Суперкомпиляция как основа для построения алгоритмов решения уравнений в словах

А. Н. Непейвода
Институт Программных систем им. А.К. Айламазяна РАН

IV совместное рабочее совещание
по функциональному языку Рефал
8 июня 2021

Постановка задачи

Дана система уравнений в словах $\mathcal{E}qs$. Существует ли последовательность подстановок σ, порождающая решение $\mathcal{E}qs$?

Уравнения в словах

Определение

Даны алфавит констант Σ и переменных $\mathcal{V}$. *Уравнение в словах* — выражение $\Phi = \Psi$, где $\Phi, \Psi \in \{\Sigma \cup \mathcal{V}\}^*$. Решение уравнения в словах — подстановка $\sigma : \mathcal{V} \to \Sigma^*$ такая, что $\Phi\sigma$ буквально совпадает с $\Psi\sigma$.

$E = xAB = BAx$, где $A, B \in \Sigma$, $x \in \mathcal{V}$. Рассмотрим $\sigma_1 : x \to Bx$, $\sigma_2 : x \to \varepsilon$. Тогда $\sigma_2 \circ \sigma_1 : x \to B$ — решение E: $(xAB)\sigma_1\sigma_2 = BAB = (BAx)\sigma_1\sigma_2$.

Краткая история

Теория:

- Решение квадратичных уравнений ($\mathrm{x}\mathrm{A}\mathrm{y} = \mathrm{y}\mathrm{A}\mathrm{x}$) и уравнений от одной переменной (Матиясевич, 1965)
- Решение уравнений от трех переменных (Хмелевский, 1971)
- Алгоритм решения уравнений в общем случае (Маканин, 1977)
- Более быстрые (но также гиперэкспоненциальные) алгоритмы (Plandowski, 2006, Jez, 2016)

Краткая история

Практика:

- эффективные алгоритмы решения линейных ($\mathtt{xxx} = \mathtt{y}A\mathtt{z}$) уравнений (Rümmer et al., 2014–...)
- алгоритмы решения квадратичных уравнений (Le et al., Lin et al., 2018)
- алгоритмы решения уравнений с ограниченной длиной решений (Bjørner, 2009–..., Day, 2019)

Язык уравнений в словах

Определение

Множество закодированных уравнений:

Eqs ::= Eq Eqs | ε

Eq ::= (Side, Side)

Side ::= Char Side | Var Side | ε

Var $\in \mathcal{V}$, Char $\in \Sigma$, ε — пустое слово.

Синтаксический сахар:

(LHS, RHS) $\longrightarrow$ LHS = RHS;

(LHS$_1$, RHS$_1$)...(LHS$_n$, RHS$_n$) $\longrightarrow \left\langle \text{ LHS}_i = \text{RHS}_i \right\rangle_{i=1}^{n}$.

Простой логический язык $\mathscr{L}$

Определение

Определим последовательность сужений Narrs.

Narrs ::= (Narr) Narrs | ε

Narr ::= $'\mathrm{Var} \to \mathrm{Char}\ \mathrm{Var}' \mid '\mathrm{Var} \to \mathrm{Var}_1\,\mathrm{Var}' \mid '\mathrm{Var} \to \varepsilon'$

$\mathrm{Var}, \mathrm{Var}_1 \in \mathcal{V}$, $\mathrm{Char} \in \Sigma$, $\mathrm{Var} \neq \mathrm{Var}_1$.

Любая последовательность из Narrs определяет подстановку $\sigma : \mathcal{V} \to (\mathcal{V} \cup \Sigma)^*$. Пусть $\mathrm{x} \in \mathcal{V}$, тогда σ — это $\mathrm{x} \to \Phi$ или $\mathrm{x} \to \Phi\mathrm{x}$, где Φ не содержит x.

Множество последовательностей из Narrs рассматриваем как простой логический язык $\mathscr{L}$ над Eqs.

Простой логический язык $\mathscr{L}$

Определение

Определим последовательность сужений Narrs.

Narrs ::= (Narr) Narrs | ε

Narr ::= $'\mathrm{Var} \to \mathrm{Char}\ \mathrm{Var}'$ | $'\mathrm{Var} \to \mathrm{Var}_1\,\mathrm{Var}'$ | $'\mathrm{Var} \to \varepsilon'$

Совместимость сужений с $\langle \Phi_1 = \Psi_1, \ldots, \Phi_n = \Psi_n \rangle$:

$'\mathrm{x} \to \varepsilon'$	$\mathrm{x}\Phi_1 = \Psi_1$ или $\Phi_1 = \mathrm{x}\Psi_1$	$'\mathrm{x} \to \mathrm{t}\mathrm{x}'$	$\mathrm{x}\Phi_1 = \mathrm{t}\Psi_1$ или $\mathrm{t}\Phi_1 = \mathrm{x}\Psi_1$
$'\mathrm{x} \to \mathrm{x}_1\mathrm{x}'$	$\mathrm{x}\Phi_1 = \mathrm{x}_1\Psi_1$ или $\mathrm{x}_1\Phi_1 = \mathrm{x}\Psi_1$		

Множество последовательностей из Narrs рассматриваем как простой логический язык $\mathscr{L}$ над Eqs.

Семантика $\mathscr{L}$

Интерпретатор $\mathscr{L}$ $\mathrm{WI}_{\mathscr{L}}$ принимает последовательность $(\sigma_1)(\sigma_2)...(\sigma_n)$ и систему $\langle \Phi_i = \Psi_i \rangle_{i=1}^{m}$.

Вызов $\mathrm{WI}_{\mathscr{L}}((\sigma_1)(\sigma_2)...(\sigma_n), \langle \Phi_i = \Psi_i \rangle_{i=1}^{m})$ возвращает T, если $\forall i,\ 1 \leqslant i \leqslant m\ (\Phi_i\sigma_1...\sigma_n = \Psi_i\sigma_1...\sigma_n)$, и F иначе.

Свойства интерпретатора $\mathrm{WI}_{\mathscr{L}}$:

- совершает не больше n шагов (всегда завершается);
- для всех $\langle \Phi_i = \Psi_i \rangle_{i=1}^{m}$ возвращает либо T, либо F.

Суперкомпиляция интерпретаторов языка $\mathscr{L}$

В вызове $\mathrm{WI}_{\mathscr{L}}(\mathrm{P}, \langle \Phi_i = \Psi_i \rangle_{i=1}^{n})$ заменяем последовательность P на параметр $\mathcal{P}$.

Получаем следующую задачу суперкомпиляции:

$$\mathrm{WI}_{\mathscr{L}}(\mathcal{P}, \langle \Phi_i = \Psi_i \rangle_{i=1}^{n})$$

Ее развертка порождает дерево (возможно, бесконечное): описание путей вычисления всех возможных $\mathscr{L}$-программ на $\langle \Phi_i = \Psi_i \rangle_{i=1}^{n}$.

Задача верификации

Пусть дана система уравнений $\mathcal{E}qs$. Скажем, что ее верификация успешна, если $\mathcal{E}qs$ если остаточная программа, порожденная суперкомпиляцией $\mathsf{WI}_{\mathscr{L}}(\mathcal{P}, \mathcal{E}qs)$, содержит функцию, возвращающую T, лишь в том случае, если $\mathcal{E}qs$ имеет решения.

Суперкомпиляция может уходить в бесконечную развертку для некоторых $\mathcal{E}qs$.

Синтаксис остаточных программ

Определение

Program ::= Rule; Program | ε

Rule ::= Name(Pattern) = Expression

Pattern ::= (Narr) | (Narr) ++ Pattern | $\mathtt{p}$ | ε

Expression ::= T | F | Name($\mathtt{p}$)

$\mathtt{p}$ пробегает Narrs, Name — имя функции.

Все функции — от одного аргумента. В левых и правых частях определений — максимум по одному вхождению переменной $\mathtt{p}$.

Пример

Для уравнения $\mathtt{A}\,\mathtt{x} = \mathtt{x}\,\mathtt{A}$ суперкомпилятор порождает следующую программу со входной точкой $\mathtt{F}(\mathtt{p})$.

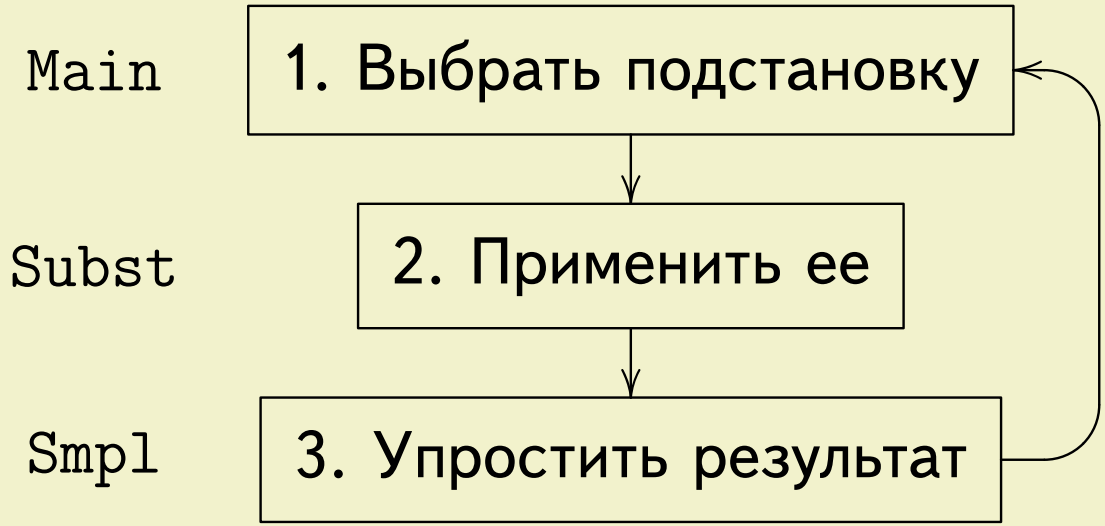
```
F(('x → ε')) = T
F(('x → Ax') ++ p) = F(p)
F(p) = F
```

Для уравнения $\mathtt{A}\,\mathtt{x} = \mathtt{x}\,\mathtt{B}$ остаточная программа будет такой: (входная точка — $\mathtt{G}(\mathtt{p})$).

```
G(('x → ε')) = F
G(('x → Ax') ++ p) = G(p)
G(p) = F
```

Общая структура интерпретаторов

Основной цикл интерпретации



- `Smpl` получает на вход статические данные (систему уравнений) и не меняет их семантику.
- `Smpl` всегда завершается.

Базовый интерпретатор $\mathrm{WIBase}_{\mathscr{L}}$

Структура функции `Smpl`

Удалить общий префикс → Удалить общий суффикс

Эту операцию далее называем нормированием.

Входной формат

`p` — последовательности правил;
$\mathcal{E}qs$ — (системы) уравнений.

$$\mathtt{Go}(\mathtt{p}, \mathcal{E}qs) = \mathtt{Main}(\mathtt{p}, \mathtt{Smpl}(\varepsilon, \mathcal{E}qs));$$

- Суперкомпиляция задачи $\mathrm{WIBase}_{\mathscr{L}}(\mathcal{P}, \Phi = \Psi)$ успешно решает все квадратичные уравнения $\Phi = \Psi$.

Интерпретатор с расщеплением $\mathrm{WISplit}_{\mathscr{L}}$

Структура функции `Smpl`

Входной формат

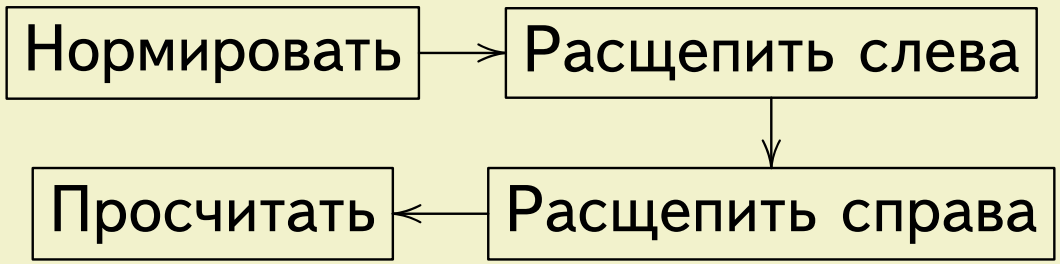
$$\mathtt{Go}(\mathtt{p}, \mathcal{E}qs) = \mathtt{Main}(\mathtt{p}, \mathtt{Smpl}(0, \varepsilon, \mathcal{E}qs));$$

- Первый аргумент `Smpl` добавлен для сохранения оптимальности (см. ниже).
- Суперкомпиляция задачи $\mathrm{WISplit}_{\mathscr{L}}(\mathcal{P}, \langle \Phi = \Psi \rangle)$ решает регулярно-упорядоченные уравнения $\Phi = \Psi$.

Интерпретатор с подсчетом $\mathrm{WICount}_{\mathscr{L}}$

Ищет противоречия путем сравнения мультимножеств термов с левой и правой стороны уравнения.

Структура функции `Smpl`



Входной формат

$$\mathtt{Go}(\mathtt{p}, \mathcal{E}qs) = \mathtt{Main}(\mathtt{p}, \mathtt{Smpl}(0, \varepsilon, \mathcal{E}qs));$$

- Суперкомпиляция $\mathrm{WICount}_{\mathscr{L}}(\mathcal{P}, \langle \Phi = \Psi \rangle)$ решает все уравнения $\Phi = \Psi$ от одной переменной.

Рекламная пауза

Классы уравнений, которые в общем не решаются CVC4 и Z3Str3, но решаются у нас:

- квадратичные уравнения, не имеющие решений ($x_1 x_2 x_3 ABABAB = AAABBB x_2 x_3 x_1$);
- регулярно-упорядоченные уравнения, не имеющие решений ($ABxxyy = xxyyBA$).

Уравнения от одной переменной, не решаемые CVC4 и Z3Str3 — те же регулярно-упорядоченные без решений.

Таблица тестирования

Тестовый набор	Тесты	Не завершились		
		CVC4	Z3str3	WICount$_{\mathscr{L}}$
Track 1 (Woorpje)	200	8	13	21
Track 5 (Woorpje)	200	4	14	19
Наш набор	50	21	28	10

Среднее время работы WICount$_{\mathscr{L}}$: 3,5 минуты на **один тест**.

Время работы CVC4 и Z3str3 — меньше 2 минут на все тесты.

Почему такая разница по времени?

Мы решаем разные задачи:

- солверы ищут одно решение;
- мы пытаемся построить описание всех решений.

Следствия

- солверы всегда решают безкоэффициентные уравнения (у них есть пустое решение);
- наш метод работает очень медленно на линейных уравнениях.

Другие интерпретаторы

- Интерпретатор, вычисляющий возможные длины переменных с последующим порождением подстановок вида $\mathtt{x} \to \mathtt{s}_1 \ldots \mathtt{s}_n$ ($\mathtt{s}_i$ — символьные переменные).
- Интерпретатор, позволяющий в некоторых случаях строить подстановки с конца уравнения.
- Интерпретатор, решающий систему линейных диофантовых уравнений на длины переменных в уравнениях (Олег Шатнюк).

Лемма об оптимальности

Лемма

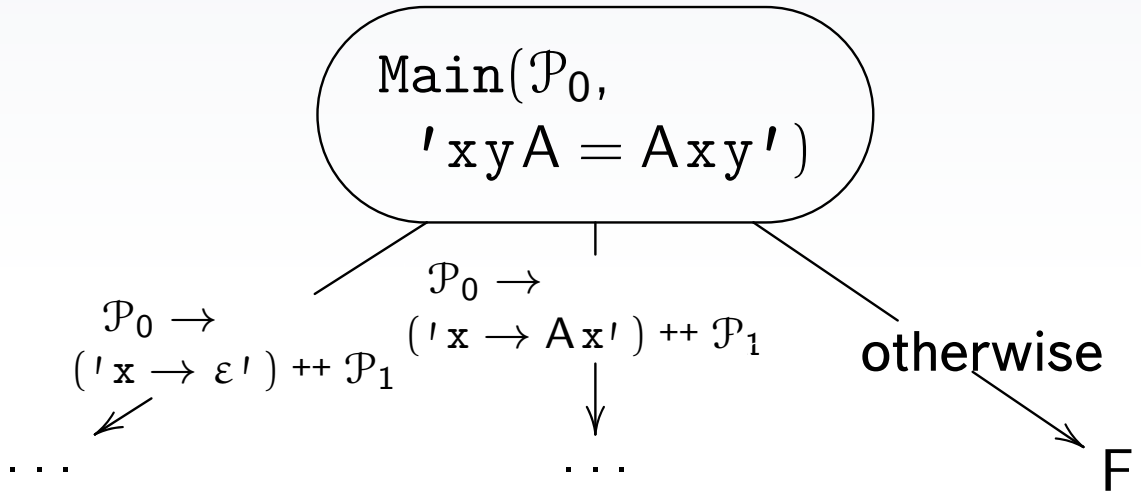
Все операции свертки в графе развертки $\mathrm{WI}_{\mathscr{L}}(\mathcal{P}, \langle \Phi_i = \Psi_i \rangle_{i=1}^{n})$ совершаются только над узлами, помеченными конфигурациями вида:

$$\mathtt{Main}(\mathcal{P}_j, \langle \Phi_i^j = \Psi_i^j \rangle_{i=1}^{n_j})$$

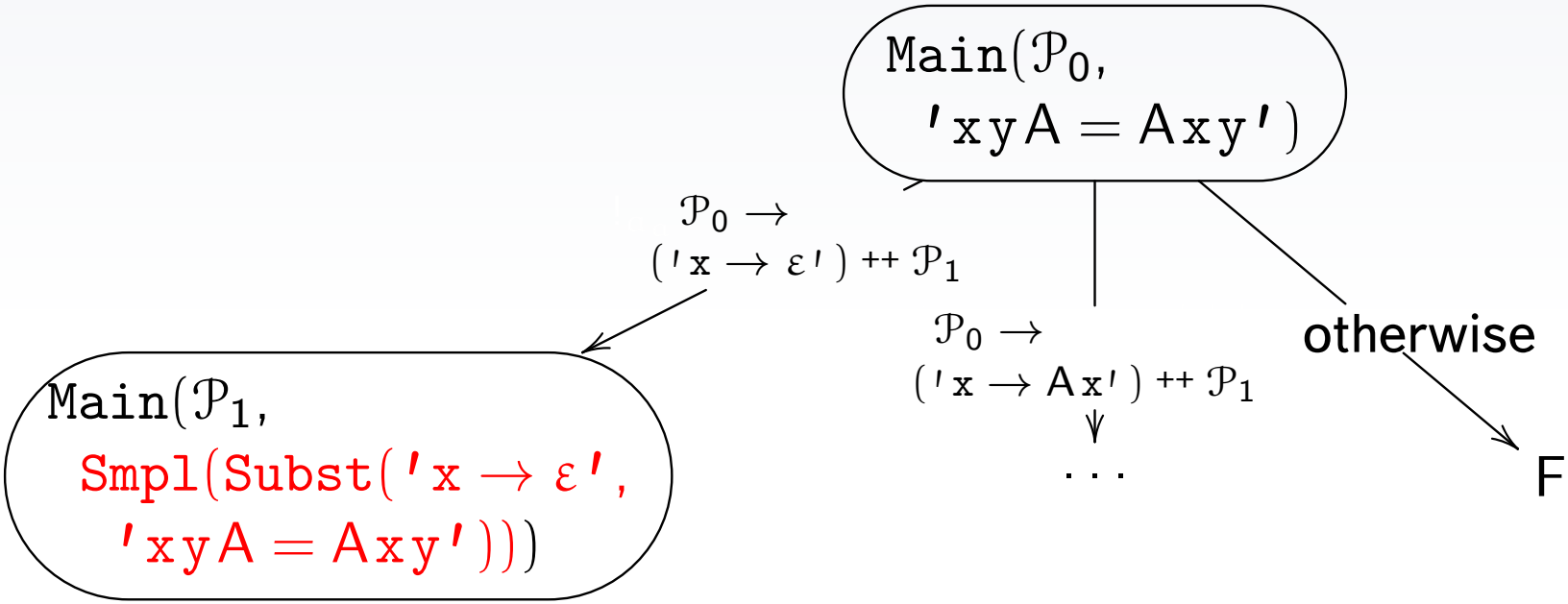
где $\mathcal{P}_j$, и других вхождений параметров нет.

Эта лемма определяет соответствие между графом развертки $\mathrm{WI}_{\mathscr{L}}(\mathcal{P}, \langle \Phi_i = \Psi_i \rangle_{i=1}^{n})$ и графом решения системы $\langle \Phi_i = \Psi_i \rangle_{i=1}^{n}$.

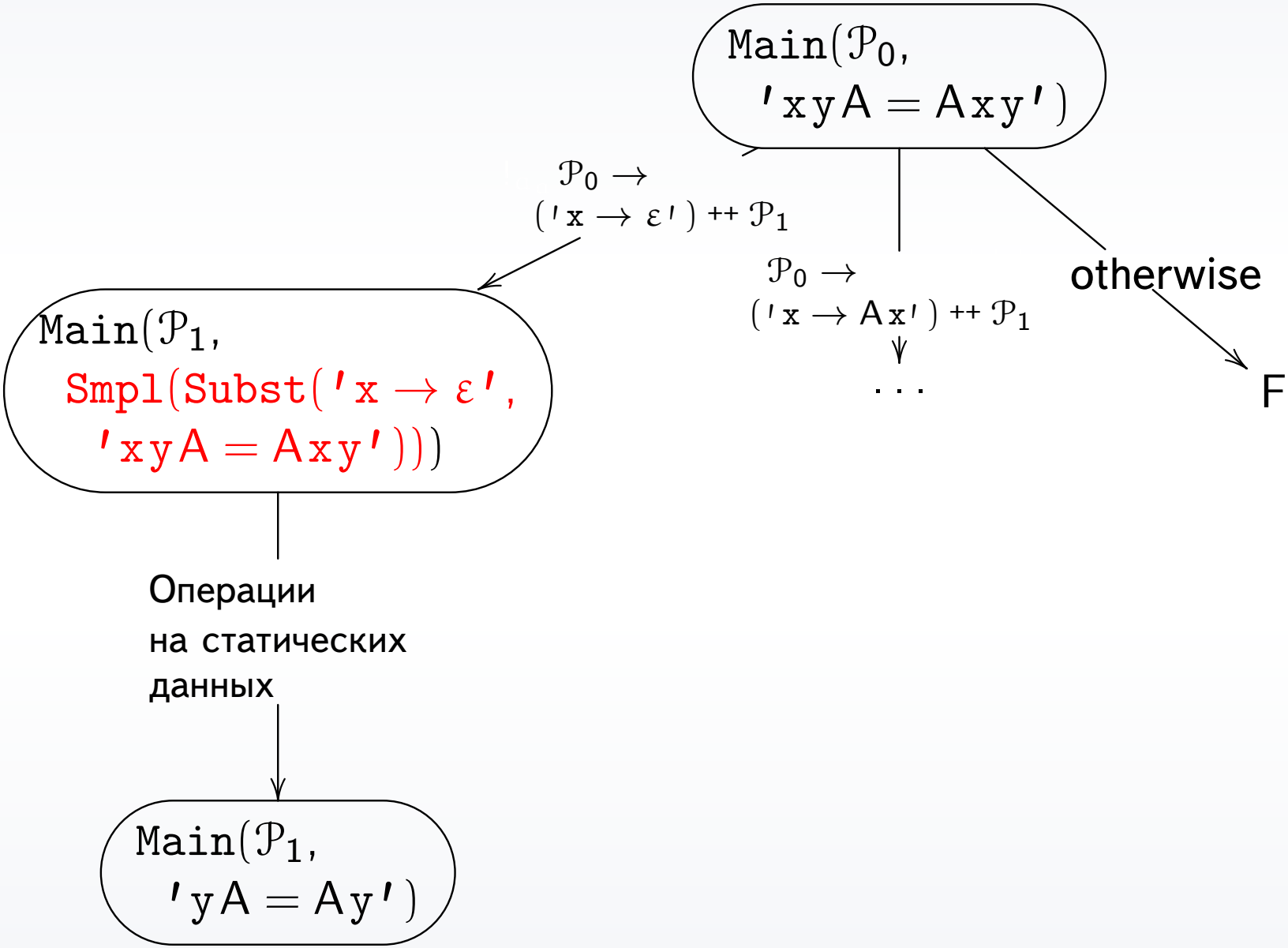
Порождение сужений



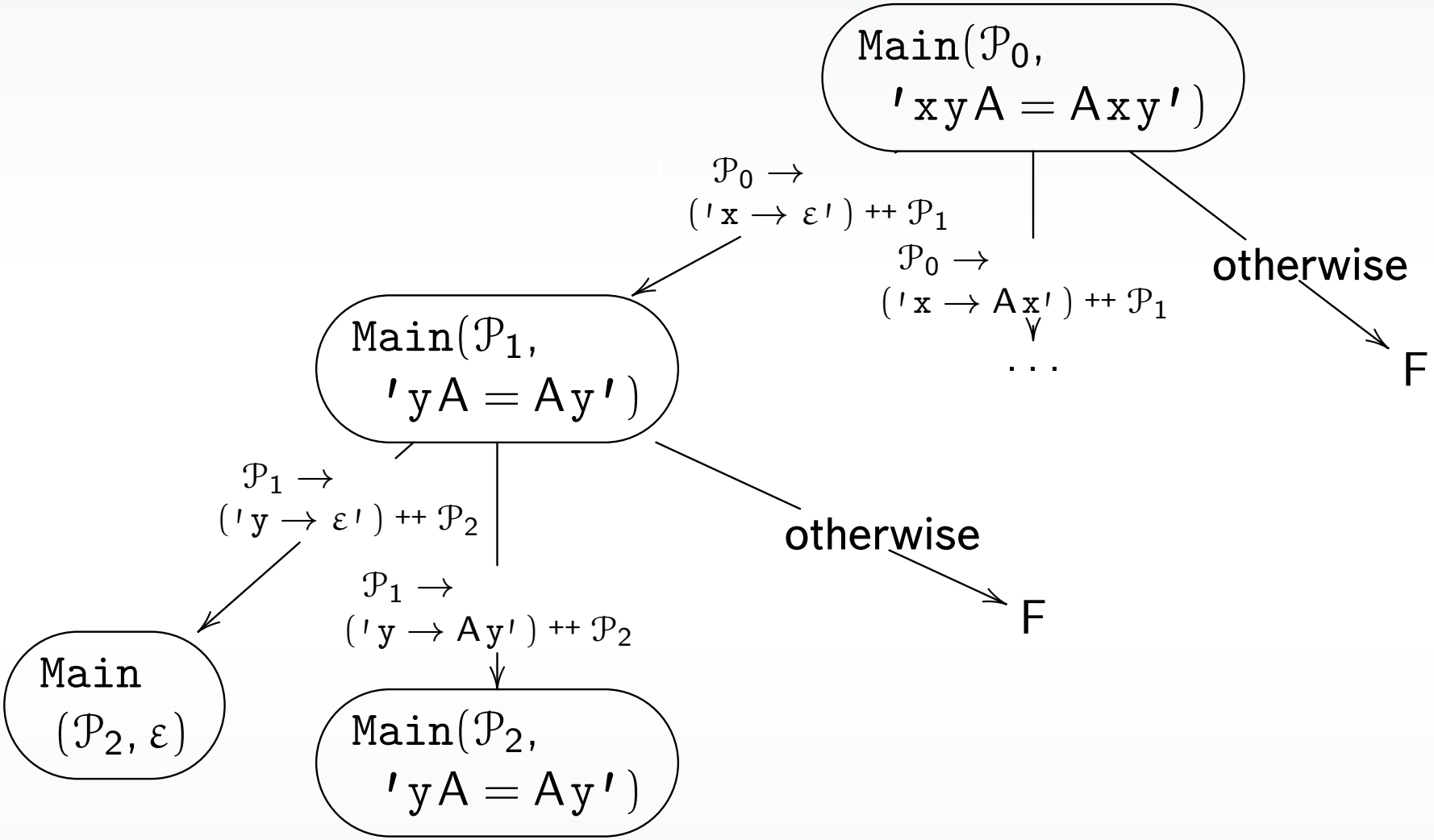
Порождение нового узла



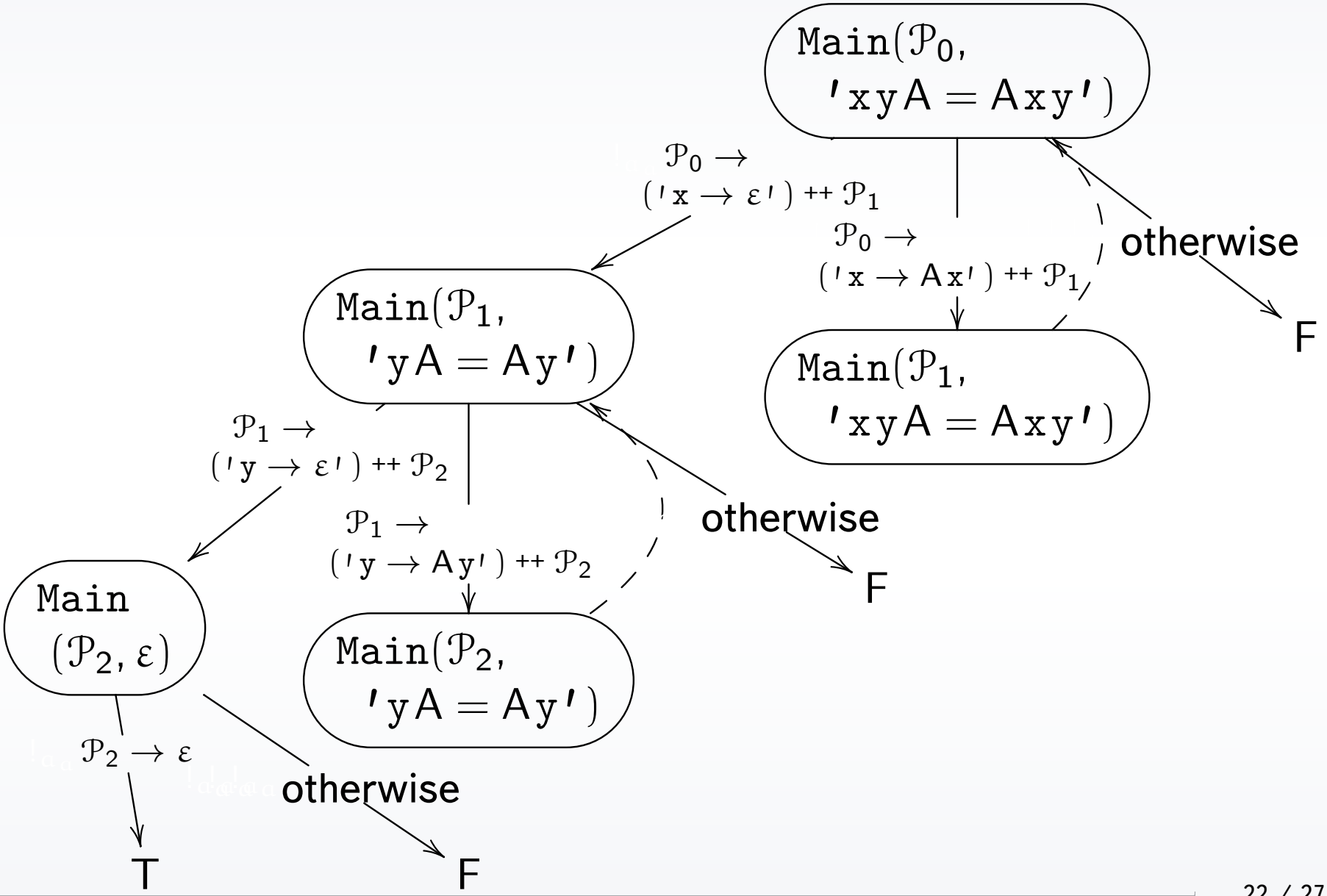
Транзитные операции



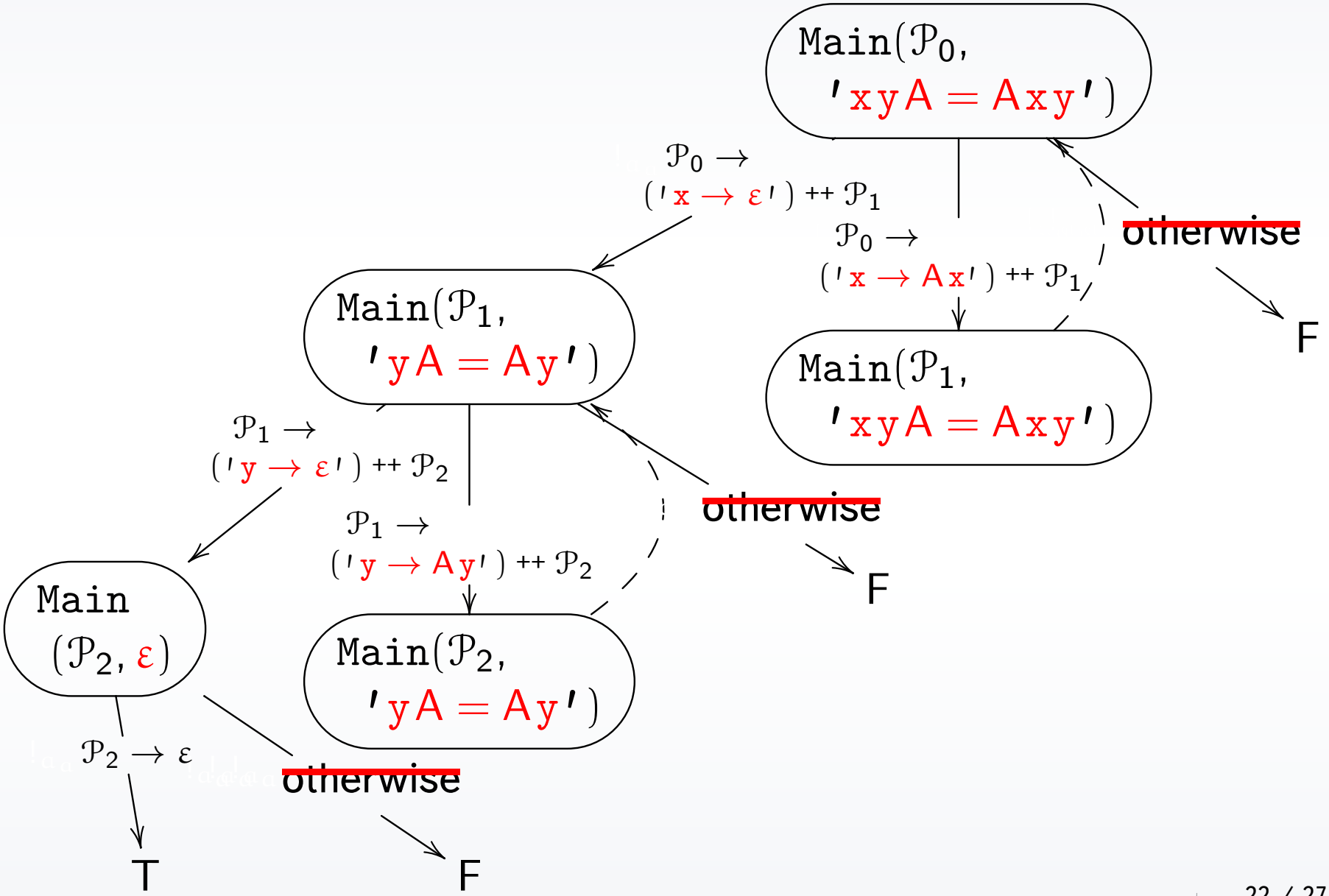
Новый шаг развертки



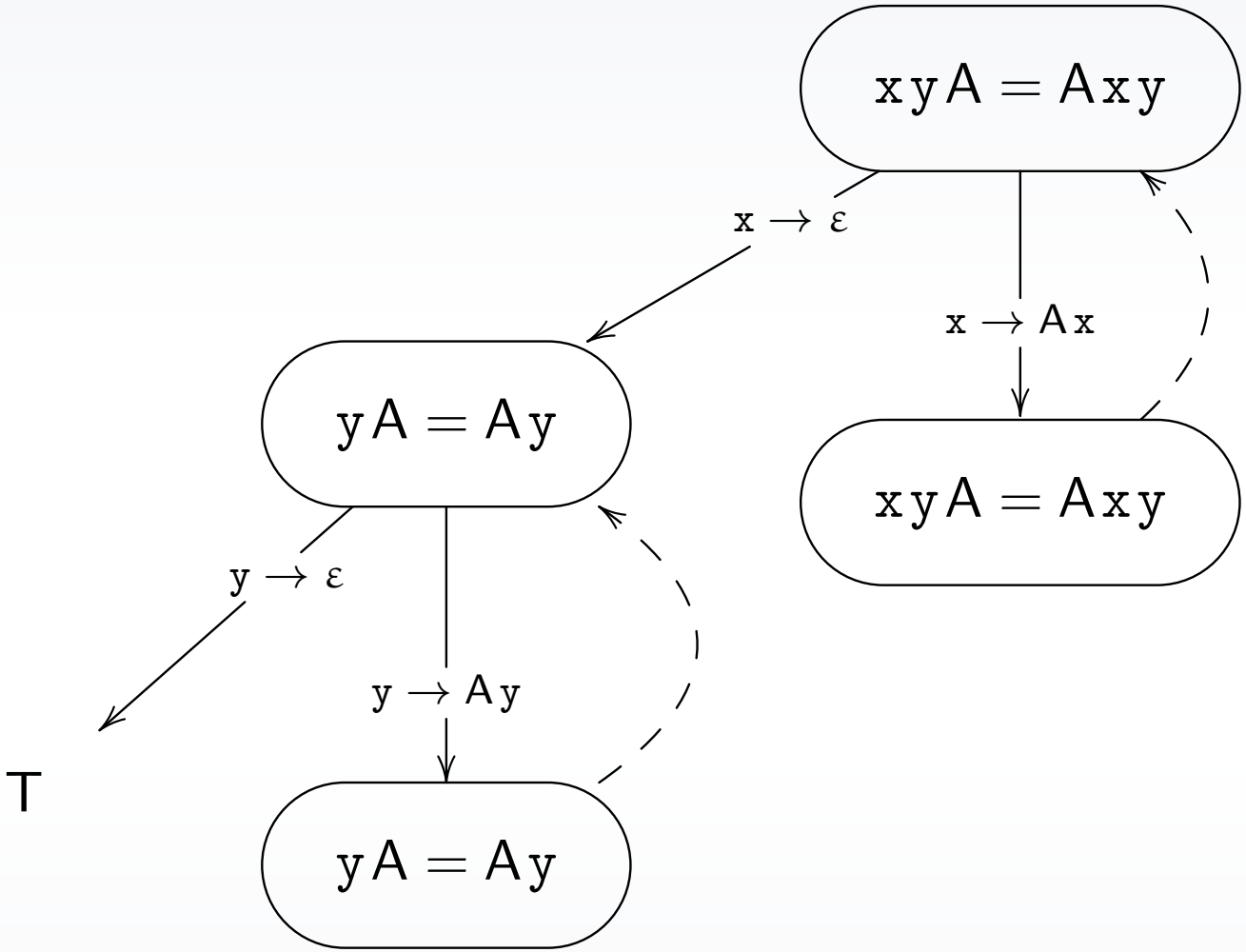
Свертка

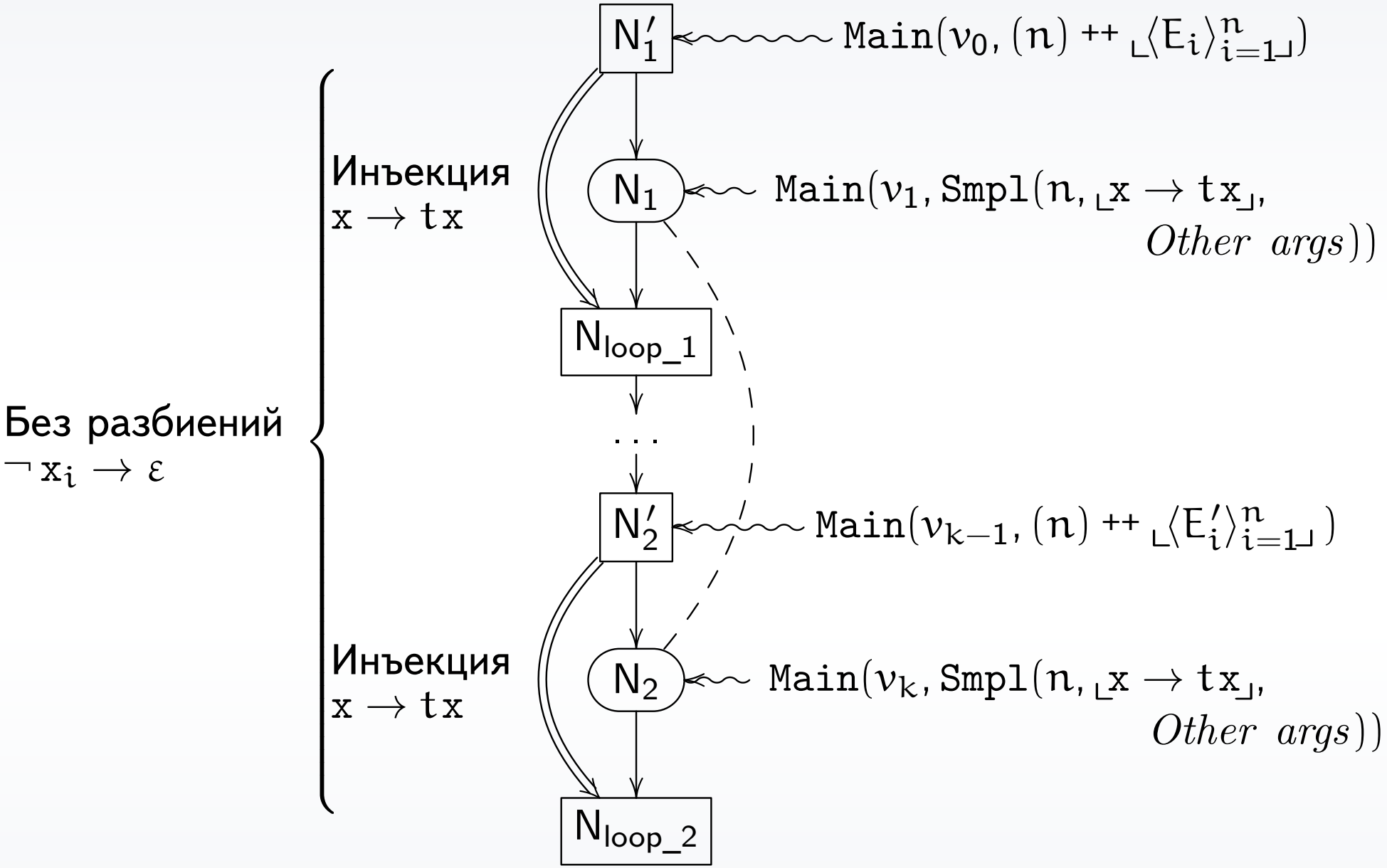

Main(P0, 'xyA = Axy')
P0 → ('x → ε') ++ P1
P0 → ('x → Ax') ++ P1
otherwise
F
Main(P1, 'yA = Ay')
Main(P1, 'xyA = Axy')
P1 → ('y → ε') ++ P2
P1 → ('y → Ay') ++ P2
otherwise
F
Main (P2, ε)
Main(P2, 'yA = Ay')
P2 → ε
otherwise
T
F

Удаление данных интерпретатора



Граф решений




Без разбиений
¬ x_i → ε
Инъекция
x → tx
Инъекция
x → tx
N'_1
Main(v_0, (n) ++ ⌊⟨E_i⟩_{i=1}^n⌋)
N_1
Main(v_1, Smpl(n, ⌊x → tx⌋, Other args))
N_loop_1
...
N'_2
Main(v_{k−1}, (n) ++ ⌊⟨E'_i⟩_{i=1}^n⌋)
N_2
Main(v_k, Smpl(n, ⌊x → tx⌋, Other args))
N_loop_2

О дополнительной разметке

- В интерпретаторе $\mathsf{WIBase}_{\mathscr{L}}$ помним только применяемую последней подстановку;
- В интерпретаторах $\mathsf{WISplit}_{\mathscr{L}}$ и $\mathsf{WICount}_{\mathscr{L}}$ — еще и количество уравнений в последней базовой конфигурации.

Структура интерпретаторов

Если преобразования не инъективны, понадобится (очередная) дополнительная разметка.

Пример

Допустим, мы умеем не только расщеплять уравнения, но и удалять дубли. Пусть после шага $\mathrm{x} \to \mathrm{Ax}$ получилась система с двумя уравнениями:

$$\mathrm{xAAy} = \mathrm{yAxA}$$
$$\mathrm{Ax} = \mathrm{xA}$$

Что можно сказать о системе, предшествующей ей, если известно, что в ней было два уравнения?

Основные особенности решения

- Для свертки было использовано отношение переименовки, чтобы избежать привязки к особенностям суперкомпилятора.
- Структура интерпретаторов подобрана таким образом, чтобы можно было использовать списочные стуктуры вместо рефальских.

Рекламная пауза

А если мы используем SCP4, можно просто написать в заголовке так:

```
*$Transient No;
```

(или не писать ничего!)

...и никакой дополнительной разметки не понадобится!

Спасибо!