

Лекция 1

**Введение
в конструирование
компиляторов**

§1. Компиляторы и интерпретаторы: основные определения

Определение. *Компилятор* (compiler) из языка S в язык T – это программа, осуществляющая перевод программ с языка S на язык T . При этом:

S – *исходный язык* (source language);

T – *целевой язык* (target language, object language);

язык H , на котором написан компилятор, – *язык реализации* (implementation language, host language).

Мы будем использовать запись $S \xrightarrow{H} T$ для обозначения компилятора из языка S в язык T , написанного на языке H .

Транслятор (translator) – синоним компилятора.

Определение. *Интерпретатор* (interpreter) для некоторого языка L – это программа, осуществляющая выполнение программ, написанных на языке L .

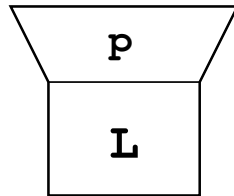
Компиляторы и интерпретаторы близки по структуре.

Компилятор из A в B можно считать интерпретатором языка A' такого, что синтаксис A' совпадает с синтаксисом A , а смысл программы p , написанной на A' , заключается в порождении программы на языке B , вычисляющей ту же функцию, что и программа p , если понимать её написанной на языке A .

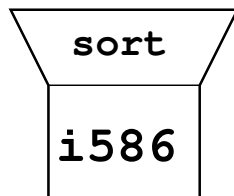
§2. Т-диаграммы

Т-диаграммы позволяют визуализировать те взаимодействия программ, в которые вовлечены компиляторы и интерпретаторы.

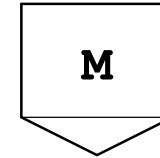
Программа p на языке L :



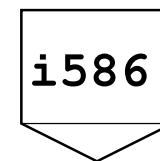
Пример:



Машина, выполняющая язык M :

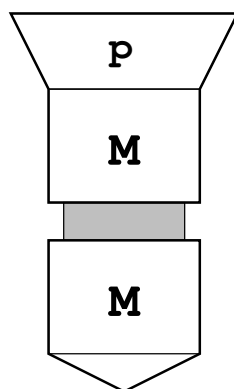


Пример:

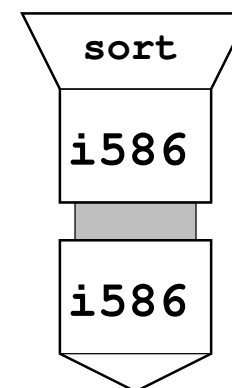


Для того чтобы выполнить программу на некоторой машине, язык, на котором написана программа, должен совпадать с языком, который понимает машина.

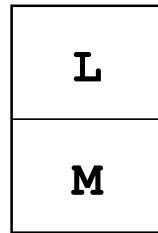
Выполнение программы p :



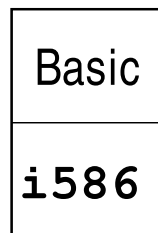
Пример:



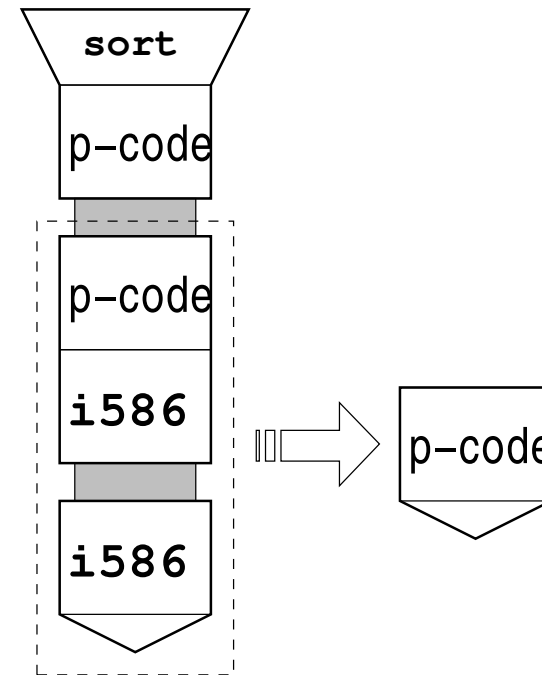
Интерпретатор языка L ,
написанный на языке M :



Пример:

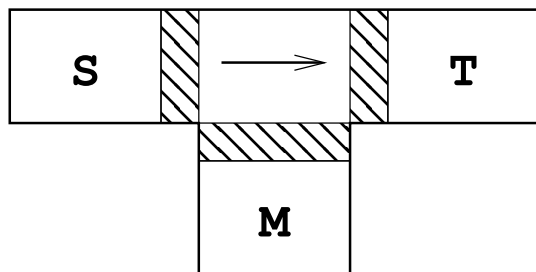


Пример
(виртуальная р-машина):

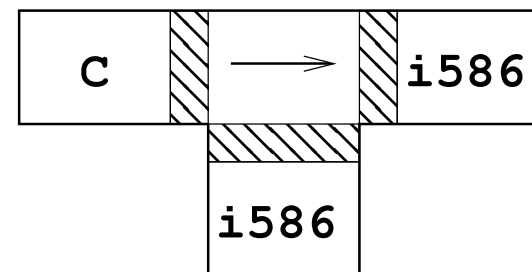


Определение. Виртуальная машина для языка L – это сочетание интерпретатора языка L , написанного на языке M , и машины, выполняющей язык M .

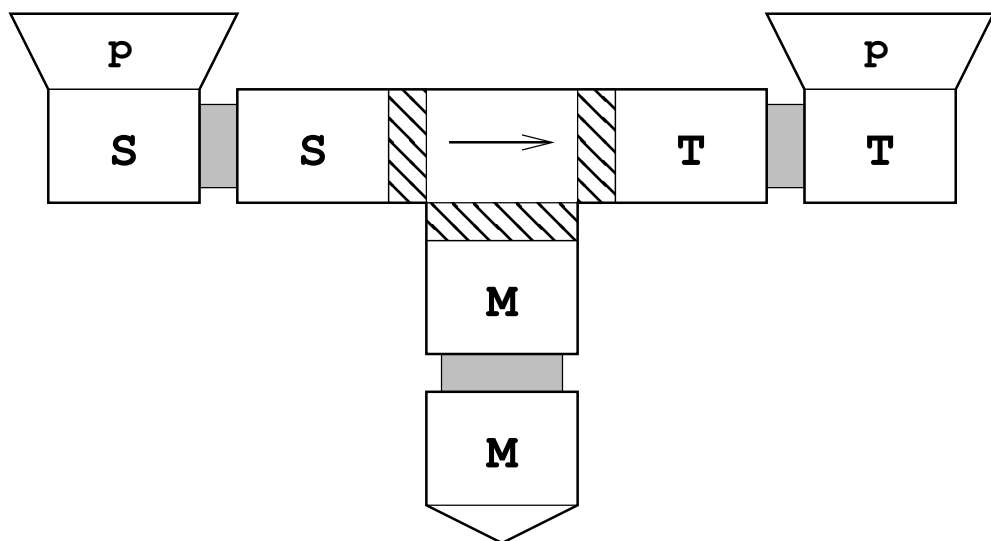
Компилятор $S \xrightarrow{M} T$:



Пример:

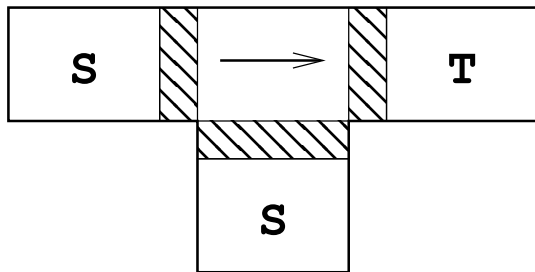


Компилятор $S \xrightarrow{M} T$, будучи запущен на машине, выполняющей язык M , переводит программы с языка S на язык T .



§3. Самоприменимые компиляторы: раскрутка и перенос

Определение. Компилятор $S \xrightarrow{S} T$ называется *самоприменимым* (self-applicable).

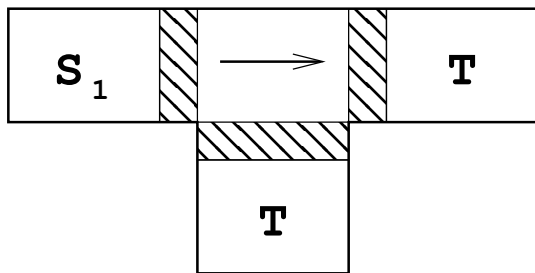


Если для языка S не существует другого компилятора или интерпретатора, то запуск самоприменимого компилятора является непростым делом.

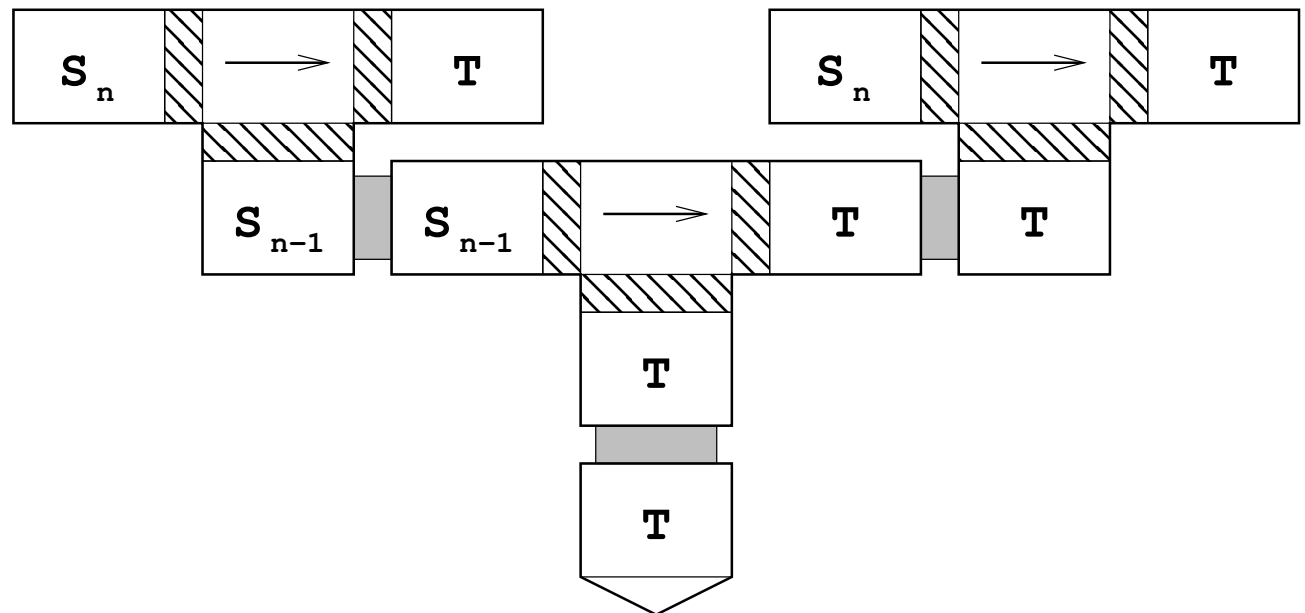
Определение. *Раскрутка* (bootstrapping) – это итерационный процесс создания самоприменимого компилятора $S \xrightarrow[S]{} T$, используемый в случае, если есть возможность запуска программ на языке T , а для программ на языке S такой возможности нет.

На каждом шаге раскрутки получается компилятор $S_i \xrightarrow[T]{} T$ для всё большего и большего подмножества S_i языка S .

Шаг 1.

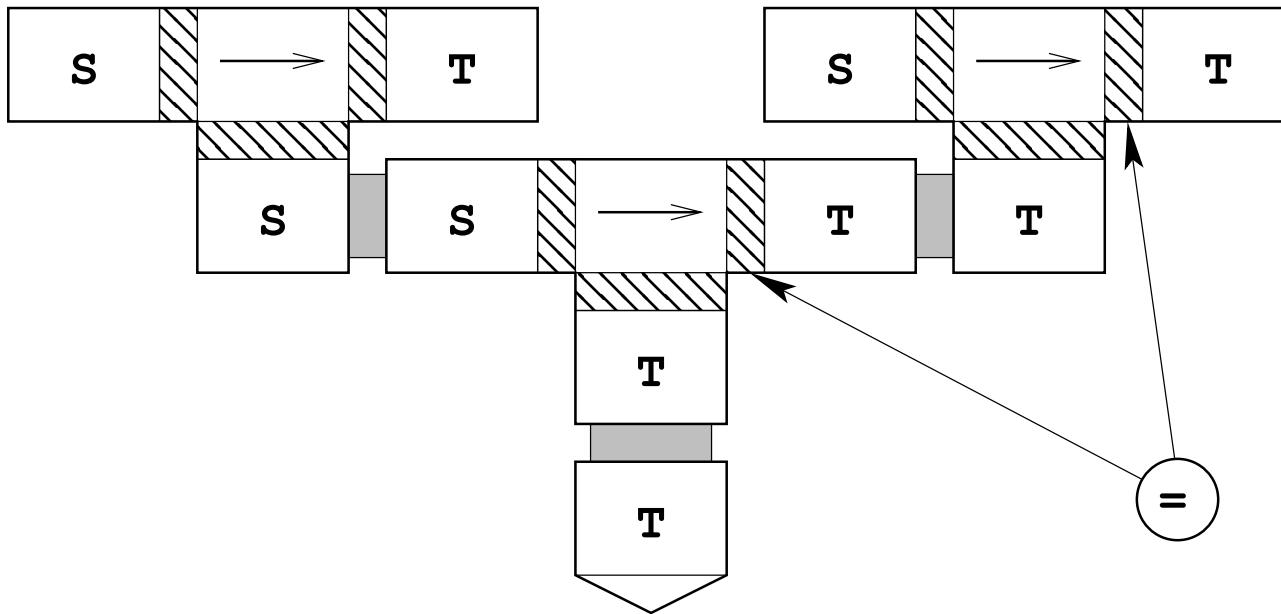


Шаг n .



$$S_1 \subset \dots \subset S_{n-1} \subset S_n \subset S$$

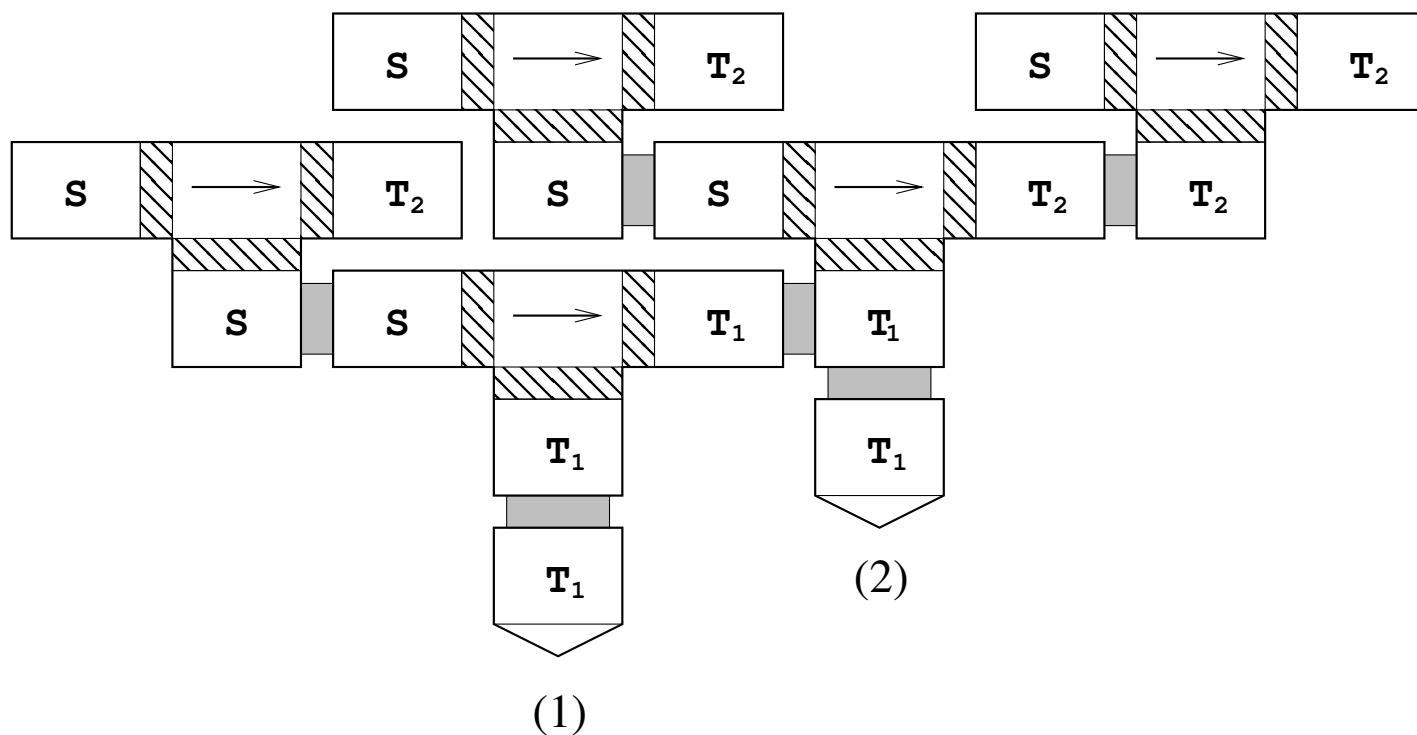
Определение. *Раскрученный самоприменимый компилятор языка S на платформе T – это пара компиляторов $(S \xrightarrow[S]{T} T, S \xrightarrow[T]{S} T)$ такая, что применение компилятора $S \xrightarrow[T]{S} T$ к компилятору $S \xrightarrow[S]{T} T$ даёт $S \xrightarrow[T]{T} T$.*



Для переноса раскрученного самоприменимого компилятора

$$\left(S \xrightarrow[S]{T_1}, S \xrightarrow{T_1} T_1 \right)$$

на платформу T_2 достаточно на основе $S \xrightarrow[S]{T_1} T_1$ разработать $S \xrightarrow[S]{T_2} T_2$, а затем выполнить два шага:



Компилятор $S \xrightarrow[S]{T_2} T_2$ называется *кросскомпилятором*.

§4. Внедрение «троянских коней» с помощью самоприменимых компиляторов

По статье:

Ken Thompson. *Reflections on Trusting Trust* // Communications of the ACM. Vol. 27, №8. August 1984.

В статье рассматривается раскрученный самоприменимый компилятор языка C:

$$\left(C \xrightarrow{C} \mathbf{Asm}, C \xrightarrow{\mathbf{Asm}} \mathbf{Asm} \right)$$

«Идеализированный» фрагмент компилятора $C \xrightarrow{\bar{C}} \text{Asm}$, обрабатывающий Escape-последовательности:

```
c = next();  
if (c != '\\')  
    return c;
```

```
c = next();  
switch (c)  
{  
case '\\': return '\\';  
case 'n':  return '\n';  
}
```

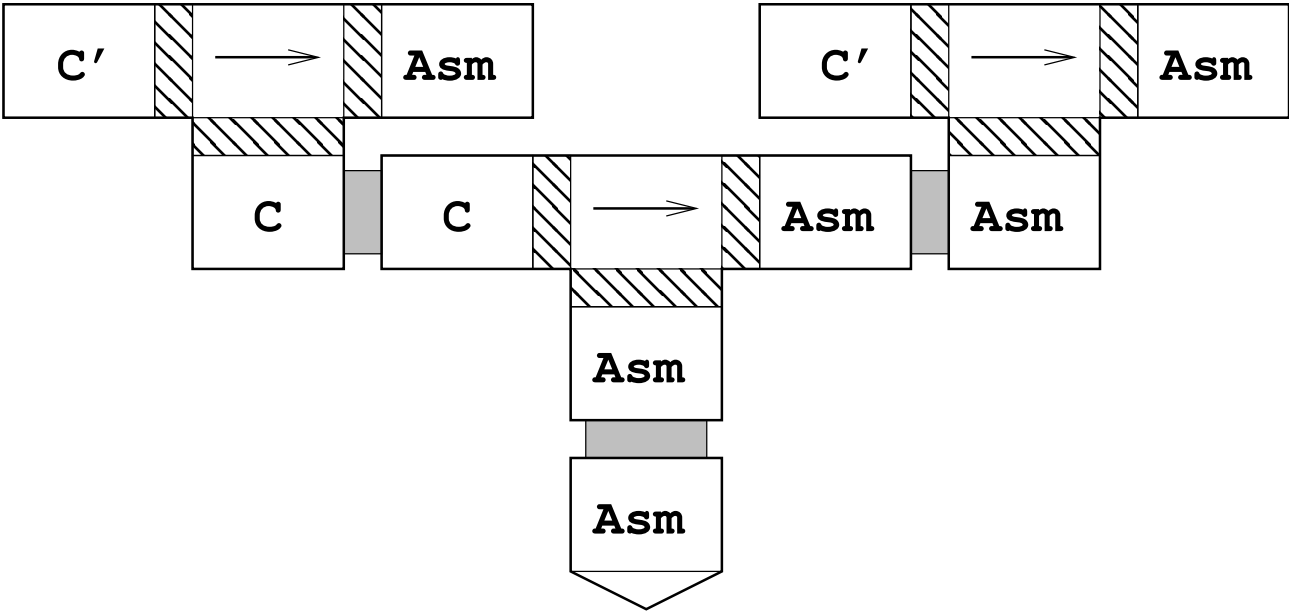
Попробуем добавить в $C \xrightarrow{C} \text{Asm}$ последовательность `\t`:

```
c = next();  
if (c != '\\')  
    return c;
```

```
c = next();  
switch (c)  
{  
case '\\': return '\\';  
case 'n':  return '\\n';  
case 't':  return 9;  
}
```

В результате получим компилятор $C' \xrightarrow{C} \text{Asm}$.

Выполняем раскрутку:



Теперь мы можем «переписать» $C' \xrightarrow{C} \text{Asm}$ на C' :

```
c = next();  
if (c != '\\')  
    return c;
```

```
c = next();  
switch (c)  
{  
case '\\': return '\\';  
case 'n':  return '\n';  
case 't':  return '\t';  
}
```

В результате получим самоприменимый компилятор $C' \xrightarrow{C'} \text{Asm}$, в тексте которого отсутствует ASCII-код символа `'\t'`. Дело в том, что этот ASCII-код «ушёл» в $C' \xrightarrow{\text{Asm}} \text{Asm}$!

Используем раскрутку для внедрения «трояна». Для этого рассмотрим функцию compile() нашего гипотетического компилятора $C \xrightarrow{C} \text{Asm}$:

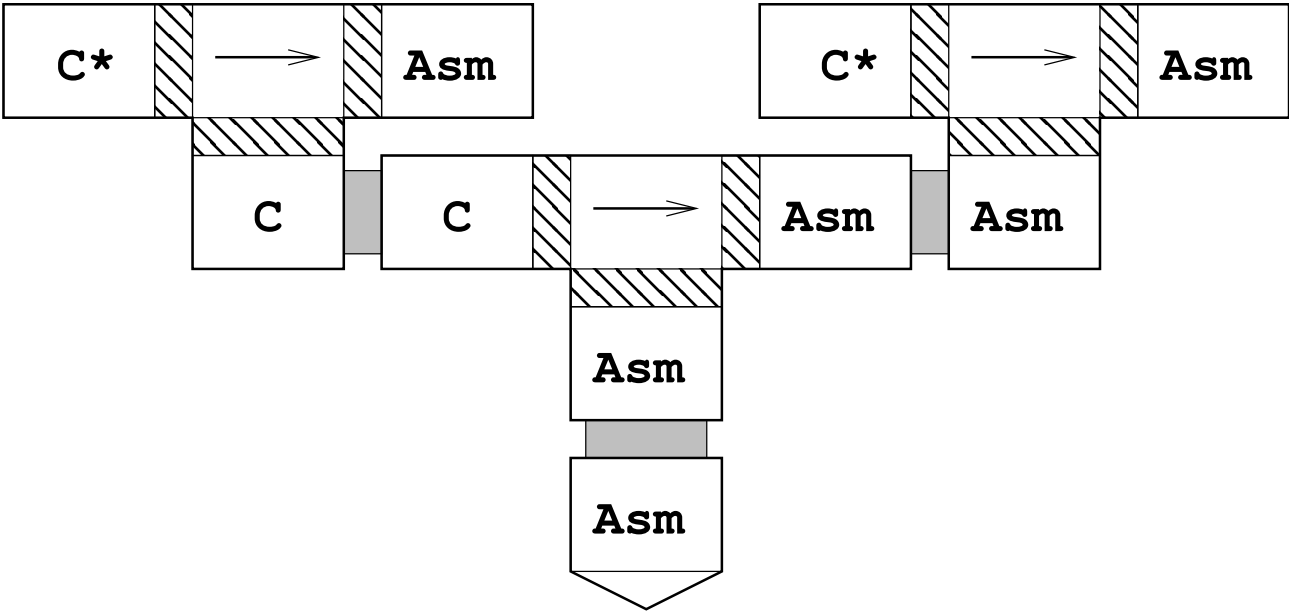
```
void compile(char *prog)
{
    . . .
    /* Компиляция. */
    . . .
}
```

Модифицируем код функции:

```
void compile(char *prog)
{
    if (match(prog, LOGIN_TEXT))
        prog = "... заражённый login ...";
    else if (match(prog, CC_TEXT))
    {
        ....
        /* Внедрение заразы в текст компилятора. */
        ....
    }
    ....
    /* Компиляция. */
    ....
}
```

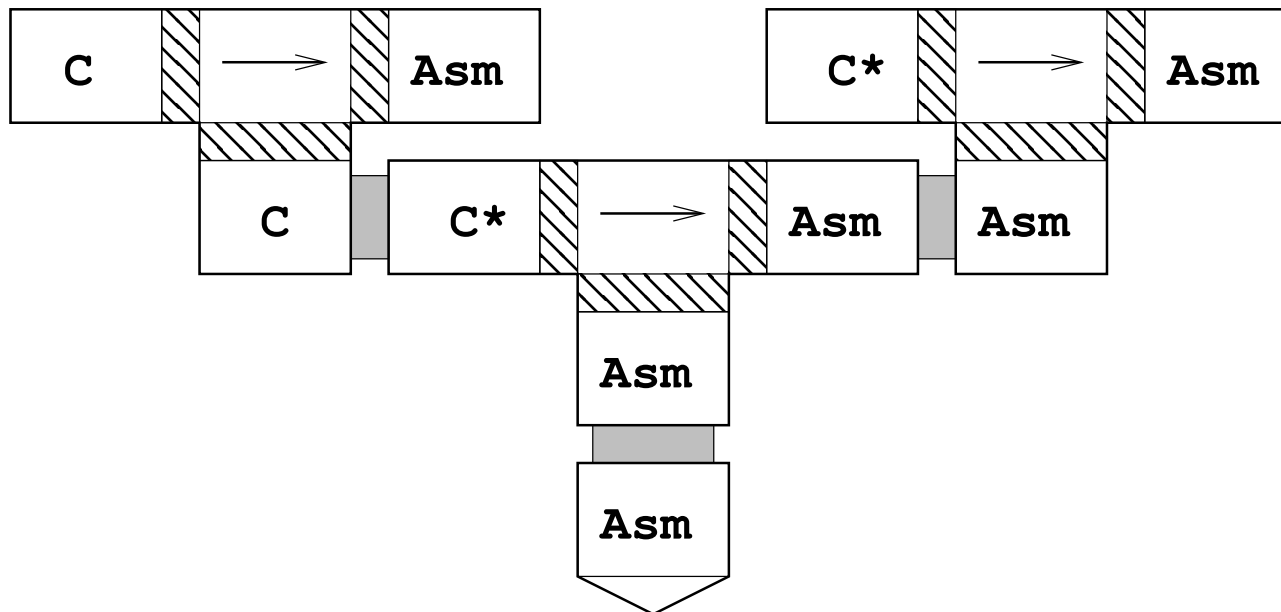
Получаем компилятор $C^* \xrightarrow{C} \text{Asm}$, подменяющий текст программы login и «заражающий» текст компилятора $C \xrightarrow{C} \text{Asm}$.

Выполняем раскрутку:



Выдаём пару $\left(C \xrightarrow{C} \text{Asm}, C^* \xrightarrow{\text{Asm}} \text{Asm} \right)$ за раскрученный самоприменимый компилятор языка C.

Никто не заметит подвоха, так как:



Дополнение. Программа на C, печатающая свой текст.

```
char *s = "\\n\\n";
int main()
{
    int i;
    printf("char *s = \\\"\\");
    for (i = 0; s[i]; i++)
        switch (s[i])
        {
            case '\\n': printf("\\n"); break;
            case '\\': printf("\\\\"); break;
            case '\\\"': printf("\\\\\""); break;
            case '\\\\': printf("\\\\\\\\"); break;
            default:    printf("%c", s[i]);
        }
    printf(s);
    return 0;
}
```