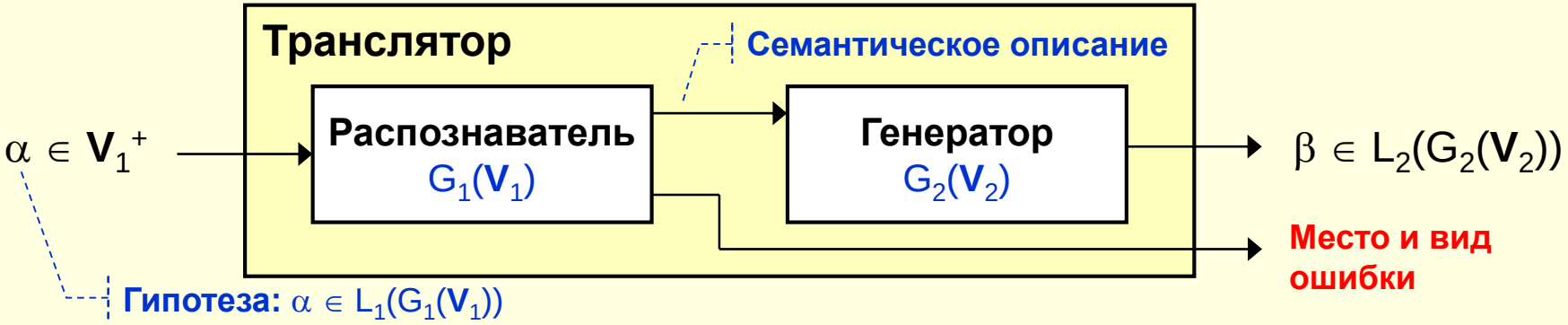
Задача трансляции

Распознаватель – основа транслятора, осуществляет грамматический разбор входной цепочки на основе грамматики, порождающей этот язык

- Алгоритм, позволяющий определить принадлежность цепочки некоторому языку
- Один из способов задания языка



Задача трансляции



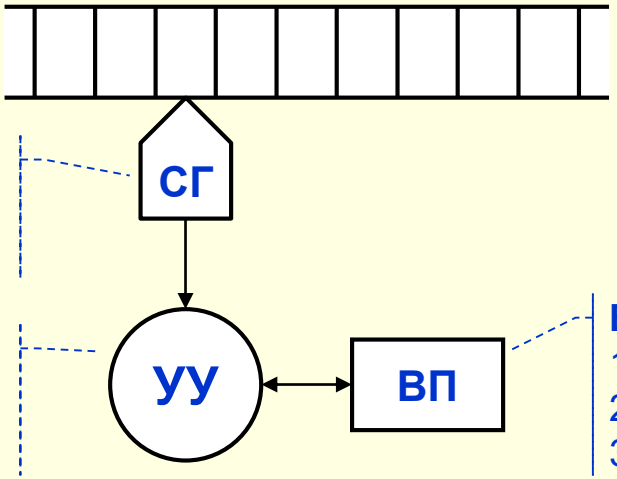
$$V_1 \neq V_2$$
$$G_1(V_1) \neq G_2(V_2)$$
$$L_1(G_1(V_1)) \neq L_2(G_2(V_2))$$
$$\alpha \neq \beta$$
смысл(α) = смысл(β)

Распознаватели языков

Тип	Характеристика	Эквивалент
Тип 0 Неограниченные	На правила грамматики не накладывается никаких ограничений – описывают языки любой сложности, в т.ч. естественные, но не используется для описания ЯП	Неограниченный автомат (НА) – машины Тьюринга, Поста и т.п. $a^n b^{f(n)}$
Тип 1 Контекстно-зависимые	Достаточны для описания ЯП, но слишком сложны и мало изучены Используются для построения автоматизированных лингвистических систем (анализ, перевод)	Линейно-ограниченный автомат (ЛО-автомат) $a^n b^m c^n d^m$ $a^n b^n c^n$
Тип 2 Контекстно-свободные	Свойства хорошо изучены и разрешимы Существует множество классов КСГ Используются для описания большинства ЯП, пригодны для автоматического разбора	Автомат с магазинной памятью (МП-автомат) $a^n b^n$ $a^n b^m c^n$
Тип 3 Регулярные	Используются для построения лексических анализаторов (сканеров) и простейших трансляторов (командных процессоров)	Конечный автомат (КА) a^n $a^n b^m$

Абстрактная модель распознавателя

Лента (входная цепочка)



Считывающая головка
1. Однонаправленная (вперед)
2. Двунаправленная (с возвратом)

Устройство управления (КА)
1. Детерминированное
2. Недетерминированное

Конфигурация
- Содержимое ленты
- Положение СГ
- Состояние УУ
- Содержимое ВП

Внешняя память
1. Отсутствует
2. Ограниченная (стек)
3. Неограниченная

Тип	СГ	УУ	ВП	Время распознавания	Функция
Тип 0 НГ	2 двунаправл.	2 недетерм.	3 неогранич.	∞ неприемлемо	нереализуемо
Тип 1 КЗГ	2 двунаправл.	2 недетерм.	2 огранич.	$k \cdot e^L$ ресурсоемко	семантический анализатор
Тип 2 КСГ	1 однонаправ.	2 недетерм.	2 огранич.	$P_n(L)$ **∃ алг-мы с n=1**	синтаксический анализатор
Тип 3 РГ	1 однонаправ.	2/1 детермин.	1 отсутствует	$k \cdot L$	лексический анализатор (сканер)

Структура транслятора

Во многих языках программирования присутствует определенный контекст:

- Локальные, глобальные и другие области определения (классы, пространства имен, пакеты и др.)
- Предварительное описание типов данных, переменных, функций и процедур (до их использования)
- Разнесенные объявления и реализации функций и процедур
- Разделы интерфейса и реализации

Такие языки относятся к контекстно-зависимым (тип 1), но отдельные конструкции описываются контекстно-свободными грамматиками (тип 2), а лексемы – регулярными (тип 3).

Структура транслятора

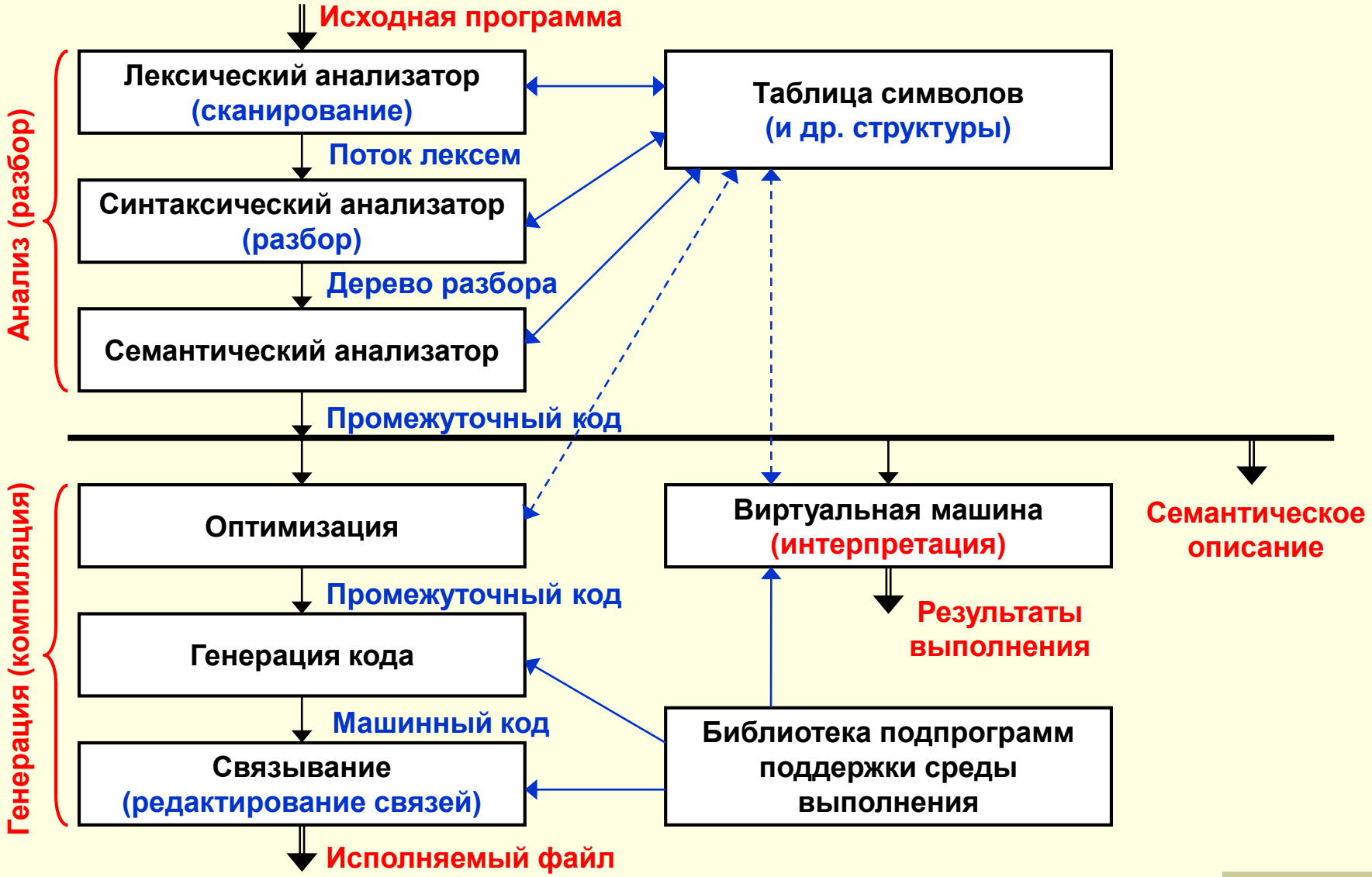
На практике для построения распознавателя используют:

- Лексический анализатор (сканер) – распознавание лексем на основе регулярных грамматик
- Синтаксический анализатор – распознавание конструкций (предложений) языка, состоящих из лексем, на основе контекстно-свободных грамматик
- Семантический анализатор – специально разработанный алгоритм (и необходимые структуры данных), реализующий контекстные зависимости

Виды трансляции по форме выходной информации:

- Семантическое описание – способ решения задачи
- Компиляция – программа, решающая задачу
- Интерпретация – результат решения задачи

Структура транслятора



Этапы трансляции

Этап трансляции	Задачи
Лексический анализ	■ Выявление лексем, определение их типов и значений (ключевые и зарезервированные слова, константы, идентификаторы и т.п.)
Синтаксический анализ	■ Выявление синтаксических единиц (выражения, операторы, описания, функции и т.п.)
Семантический анализ	■ Определение смысла синтаксических единиц ■ Поддержка таблицы символов ■ Включение неявной информации ■ Обнаружение ошибок, в т.ч. потенциальных ■ Макрообработка

Этапы трансляции

Этап трансляции	Задачи
Оптимизация ■ машинонезависимая ■ машинозависимая ■ по скорости ■ по объему (кода, данных)	■ Исключение неиспользуемых и промежуточных переменных ■ Вычисление значений константных выражений, в т.ч. включающих переменные ■ Однократное вычисление общих подвыражений, создание промежуточных переменных ■ Использование регистров ■ Вынесение инвариантных конструкций из циклов и ветвлений ■ Объединение одинаковых фрагментов ■ Исключение переходов и ветвлений ■ Исключение неэффективного кода ■ Оптимизация ссылок и индексов ■ Реорганизация циклов (использование счетчика, структура цикла) ■ Векторизация и распараллеливание вычислений

Этапы трансляции

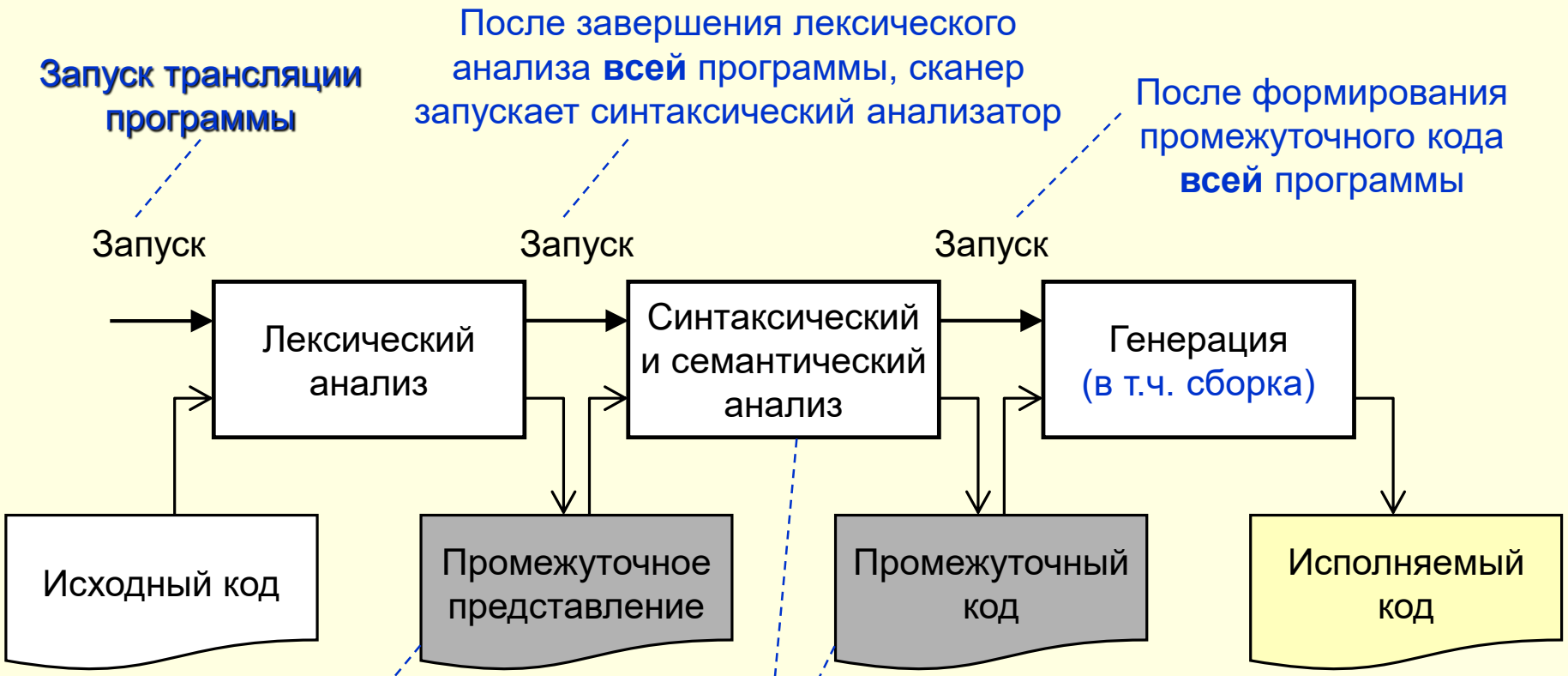
Этап трансляции	Задачи
Генерация	■ Создание машинного кода ■ Макрообработка и трансляция ассемблерных фрагментов ■ Оптимизация машинного кода (задачи аналогичны оптимизации промежуточного кода, но имеют более локальный характер)
Связывание	■ Объединение фрагментов, компилируемых отдельно ■ Компоновка сегментов ■ Вычисление адресов ■ Разрешение внешних ссылок ■ Создание таблиц загрузчика

Структура транслятора

- Многопроходный транслятор – каждый этап трансляции (лексический, синтаксический и семантический анализ, генерация) выполняется отдельно и полностью, в результате чего формируется промежуточное представление всей программы, которое является входным для следующего этапа
- Двухпроходный транслятор – в результате разбора программы (лексический, синтаксический и семантический анализ) формируется промежуточное представление всей программы, которое является входным для генератора кода на целевом языке
- Однопроходный транслятор – разбор входной программы и генерация кода на целевом языке выполняются за один проход

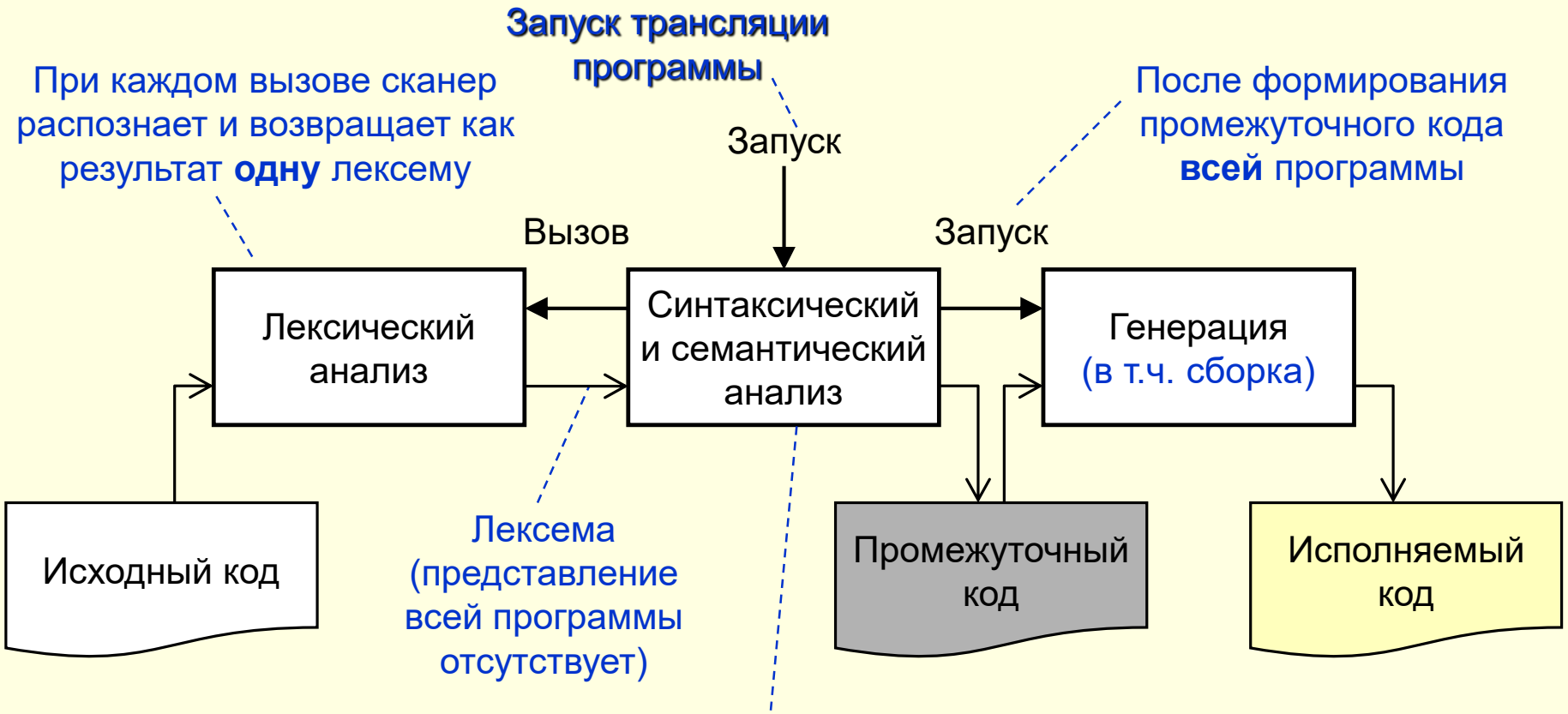
Структура транслятора

- Многопроходный транслятор



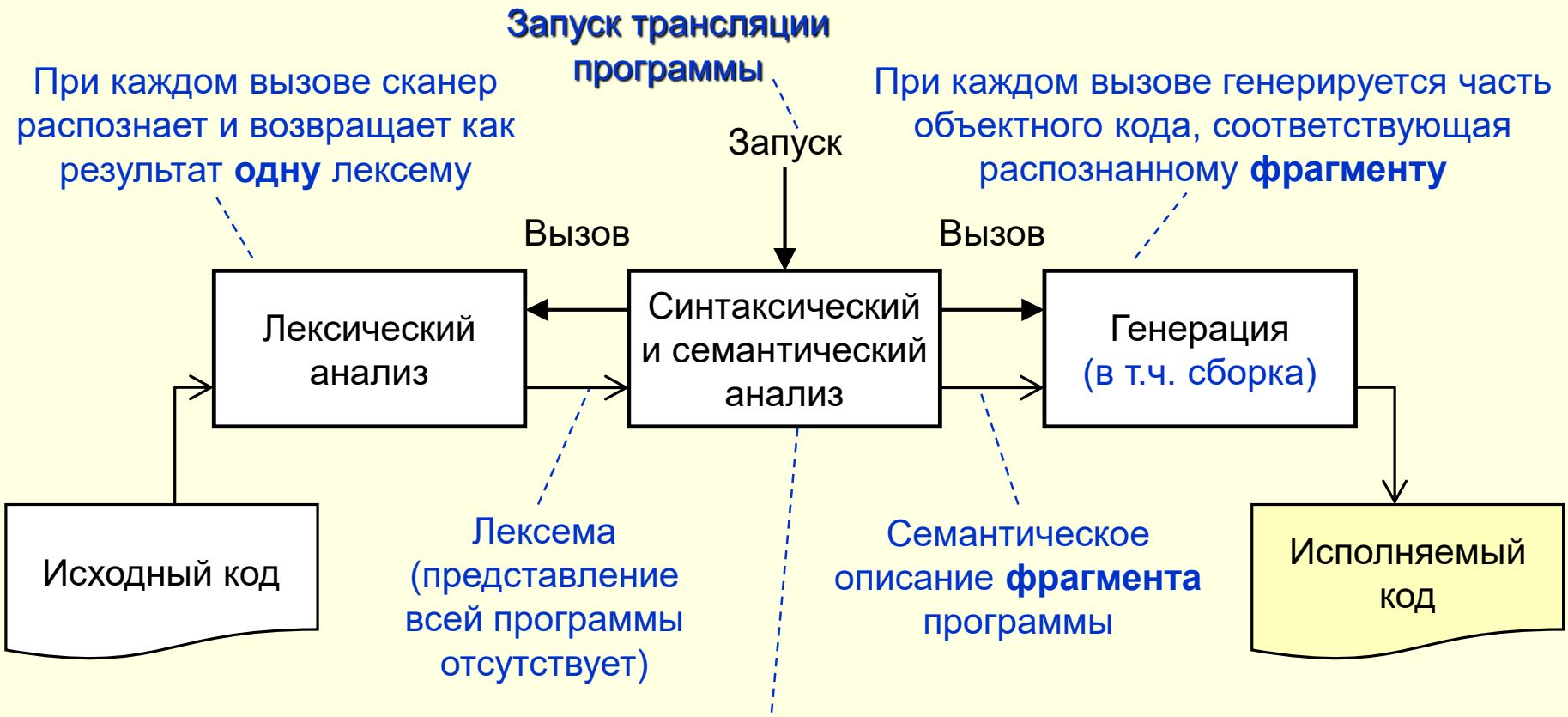
Структура транслятора

- Двухпроходный транслятор



Структура транслятора

- Однопроходный транслятор



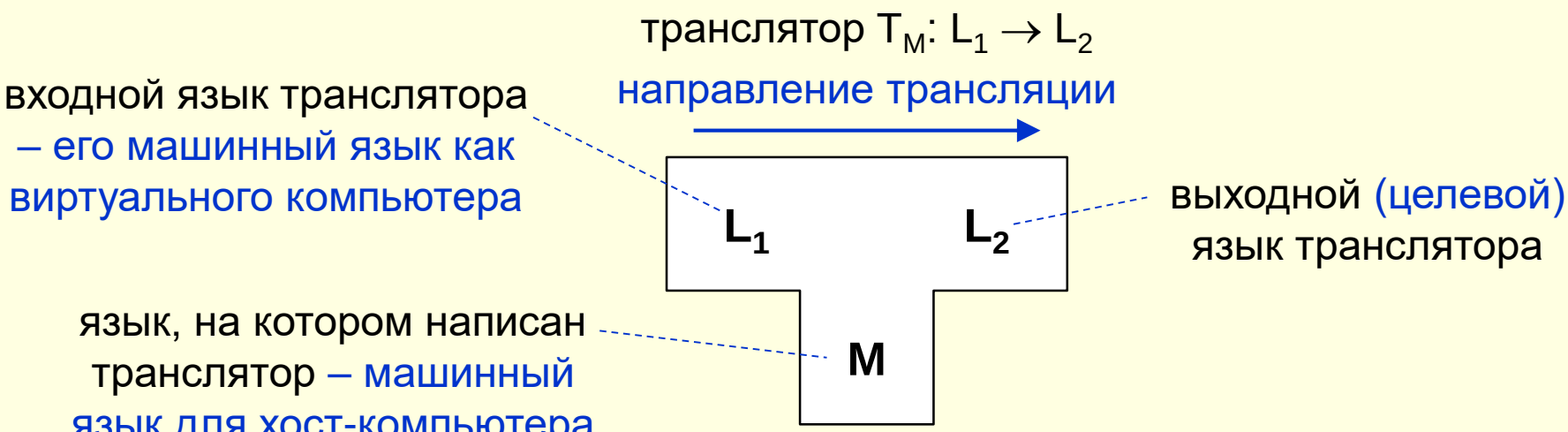
Разработка транслятора

Методики разработки компиляторов

- прямой метод – целевым языком и языком реализации является язык ассемблера или машинный язык
- метод раскрутки – языком реализации является разрабатываемый язык или его подмножество
- использование кросс-трансляторов – целевым языком является машинный язык, отличный от языка реализации
- использование виртуальных машин – целевым языком является интерпретируемый язык (машинный язык виртуальной машины)
- компиляция на лету (JIT – Just In Time, динамическая компиляция) – трансляция в машинный код осуществляется однократно фрагментами «по требованию»

Разработка транслятора

Т-схема (Т-диаграмма) – графическое представление транслятора как виртуального компьютера, написанного на языке, который является машинным для некоторого хост-компьютера, и выполняющего преобразование программы на входном языке в выходной – перевод программы с машинного языка транслятора в целевой язык, машинный для некоторого другого компьютера

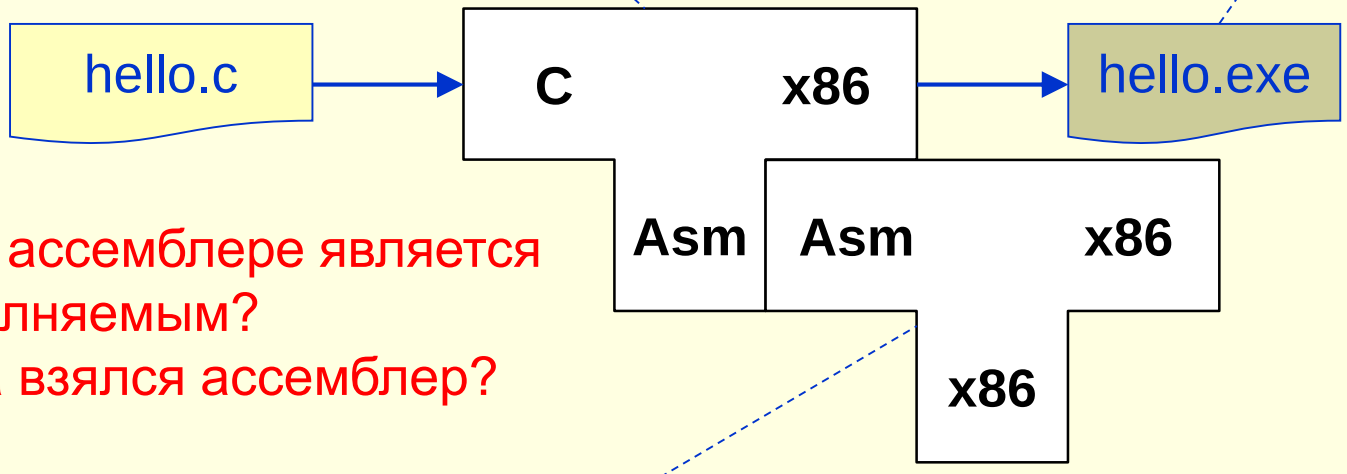


Разработка транслятора

- Прямой метод

транслятор языка C в машинный код x86, написанный на ассемблере

исполняемый файл, который выполняется непосредственно аппаратным компьютером x86



Код на ассемблере является исполняемым?

Откуда взялся ассемблер?

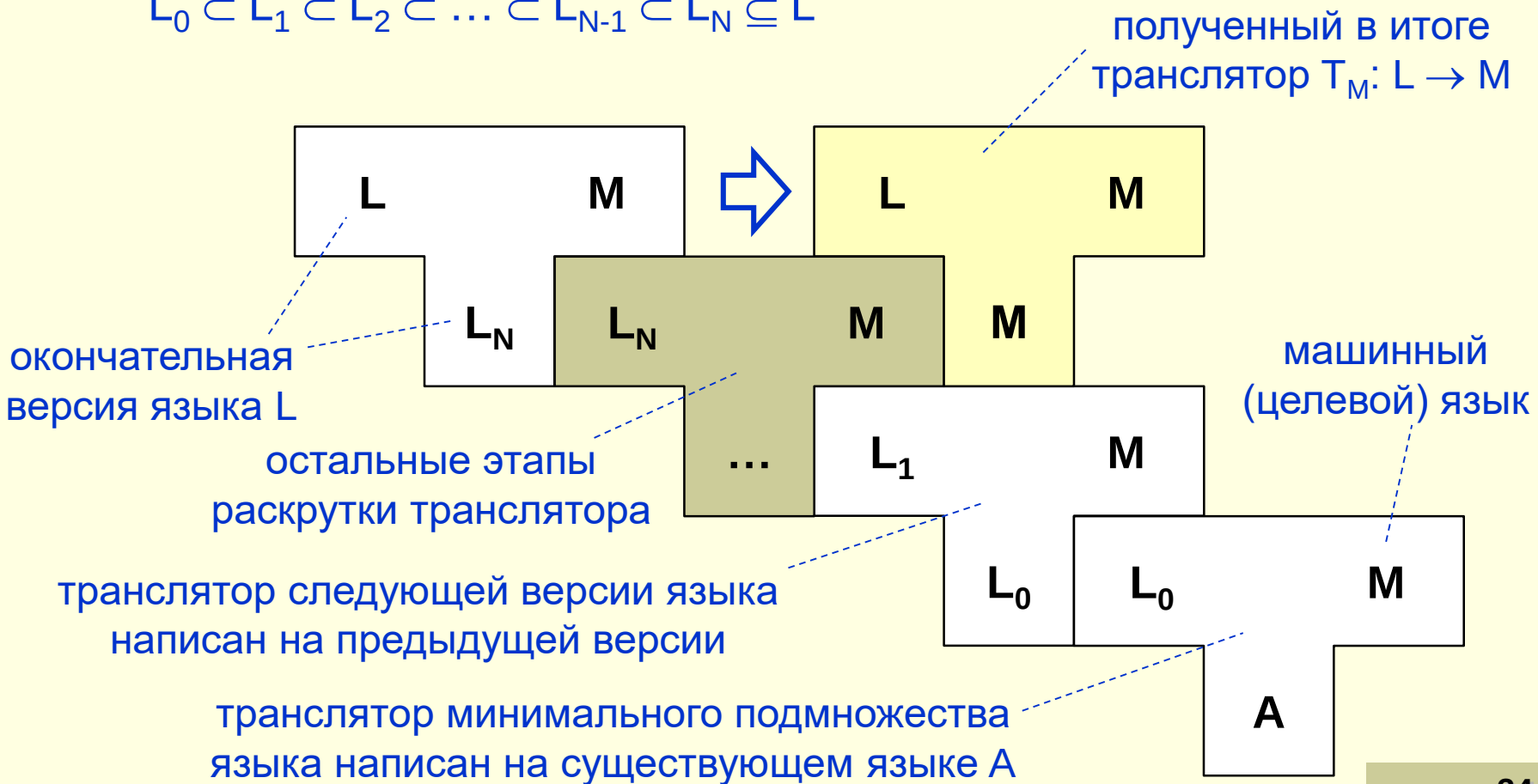
транслятор ассемблера, написанный в машинном коде x86

Разработка транслятора

- **Метод раскрутки**

  Итерационная реализация языка как последовательности версий, каждая из которых является подмножеством следующей

  $L_0 \subset L_1 \subset L_2 \subset \ldots \subset L_{N-1} \subset L_N \subseteq L$

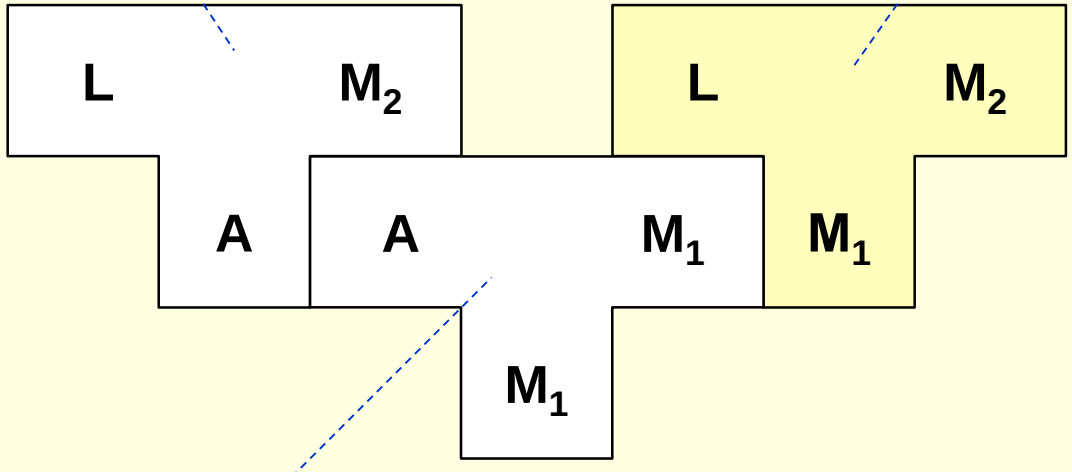


Разработка транслятора

- Кросс-транслятор

транслятор языка L в машинный код M_2

транслятор языка L в машинный код M_2, исполняемый на платформе M_1



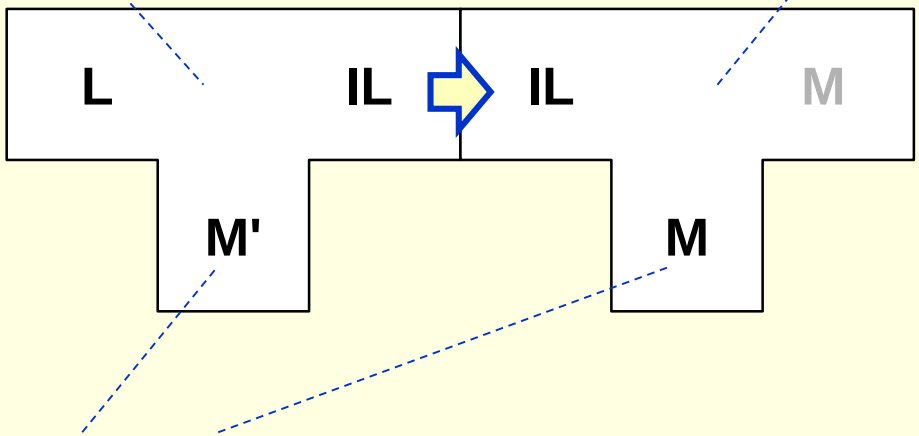
написан на некотором языке A, для которого существует транслятор для платформы M_1

Разработка транслятора

- Виртуальная машина

транслятор языка L в интерпретируемый язык IL (промежуточный язык – Intermediate Language, байт-код и т.п.)

виртуальная машина – интерпретатор языка IL, исполняемый на платформе M



трансляция в промежуточный язык может выполняться как на целевой платформе M (M' = M), так и на некоторой другой M' – этим обеспечивается кроссплатформенность ПО