Применение композиционных параметров в суперкомпиляции

А. В. Коновалов

МГТУ имени Н. Э. Баумана

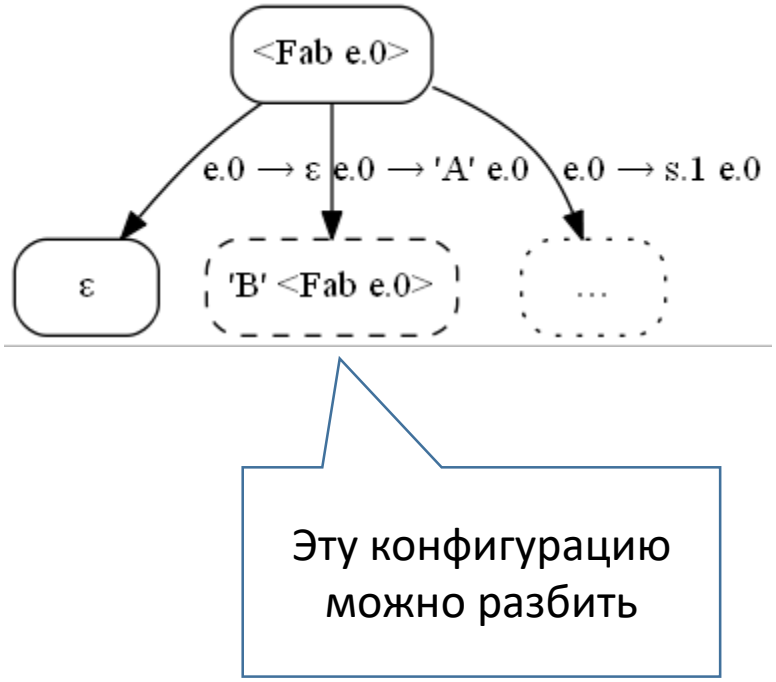
Второе совместное рабочее совещание
ИПС имени А. К. Айламазяна РАН и МГТУ имени Н. Э. Баумана
по функциональному языку программирования Рефал

11 июня 2019 года

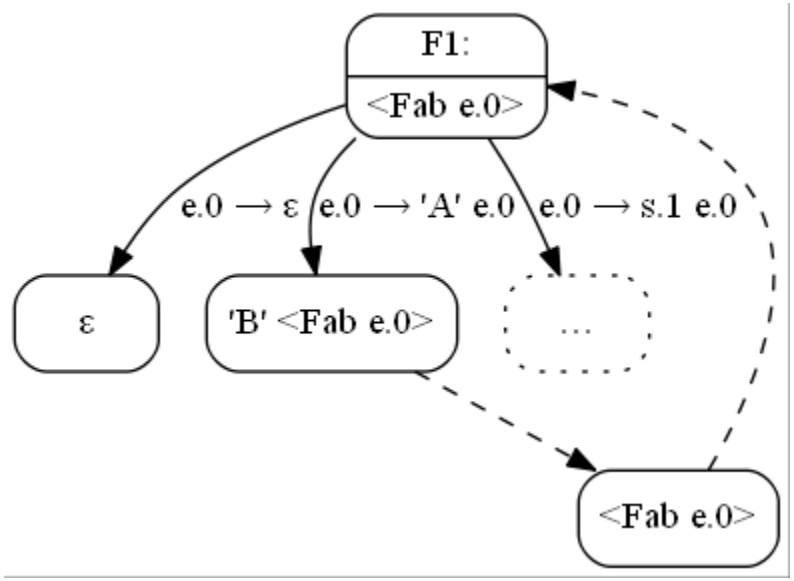
Наблюдение: некоторые программы на Рефале можно суперкомпилировать двумя способами

```
$ENTRY Go {
  e.X = <Fab e.X>;
}

Fab {
  ε = ε;
  'A' e.X = 'B' <Fab e.X>;
  s.1 e.X = s.1 <Fab e.X>;
}
```

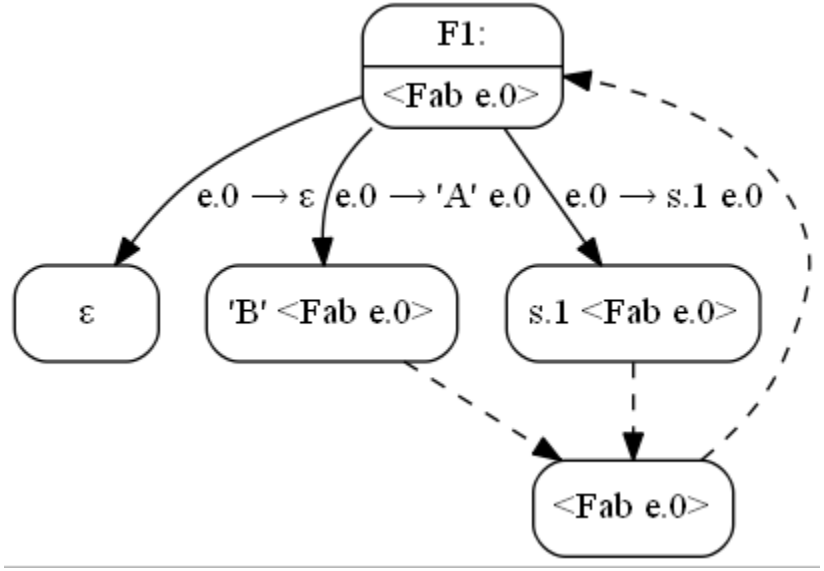
Наблюдение: некоторые программы на Рефале можно суперкомпилировать двумя способами



Наблюдение: некоторые программы на Рефале можно суперкомпилировать двумя способами



Наблюдение: некоторые программы на Рефале можно суперкомпилировать двумя способами

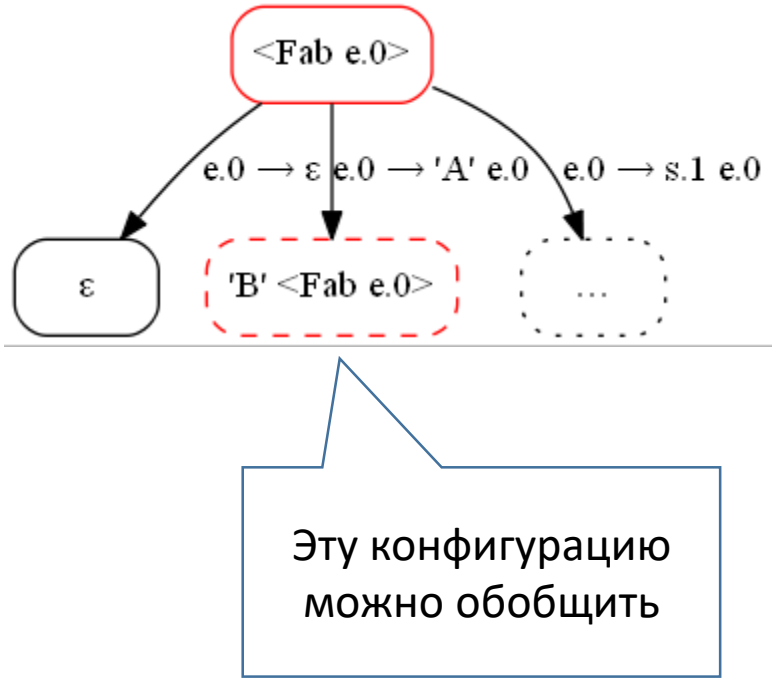


Наблюдение: некоторые программы на Рефале можно суперкомпилировать двумя способами

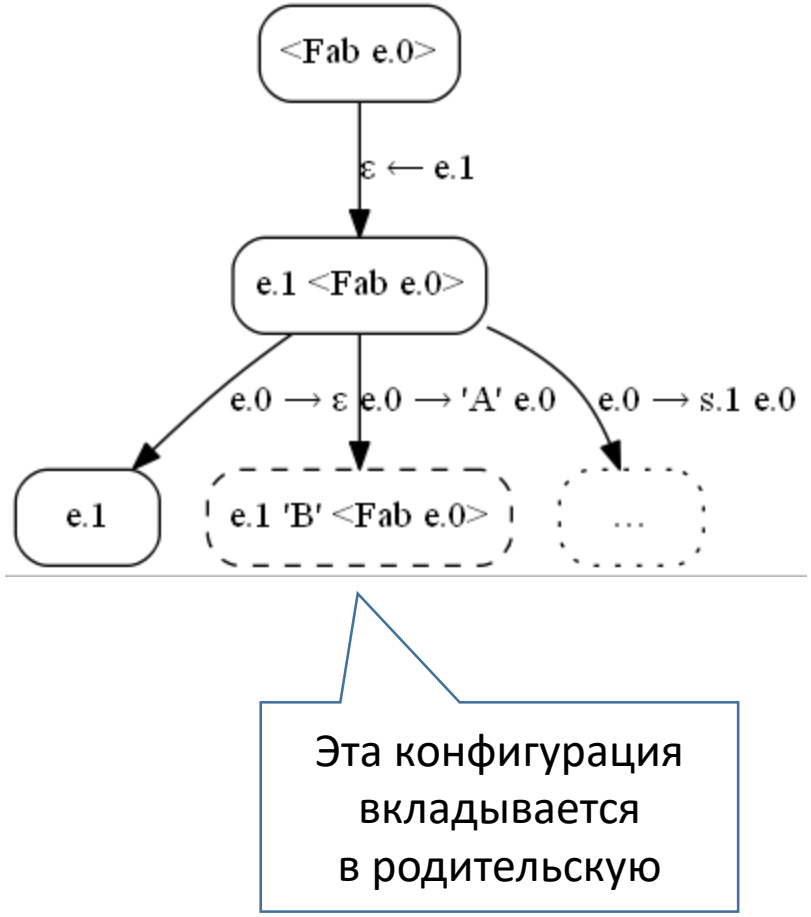
- Остаточная программа, очевидно, эквивалентна исходной:

```
$ENTRY Go {
  e.0 = <F1 e.0>;
}

F1 {
  ε = ε;
  'A' e.0 = 'B' <F1 e.0>;
  s.1 e.0 = s.1 <F1 e.0>;
}
```

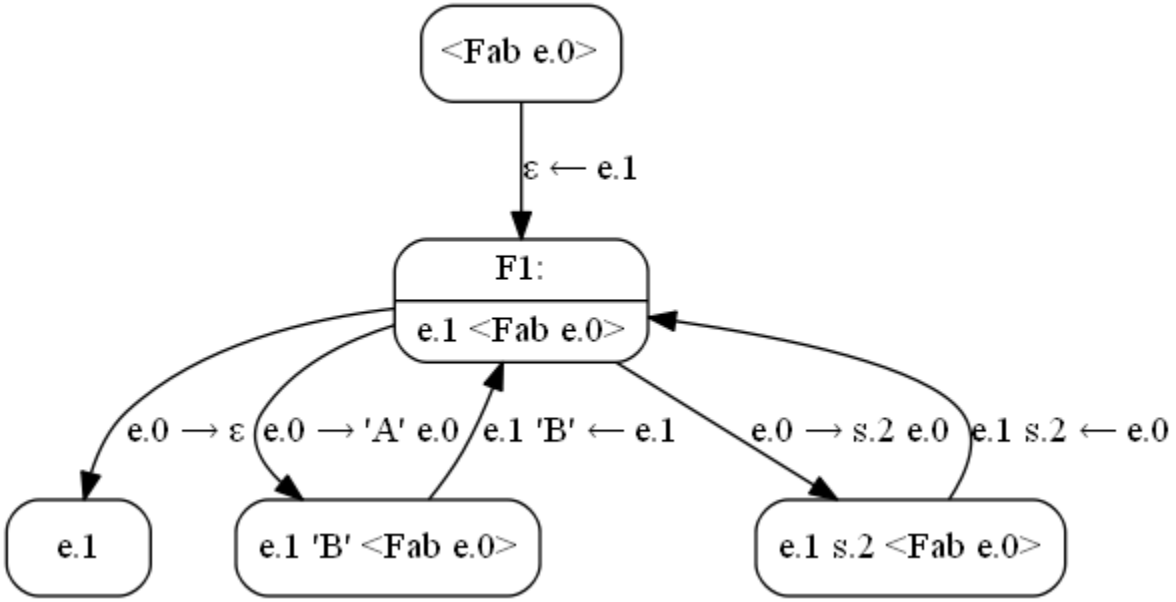
Наблюдение: некоторые программы на Рефале можно суперкомпилировать двумя способами



Наблюдение: некоторые программы на Рефале можно суперкомпилировать двумя способами



Наблюдение: некоторые программы на Рефале можно суперкомпилировать двумя способами



Наблюдение: некоторые программы на Рефале можно суперкомпилировать двумя способами

- Остаточная программа уже будет иной:

```
$ENTRY Go {
  e.0 = <F1 ε, e.0>;
}

F1 {
  e.1, ε = e.1;
  e.1, 'A' e.0 = <F1 e.1 'B', e.0>;
  e.1, s.2 e.0 = <F1 e.1 s.2, e.0>;
}
```

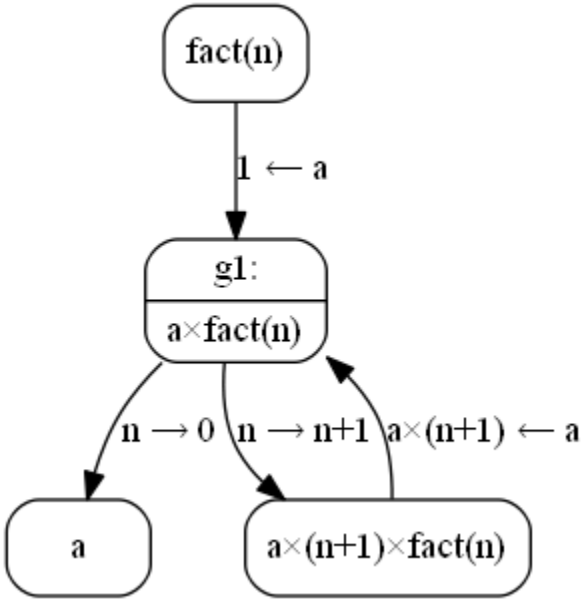
Наблюдение: некоторые программы на Рефале можно суперкомпилировать двумя способами

- Почему так получилось?
- Потому что конкатенация является моноидом: ассоциативной операцией с нейтральным элементом.
- Допустим, у конфигурации вида f(x) появился потомок f(x)•y, где «•» — ассоциативная операция.
- Тогда дочернюю можно или разбить, или обобщить с родительской.

Ещё пример на моноид: факториал

```
fact(n)

where
fact(0) = 1;
fact(n+1) = (n+1)×fact(n);
```

Ещё пример на моноид: факториал



Ещё пример на моноид: факториал

- Остаточная программа:

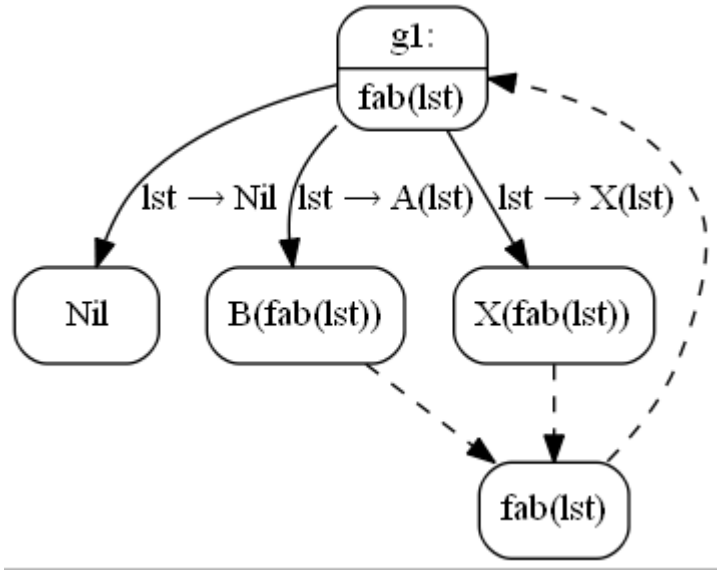
```
g1(1, n)

where
g1(a, 0) = a;
g1(a, n+1) = g1(a×(n+1), n);
```

Можно ли эту программу суперкомпилировать двумя способами?

```
fab(lst)

where
fab(Nil) = Nil;
fab(A(lst)) = B(fab(lst));
fab(X(lst)) = X(fab(lst));
```

Можно ли эту программу суперкомпилировать двумя способами?

- Один способ очевиден:



- Остаточная программа будет эквивалентна исходной.

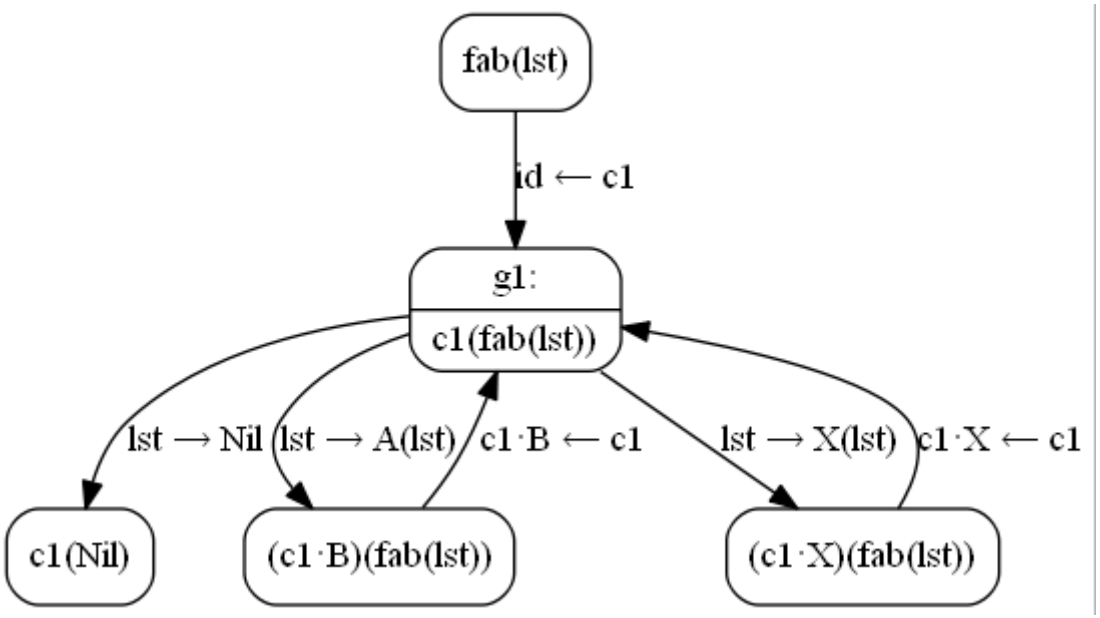
Можно ли эту программу суперкомпилировать двумя способами?

- А есть ли другой способ?
- Можно ли *обобщить* конфигурации fab(lst) и B(fab(lst))?

Можно ли эту программу суперкомпилировать двумя способами?

- А есть ли другой способ?
- Можно ли *обобщить* конфигурации fab(lst) и B(fab(lst))?
- В первом порядке нельзя. В высшем — можно.
- Их обобщение в высшем порядке: c1(fab(lst)), сводящие подстановки:
  - λx.x ← c1
  - B ← c1
- Композиция функций $(f \cdot g)(x) \equiv f(g(x))$ (при $f, g : D \to D$) является моноидом: она ассоциативна и имеет нейтральный элемент $id \equiv \lambda x.x$.
- Попробуем суперкомпилировать и посмотрим, что получится.

Можно ли эту программу суперкомпилировать двумя способами?

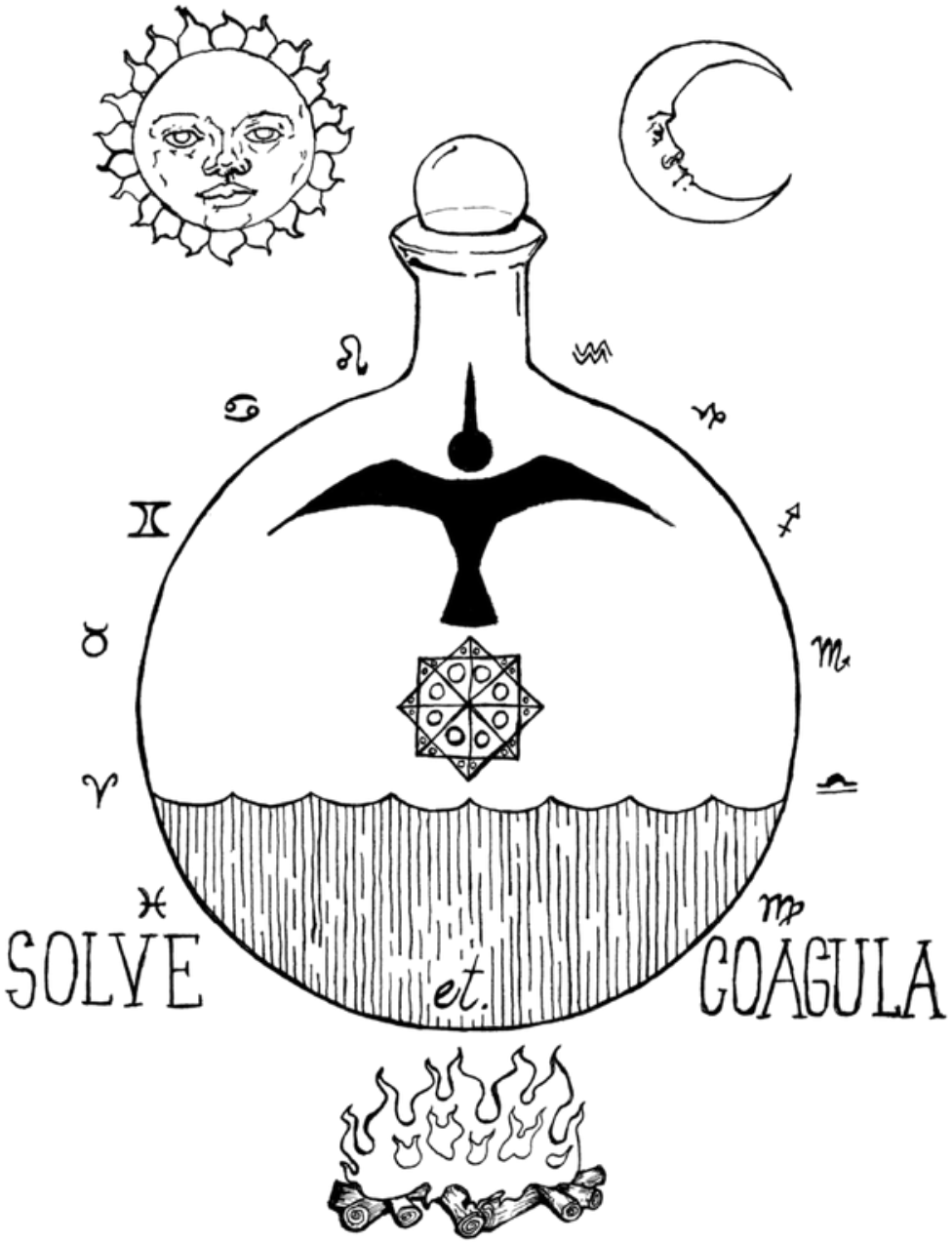


Можно ли эту программу суперкомпилировать двумя способами?

- Остаточная программа будет иметь вид

```
g1(id, lst)
where

g1(c1, Nil) = c1(Nil);
g1(c1, A(lst)) = g1(c1·B, lst);
g1(c1, X(lst)) = g1(c1·X, lst);
```

- Мы перешли в высший порядок. Мы получили хвостовую рекурсию. Но не очевидно, решение лучше или хуже исходного.
- А можно ли как-нибудь вернуться из высшего порядка в первый?



Источник картинки: https://closetofmysteries.com/tag/solve-et-coagula/

Расширение суперкомпиляции

- Предлагаемый метод работает только с примитивной рекурсией — когда существует аргумент, который на каждом рекурсивном вызове уменьшается.
- Метод работает только в довольно узких рамках — при выходе за рамки следует отбросить построенный граф и суперкомпилировать «обычным» способом.

Расширение суперкомпиляции

- Суперкомпиляция выполняется в три этапа, при выполнении которых мы переходим в высший порядок и возвращаемся обратно:

1. **Сгущение** (coagula) — при построении графа используем только обобщения с композиционными параметрами. Граф может получиться не самодостаточный.
2. **Выворачивание** — меняем направления обратных стрелок.
3. **Растворение** (solve) — прогоняем точки выхода из рекурсии, используя вывернутый граф. В остальном выполняем суперкомпиляцию по обычным правилам.

Расширение суперкомпиляции

- Рассмотрим метод на примере (не особо содержательном, интересные будут дальше):

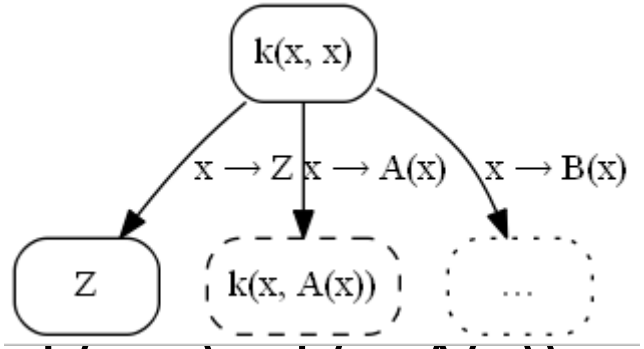
```
k(x, x)

where
k(Z, y) = y;
k(A(x), y) = k(x, y);
k(B(x), y) = k(x, y);
```

Расширение суперкомпиляции

- На стадии *сгущения* (coagula) можно выполнять только обобщения конфигураций, причём только с использованием композиционных параметров.
- Если композиционный параметр препятствует дальнейшим вычислениям, объявляем конфигурацию одним из выходов из рекурсии.
- Построенный граф, будем называть его *графом коагуляции*, может получиться не самодостаточным. Это нормально, отложенные вызовы вычислятся на третьей стадии.
- В результате должен получиться граф, в выходах из рекурсии которого не осталось параметров, по которым велась прогонка. Иначе метод не применим.

Расширение суперкомпиляции

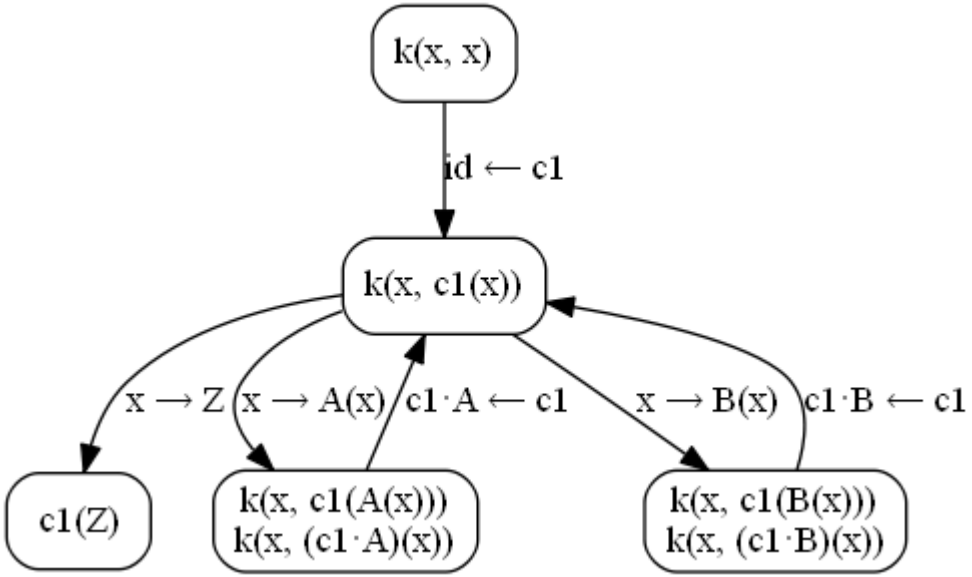
- Стадия сгущения:



- Конфигурации k(x, x) и k(x, A(x)) похожи по Хигману-Крускалу. Их следует обобщить с использованием композиционного парамера.
- Их обобщение: k(x, c1(x)), сводящие подстановки id ← c1 и A ← c1.

Расширение суперкомпиляции

- Стадия сгущения:

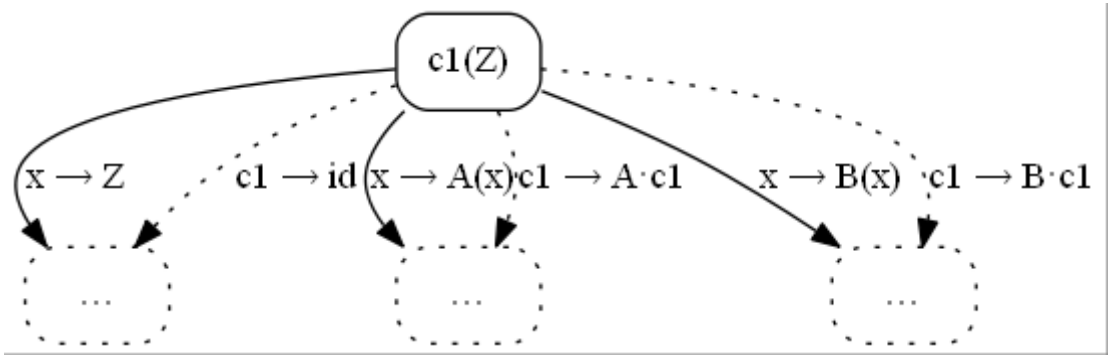


Расширение суперкомпиляции

- На стадии **выворачивания** меняем направления обратных стрелок.
- Присваивания на этих стрелках зеркально отражаем:
  `c1·A ← c1`
  меняется на
  `c1 → A·c1`
- Для ветки выхода из рекурсии рисуем дополнительную дугу, помеченную c1 → id.
- В корень графа помещаем выражение выхода из рекурсии (или кортеж выражений).

Расширение суперкомпиляции

- Вывернутый граф:

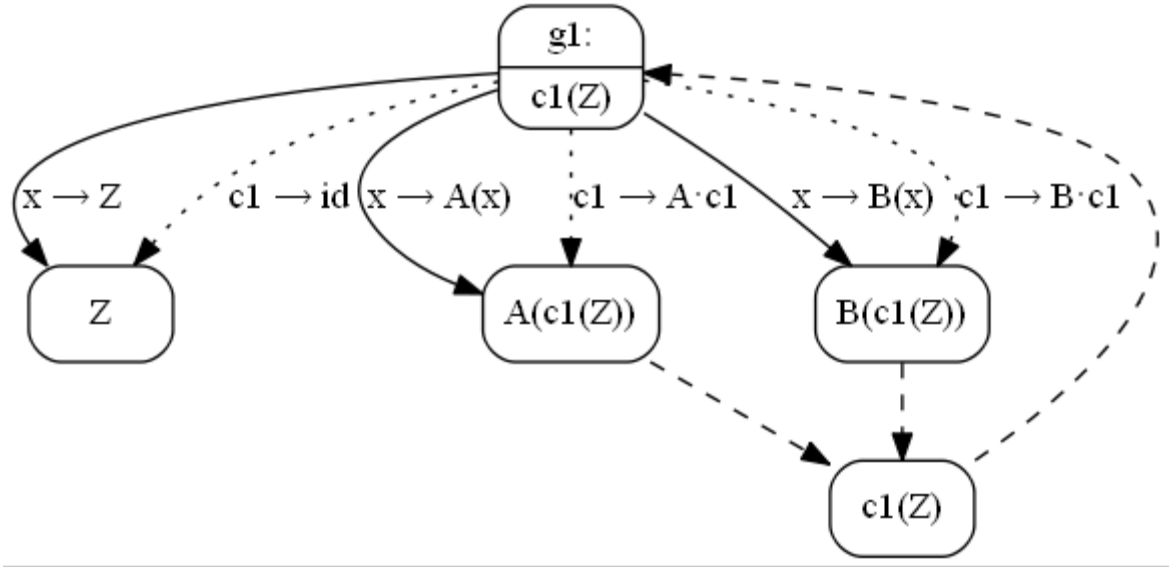


Расширение суперкомпиляции

- На стадии **растворения** подразумеваем прогонку по исходным парамерам, но осуществляем прогонку по псевдосужениям композиционных параметров.
- В остальном суперкомпилируем как обычно:
  - когда видим конструктор снаружи, выполняем декомпозицию,
  - когда конфигурации похожи по отношению Хигмана-Крускала, обобщаем обычным образом.
- В результате композиционные параметры должны *раствориться,* остаточная программа должна вернуться в первый порядок.

Расширение суперкомпиляции

- Стадия растворения:



Расширение суперкомпиляции

- Остаточная программа:

```
g1(x)

where
g1(Z) = Z;
g1(A(x)) = A(g1(x));
g1(B(x)) = B(g1(x));
```

- Остаточная программа уже не похожа на исходную.
- Местность функции понизилась, аккумулирующий параметр исчез.

Уравнения в словах

- Уравнение в словах — это запись некоторого равенства, слева и справа в котором находятся конкатенации либо константных, либо неизвестных строк.
- Пример:
  x 'abra' y = 'abracadabra'
- Два решения:
  $x_1 = \varepsilon$, y_1 = 'cadabra'
  x_2 = 'abracad', $y_2 = \varepsilon$

Уравнения в словах

- Нас будут интересовать уравнения вида

  x A = A x

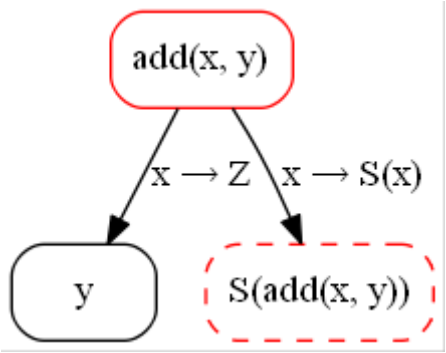
- Множество его решений можно описать в виде x = A^*, т.е. повторение A ноль или более раз.
- Мы их будем использовать в качестве рестрикций для композиционных параметров.

Пример на рестрикции в уравнениях в словах

```
add(x, y)

where
add(Z, y) = y;
add(S(x), y) = S(add(x, y));
```

Пример на рестрикции в уравнениях в словах

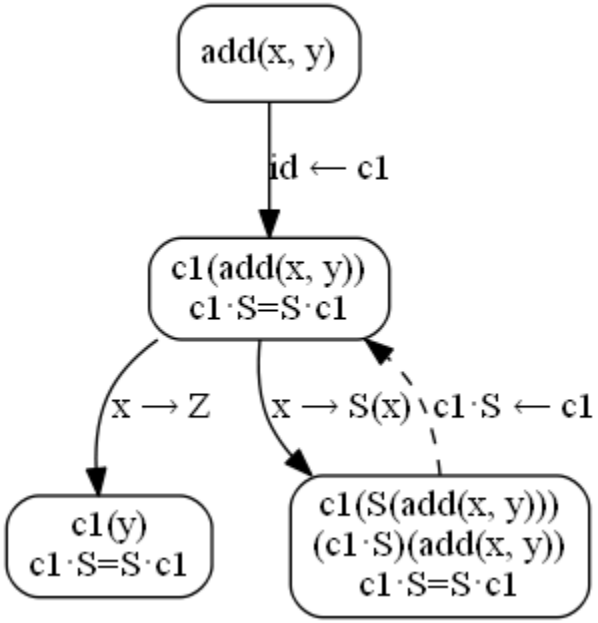
- Сгущаем:



- Обобщаем как c1(add(x, y)), сводящие подстановки id ← c1 и S ← c1.
- Применима рестрикция c1·S = S·c1

Пример на рестрикции в уравнениях в словах

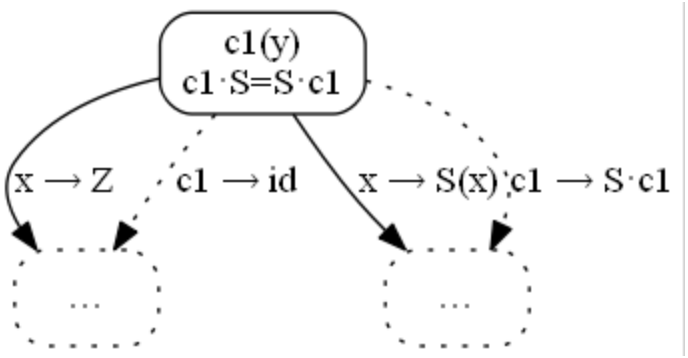
- Сгущаем:



- Подстановка c1·S ← c1 сохраняет рестрикцию:
  (c1·S)·S = S·(c1·S)
  c1·S·S = S·c1·S
  c1·S = S·c1

Пример на рестрикции в уравнениях в словах

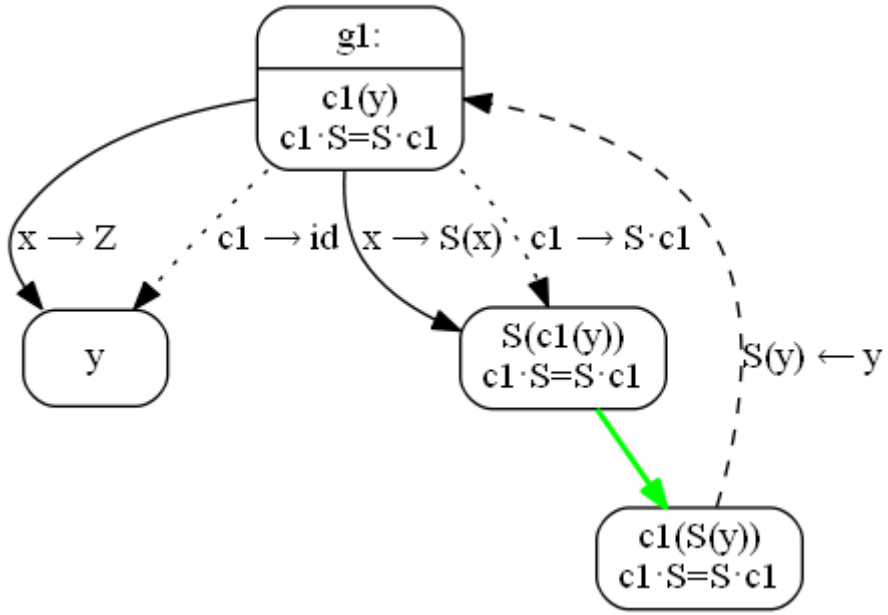
- Выворачиваем:



- Рестрикцию сохраняем в точке выхода из рекурсии. Она нам ещё понадобится.

Пример на рестрикции в уравнениях в словах

- Растворяем:



- Зелёной стрелкой показано применение рестрикции.

Пример на рестрикции в уравнениях в словах

- Остаточная программа:

```
g1(x, y)

where
g1(Z, y) = y;
g1(S(x), y) = g1(x, S(y))
```

- Рестрикция позволила преобразовать рекурсию в итерацию: остаточная программа использует хвостовую рекурсию и аккумулятор.
- Можно делать и обратные преобразования.

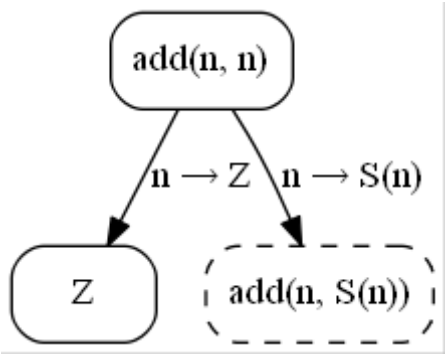
Пример: вычислим n+n

- Следующий пример демонстрирует уже чуть более интересное преобразование.

```
add(n, n)

where
add(Z, y) = y;
add(S(x), y) = S(add(x, y));
```

Пример: вычислим n+n

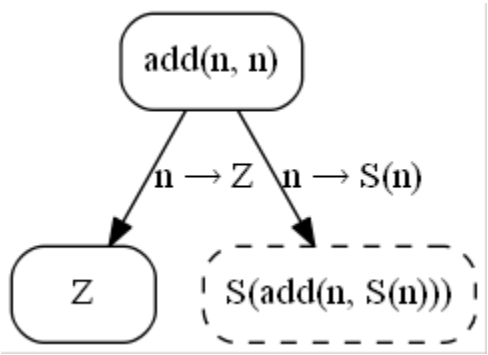
- Сгущаем:



- Две конфигурации обобщаем до add(n, c1(n))

Пример: вычислим n+n

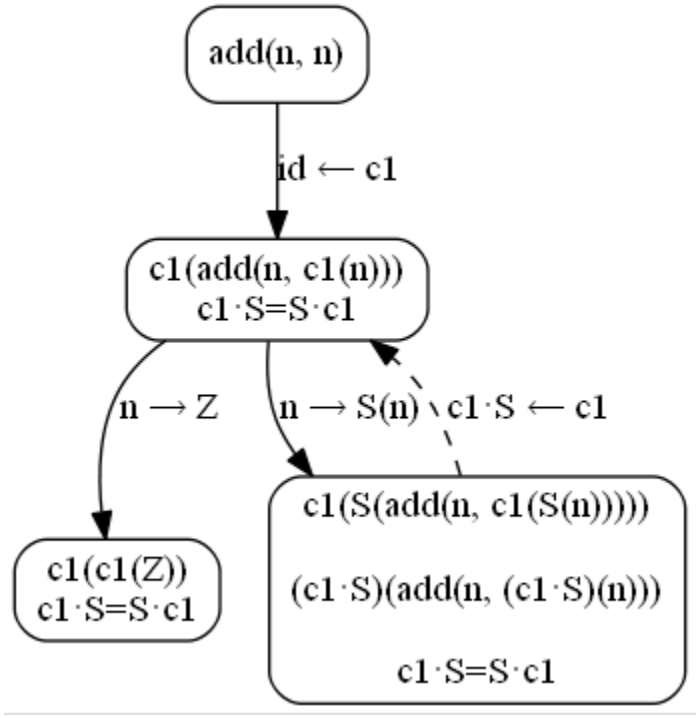
- Сгущаем:



- Две конфигурации похожи.
  Их можно обобщить до
  c1(add(n, c1(n)))
- Заметим, что для c1 применима рестрикция
  c1·S = S·c1

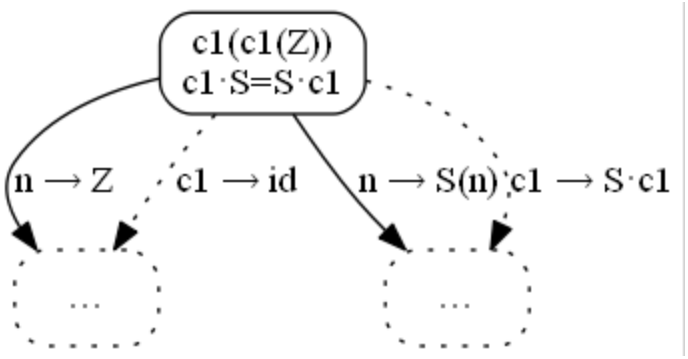
Пример: вычислим n+n

- Сгущаем:



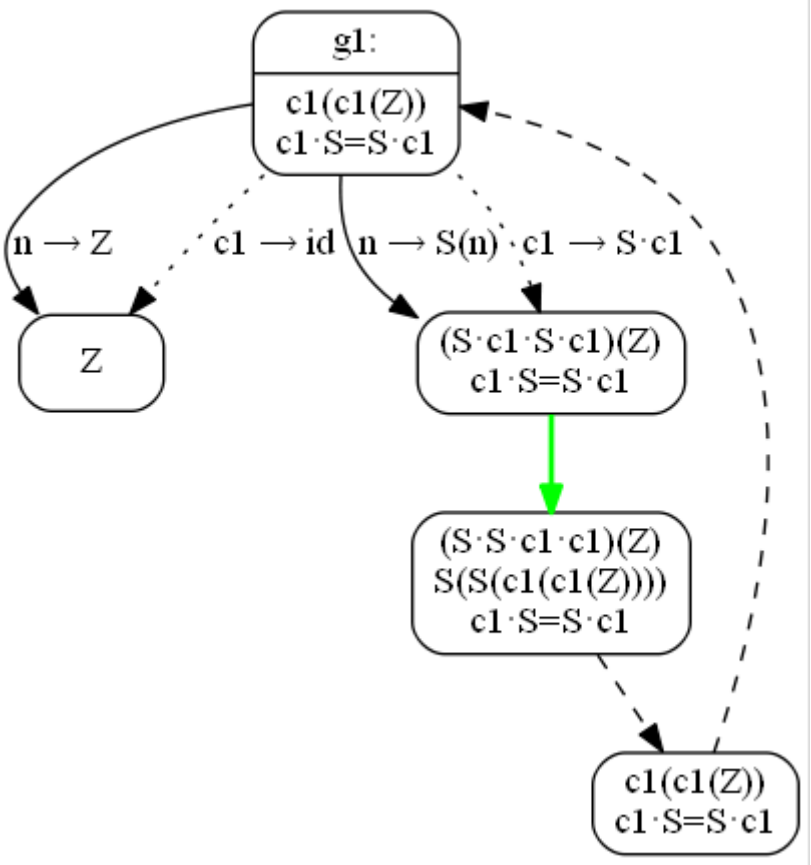
Пример: вычислим n+n

- Выворачиваем:



Пример: вычислим n+n

- Растворяем:



Пример: вычислим n+n

- Остаточная программа:

```
g1(n)

where
g1(Z) = Z;
g1(S(n)) = S(S(g1(n)));
```

- Снизилась местность среды, программа стала очевидно проще.

Верификация: x ≤ y + x

$$
\begin{array}{l}
leq\ x\ (add\ y\ x) \\
\textbf{where} \\
leq\ \ = \lambda x.\lambda y.\textbf{case}\ x\ \textbf{of} \\
\qquad\qquad\qquad\qquad 0 \qquad\ : True \\
\qquad\qquad\qquad\ \mid x' + 1\ : \textbf{case}\ y\ \textbf{of} \\
\qquad\qquad\qquad\qquad\qquad\qquad 0 \qquad\ : False \\
\qquad\qquad\qquad\qquad\qquad\ \mid y' + 1\ : leq\ x'\ y' \\
add = \lambda x.\lambda y.\textbf{case}\ x\ \textbf{of} \\
\qquad\qquad\qquad\qquad 0 \qquad\ : y \\
\qquad\qquad\qquad\ \mid x' + 1\ : (add\ x'\ y) + 1
\end{array}
$$

- Пример из статьи
  Hamilton, G.W.: Distillation: Extracting the Essence of Programs. In: Proceedings of the ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation. (2007) 61–70

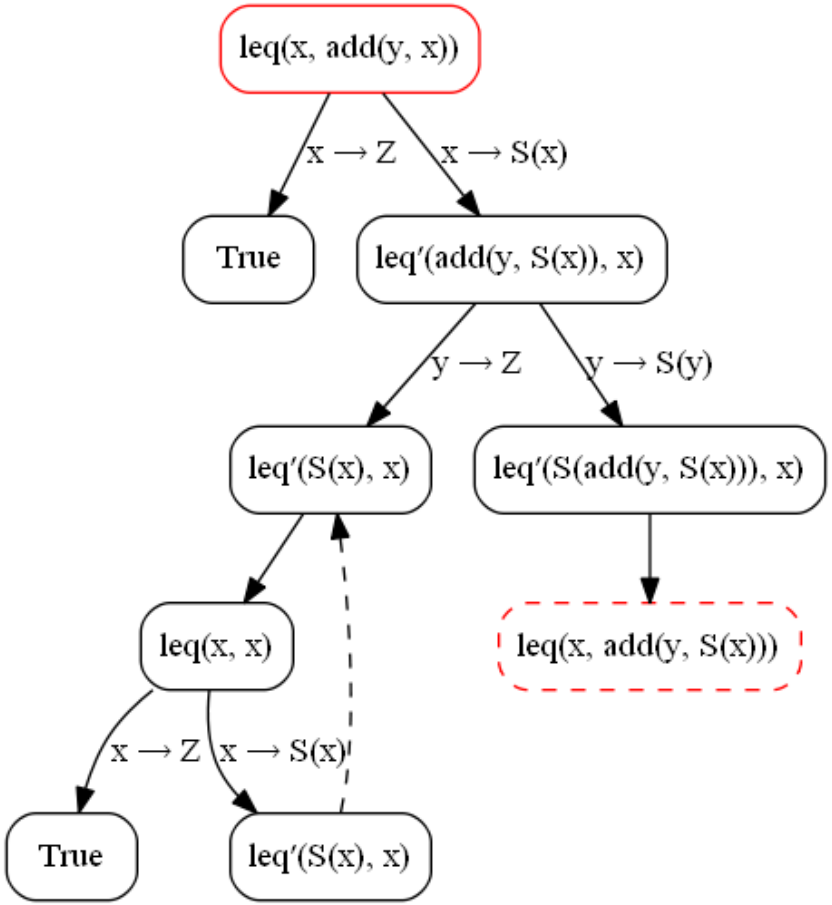
Верификация: $x \leq y + x$

```
leq(x, add(y, x))

where
leq(Z, y) = True;
leq(S(x), y) = leq'(y, x);

leq'(Z, x) = False;
leq'(S(y), x) = leq(x, y);

add(Z, y) = y;
add(S(x), y) = S(add(x, y));
```

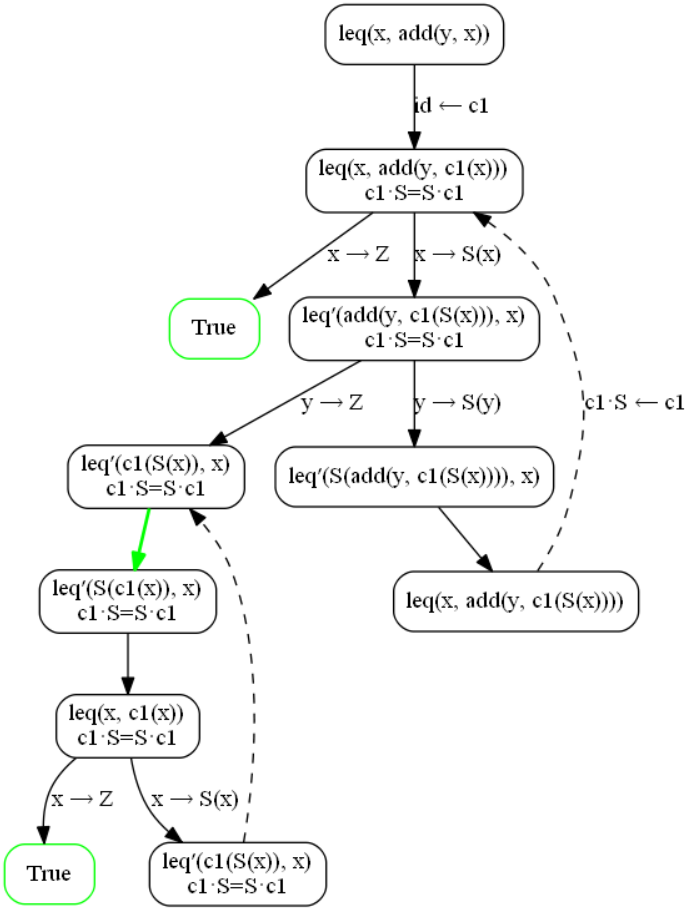
Верификация: $x \leq y + x$

```
leq(x, add(y, x))

where
leq(Z, y) = True;
leq(S(x), y) = leq'(y, x);

leq'(Z, x) = False;
leq'(S(y), x) = leq(x, y);

add(Z, y) = y;
add(S(x), y) = S(add(x, y));
```

Верификация: x ≤ y + x

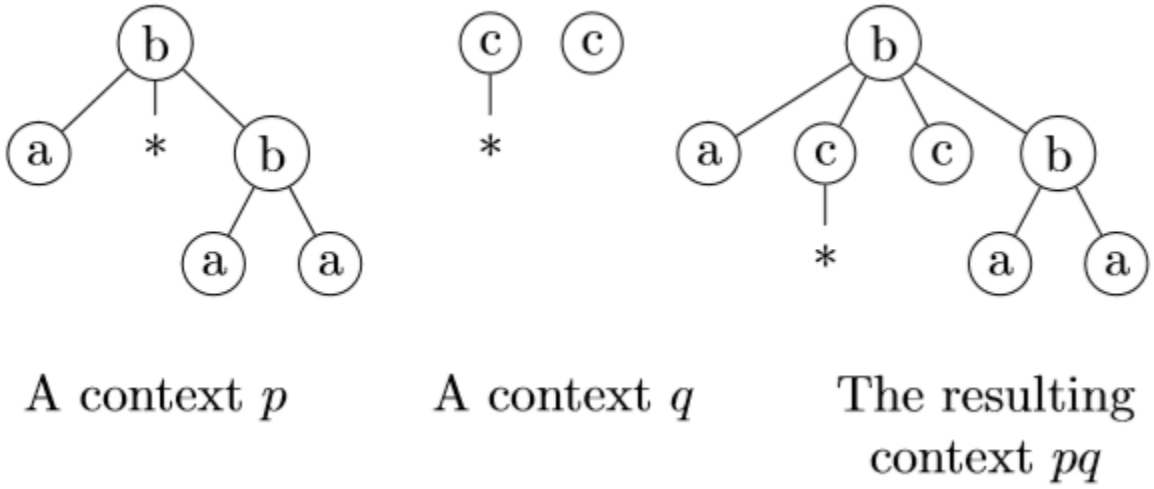


Верификация: x ≤ y + x



Композиционные параметры и Рефал

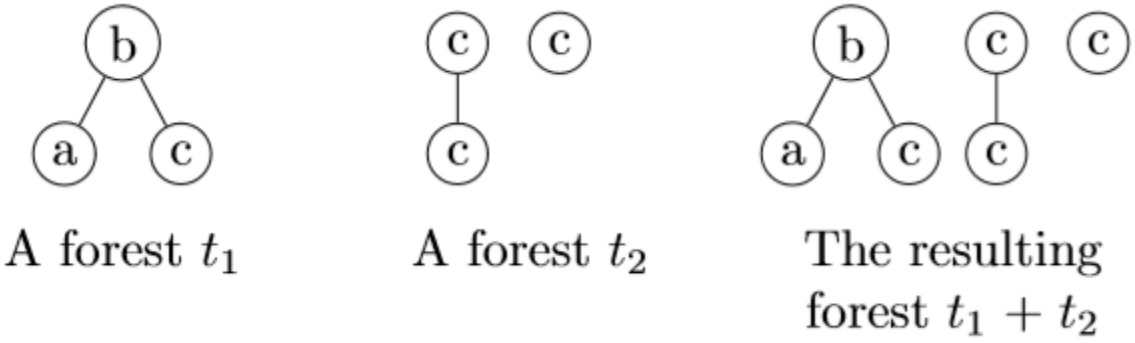
- Вертикальная композиция лесов:



Изображение из статьи
M. Bojańczyk, I. Walukiewicz. *Forest Algebras.* In: J. Flum, E. Graedel, T. Wilke eds. "Logic and Automata", Texts in Logic and Games, Amsterdam University Press, 2007

Композиционные параметры и Рефал

- Горизонтальная композиция лесов:



Изображение из статьи
M. Bojańczyk, I. Walukiewicz. *Forest Algebras.* In: J. Flum, E. Graedel, T. Wilke eds. "Logic and Automata", Texts in Logic and Games, Amsterdam University Press, 2007

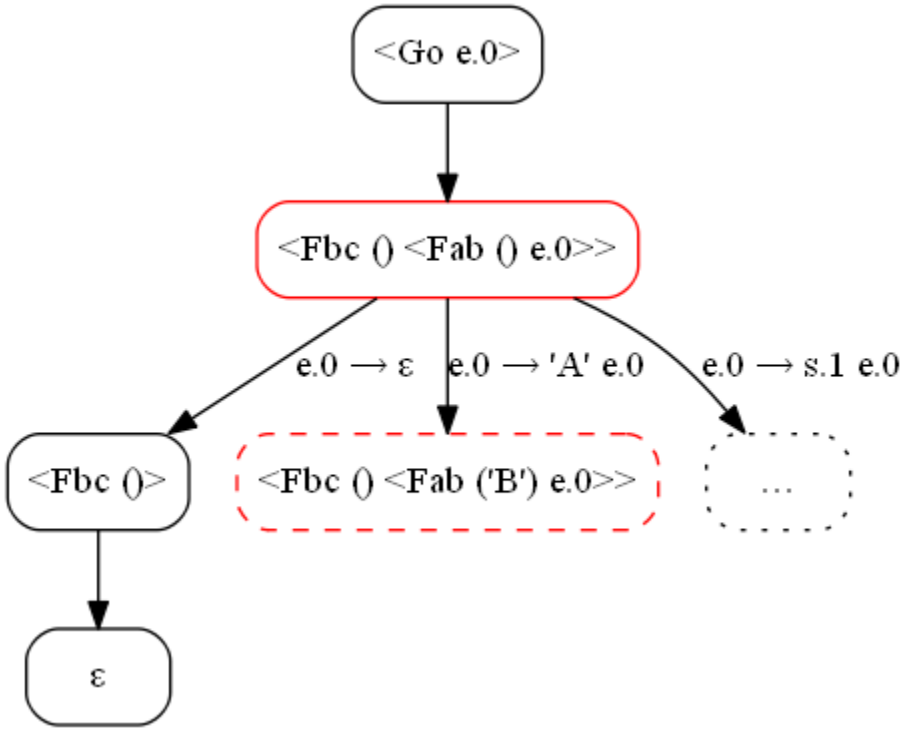
Композиционные параметры и Рефал

- В Рефале можно вводить горизонтальные композиционные параметры.
- Это почти как обычные e-переменные (вернее, e-параметры), но к ним применимы процедуры сгущения, выворачивания и растворения.
- Т.е. это e-параметры, которые запоминают свою историю.

Композиционные параметры и Рефал

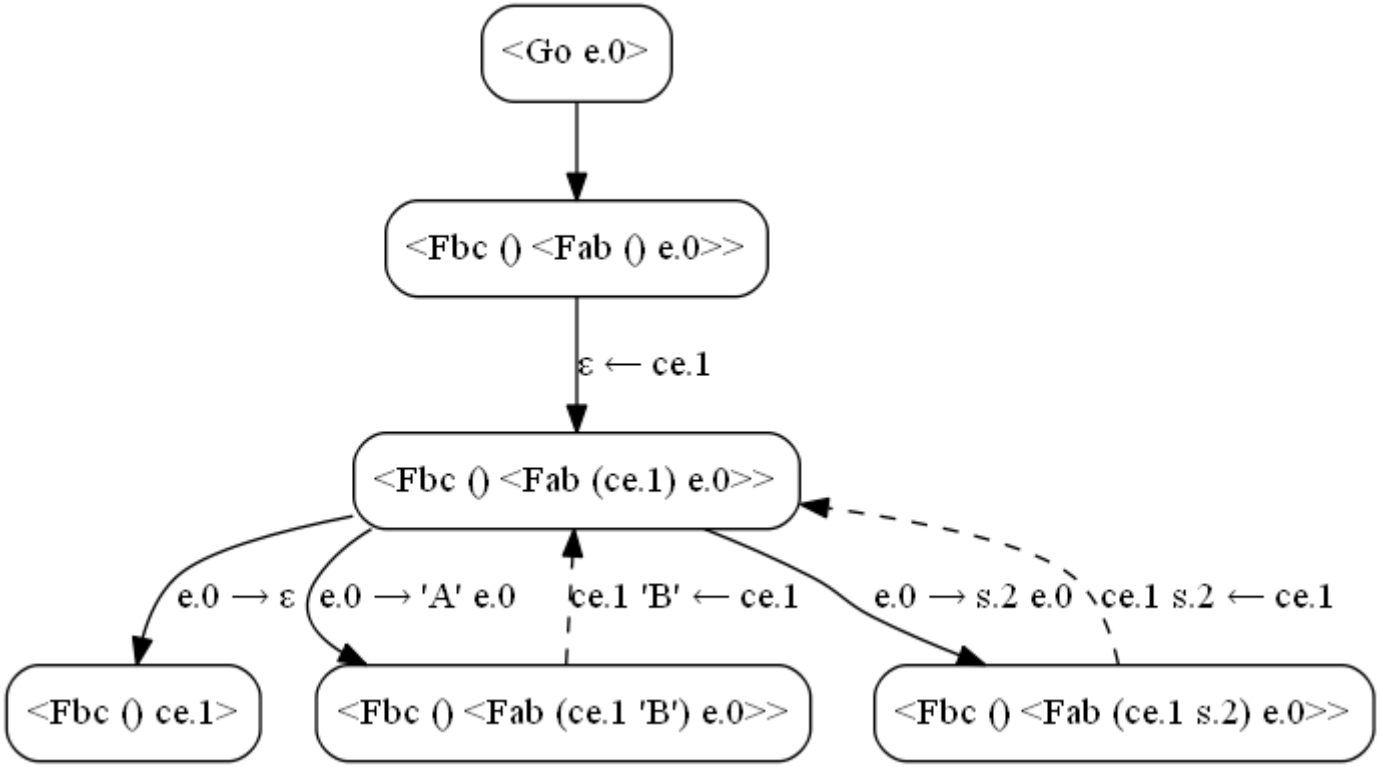
```
$ENTRY Go {
  e.0 = <Fbc () <Fab () e.0>>;
}

Fab {
  (e.Res) ε = e.Res;
  (e.Res) 'A' e.X = <Fab (e.Res 'B') e.X>;
  (e.Res) s.1 e.X = <Fab (e.Res s.1) e.X>;
}

Fbc {
  (e.Res) ε = e.Res;
  (e.Res) 'B' e.X = <Fbc (e.Res 'C') e.X>;
  (e.Res) s.1 e.X = <Fbc (e.Res s.1) e.X>;
}
```

Композиционные параметры и Рефал



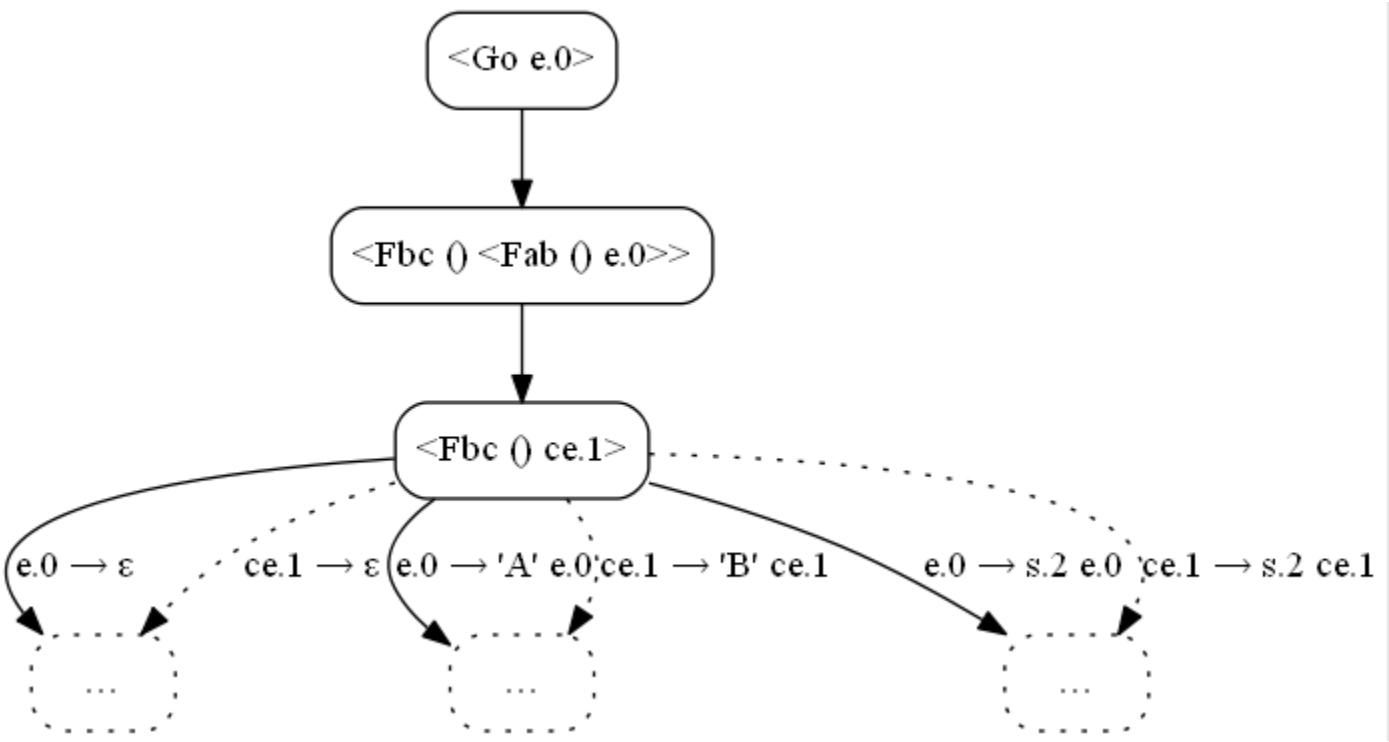
Композиционные параметры и Рефал

- Сгущаем:



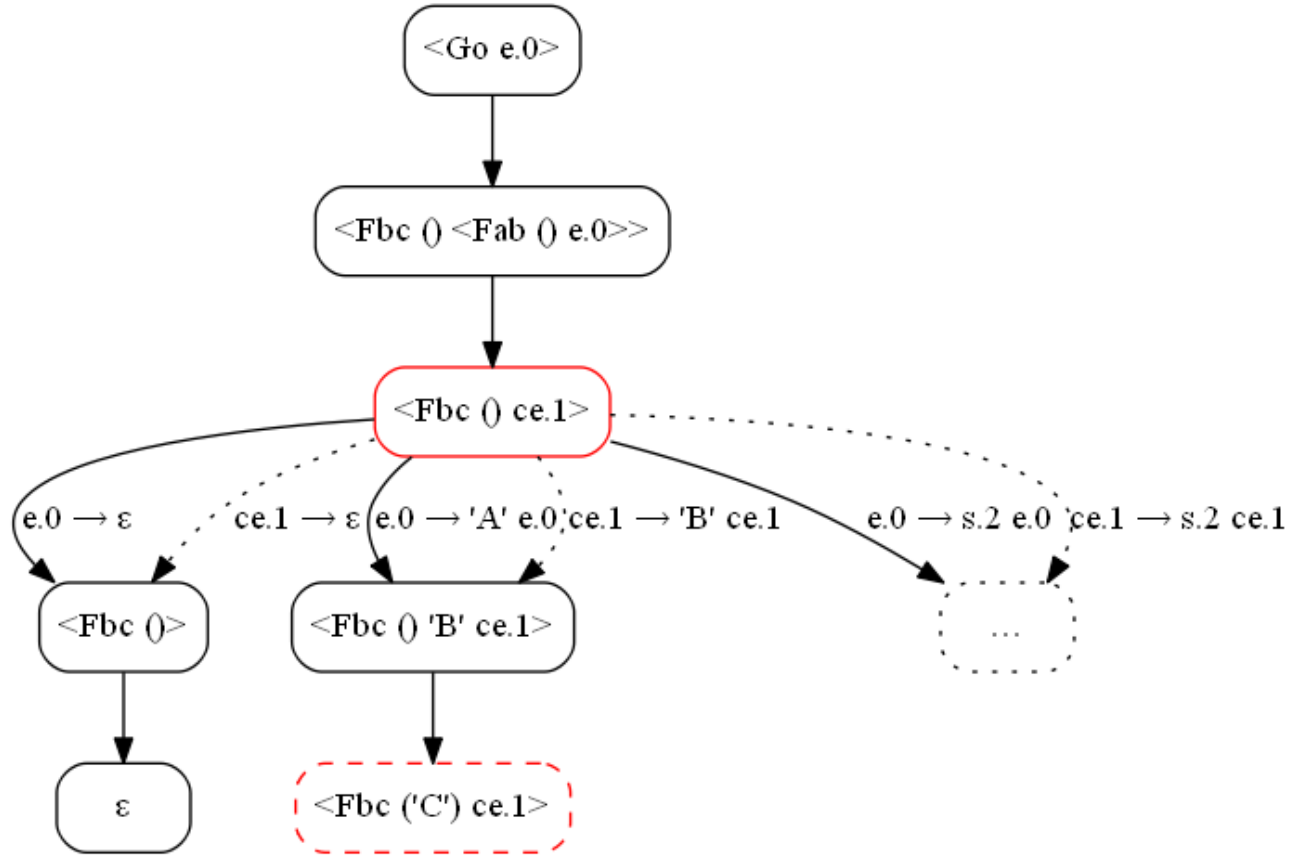
Композиционные параметры и Рефал

- Выворачиваем:



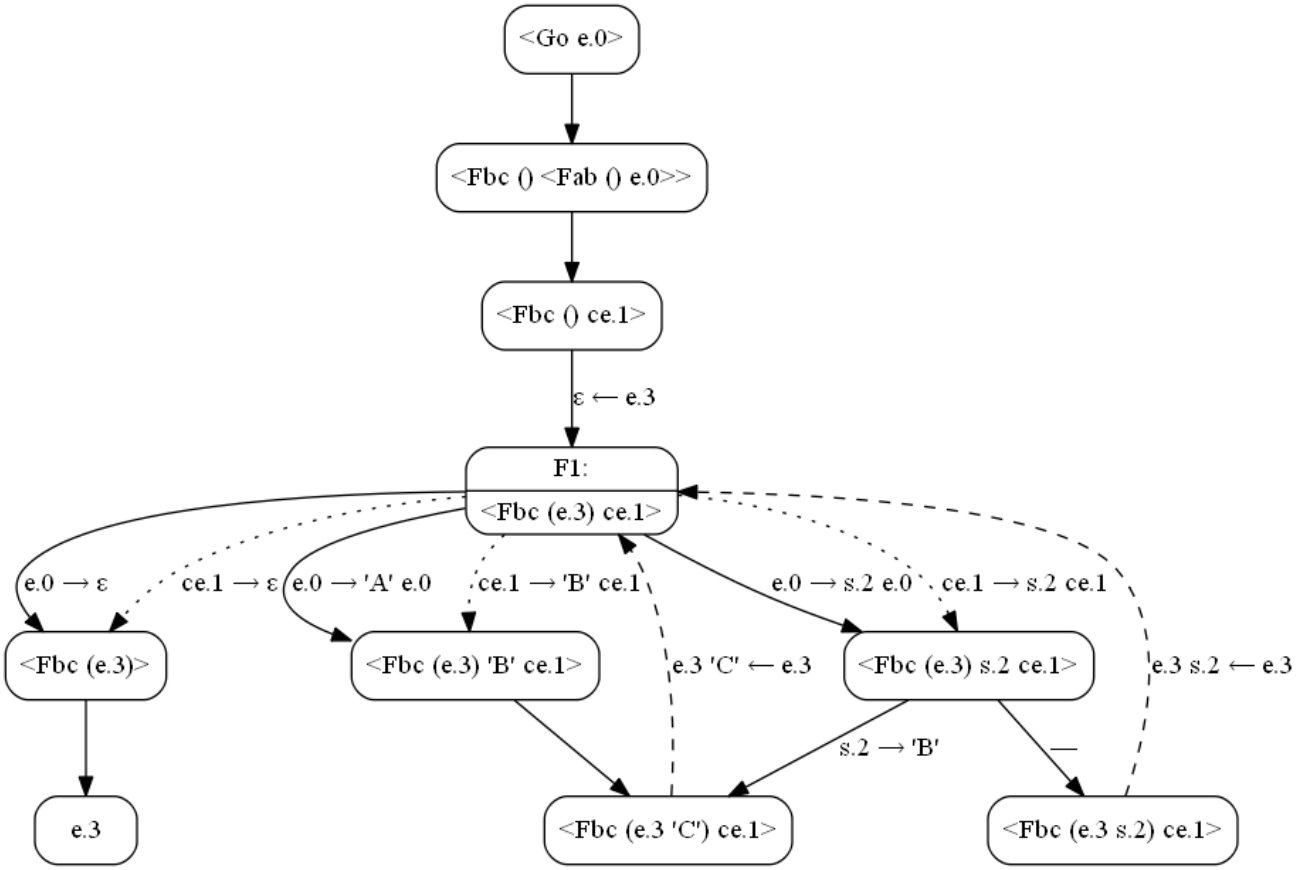
Композиционные параметры и Рефал

- Растворяем:



Композиционные параметры и Рефал

- Растворяем:



Композиционные параметры и Рефал

- Остаточная программа:

```
$ENTRY Go {
  e.0 = <F1 ε, e.0>;
}

F1 {
  e.3, ε = e.3;
  e.3, 'A' e.0 = <F1 e.3 'C', e.0>;
  e.3, 'B' e.0 = <F1 e.3 'C', e.0>;
  e.3, s.2 e.0 = <F1 e.3 s.2, e.0>;
}
```

Композиционные параметры и Рефал

- На самом деле, можно было на стадии растворения вместо e.3 ввести композиционный параметр ce.3 и выполнить новое сгущение, но я не стал этого делать для экономии времени.
- Нетрудно убедиться, что остаточная программа имела бы вид

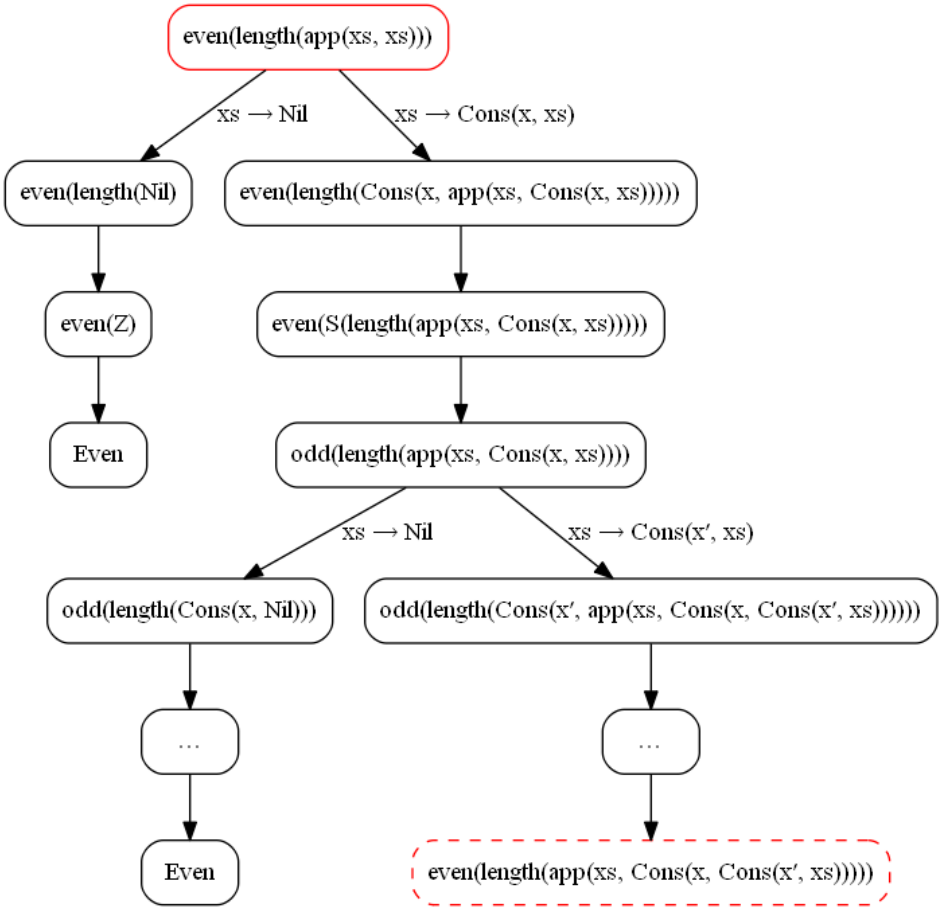
```
$ENTRY Go {
  e.0 = <F1 e.0>;
}

F1 {
  ε = ε;
  'A' e.0 = 'C' <F1 e.0>;
  'B' e.0 = 'C' <F1 e.0>;
  s.2 e.0 = s.2 <F1 e.0>;
}
```

Спасибо за внимание

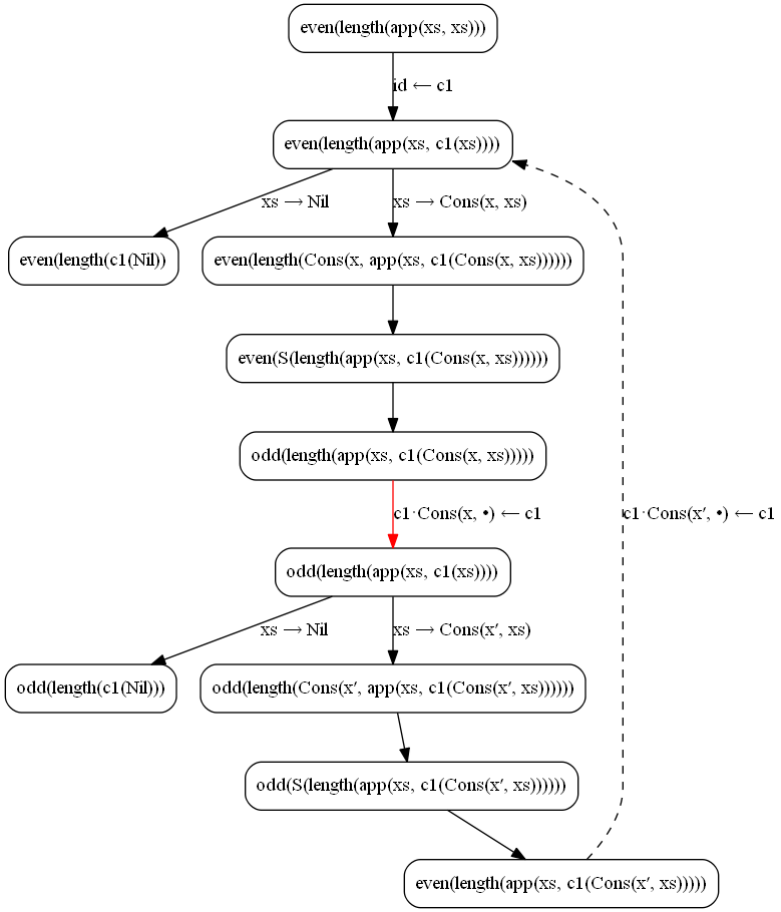
Верификация: length(xs ++ xs) чётно

```
even(length(app(xs, xs)))

where
app(Nil, zs) = zs;
app(Cons(y, ys), zs) = Cons(y, app(ys, zs));

length(Nil) = Z;
length(Cons(x, xs)) = S(length(xs));

even(Z) = Even;
even(S(n)) = odd(n);

odd(Z) = Odd;
odd(S(n)) = even(n);
```

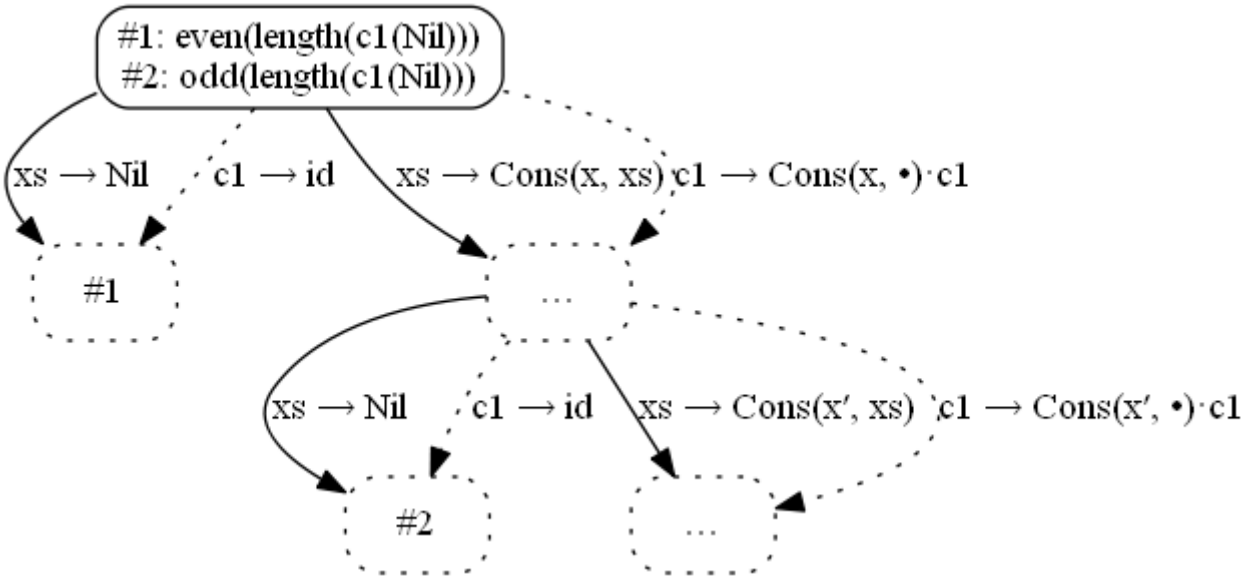
Верификация: length(xs ++ xs) чётно



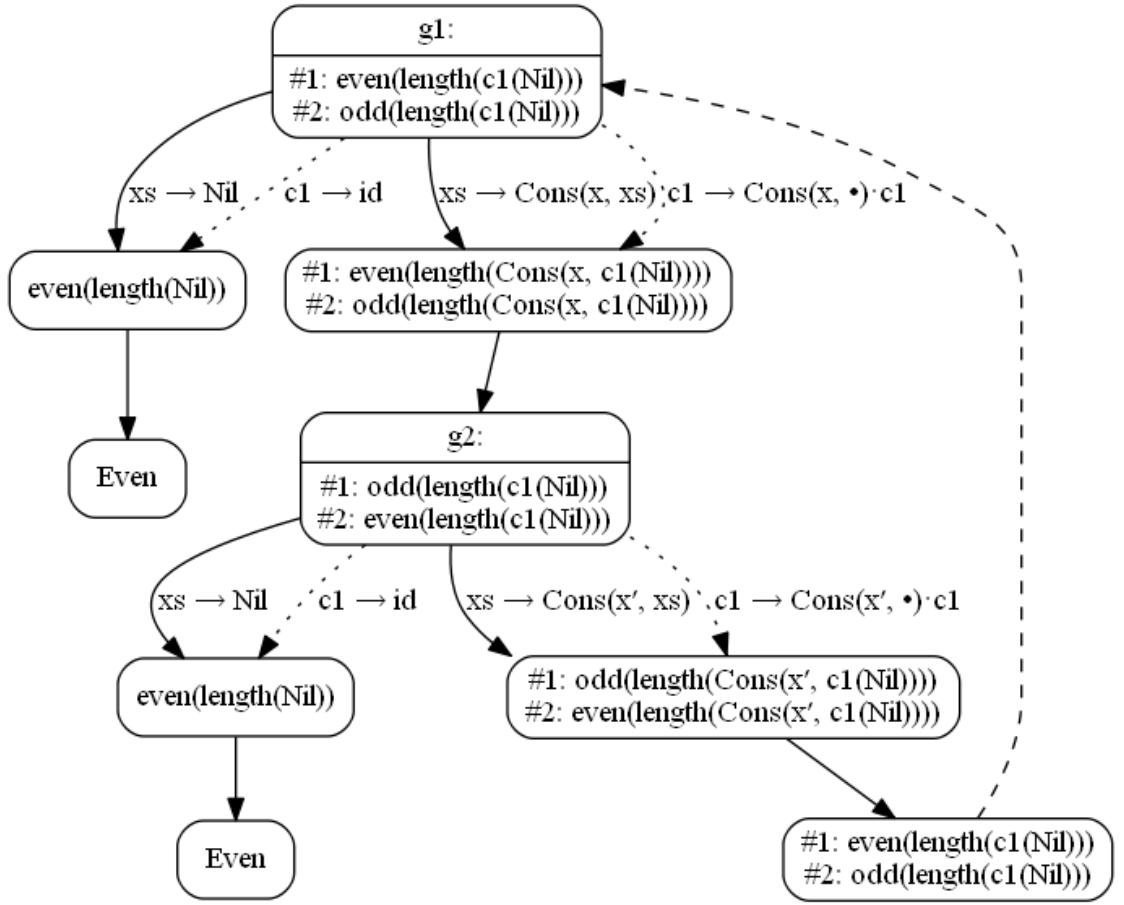
Верификация: length(xs ++ xs) чётно



Верификация: length(xs ++ xs) чётно



Верификация: length(xs ++ xs) чётно



Верификация: length(xs ++ xs) чётно

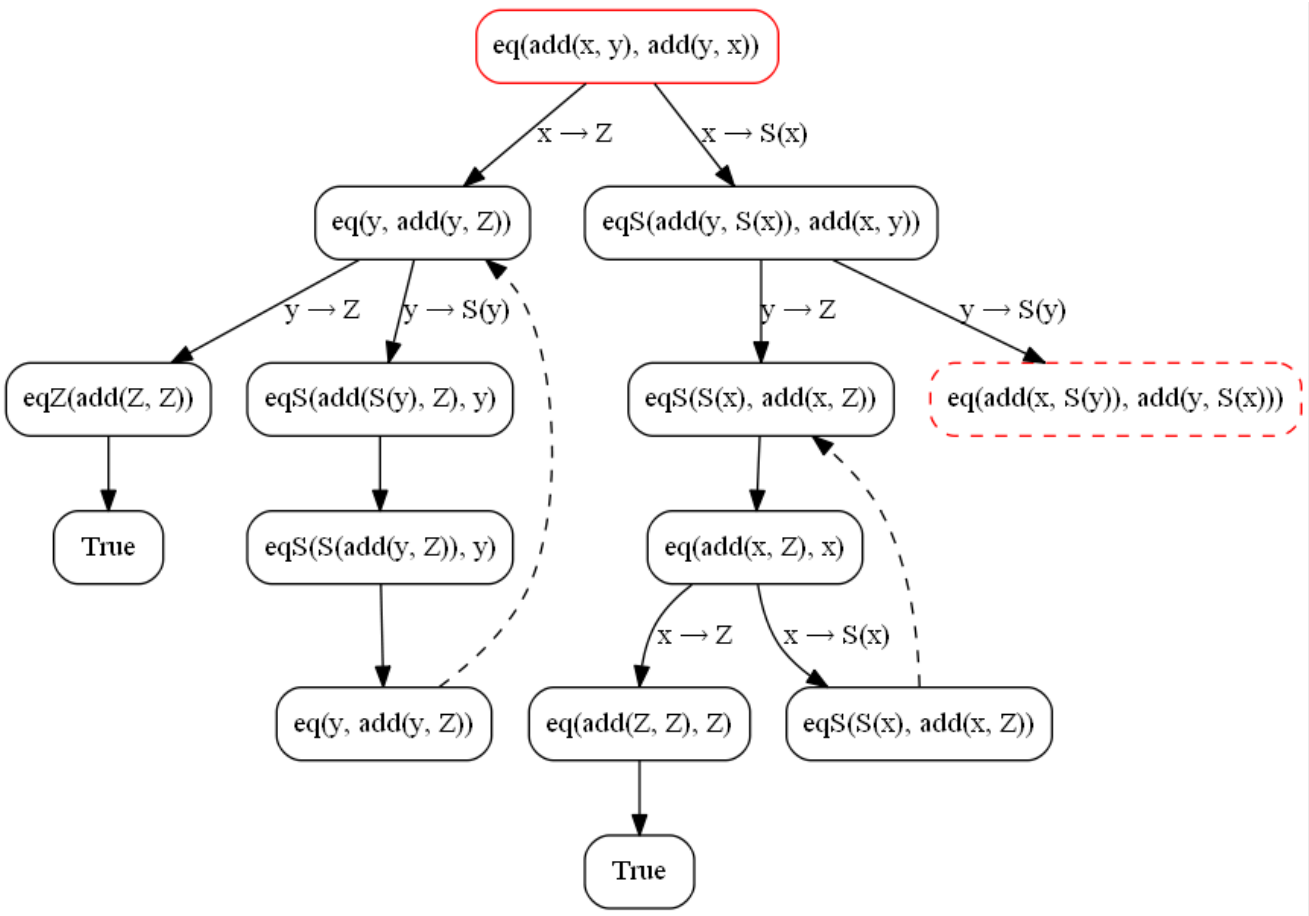
- Остаточная программа:

```
g1(xs)

where
g1(Nil) = Even;
g1(Cons(x, xs)) = g2(xs);

g2(Nil) = Even;
g2(Cons(x', xs)) = g1(xs);
```

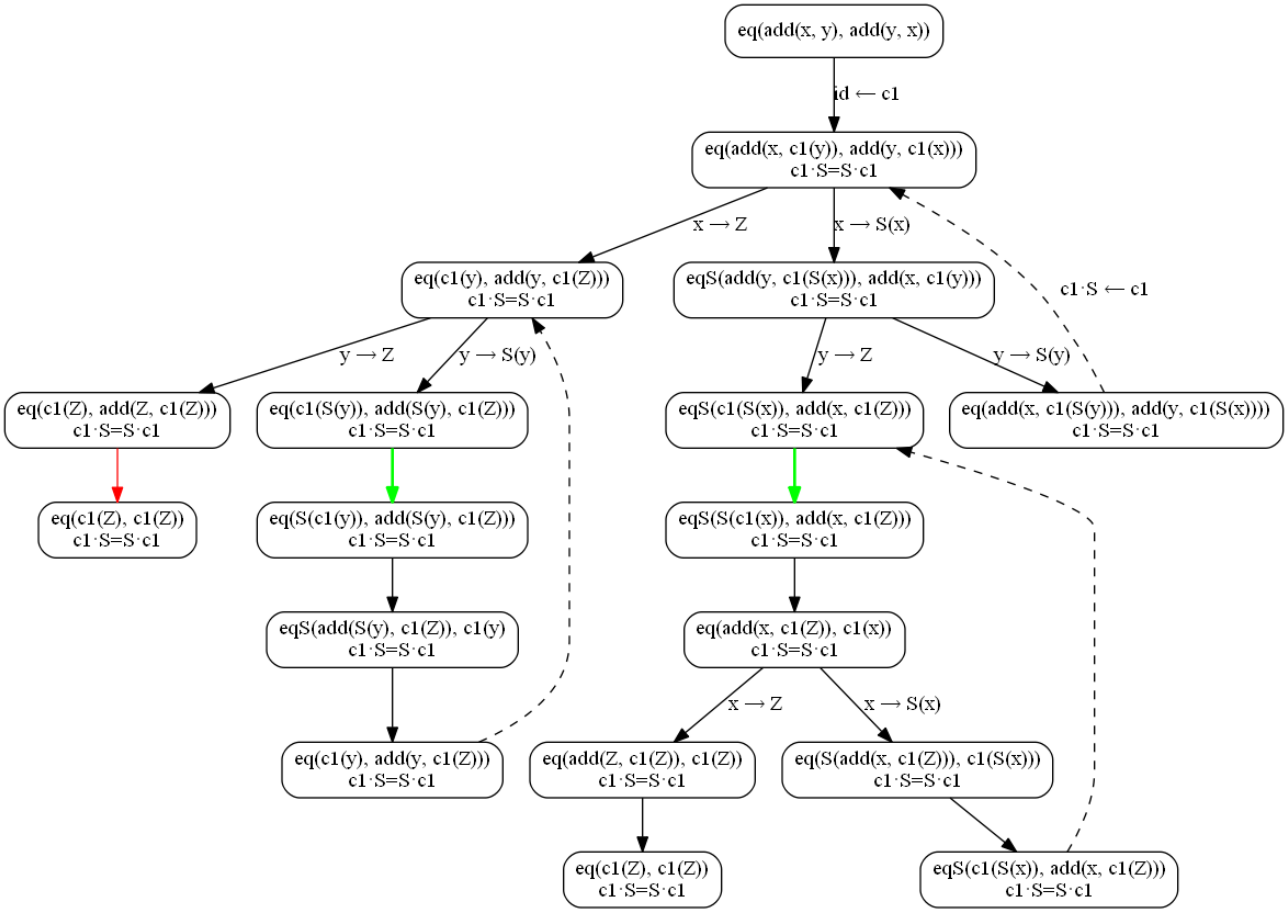
Верификация: x + y = y + x

```
eq(add(x, y), add(y, x)

where
add(Z, y) = y;
add(S(x), y) = S(add(x, y));

eq(Z, y) = eqZ(y);
eq(S(x), y) = eqS(y, x);

eqZ(Z) = True;
eqZ(S(y)) = False;

eqS(Z, x) = False;
eqS(S(y), x) = eq(x, y);
```

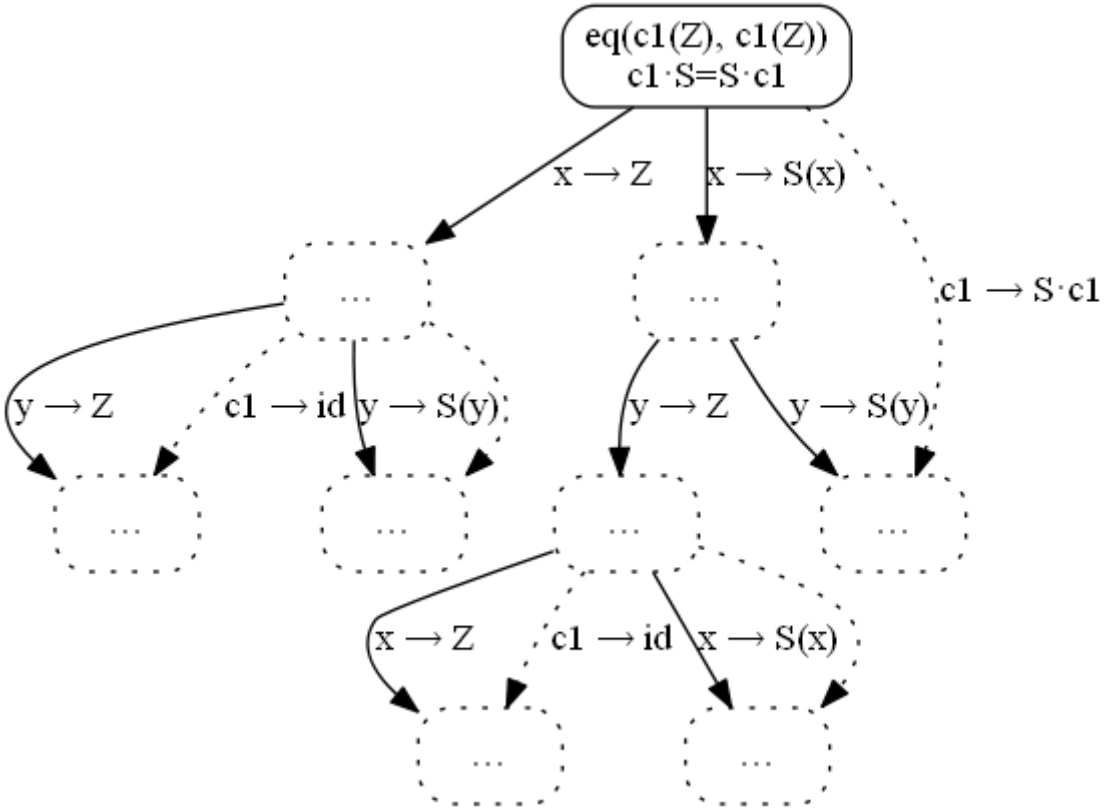
Верификация: x + y = y + x



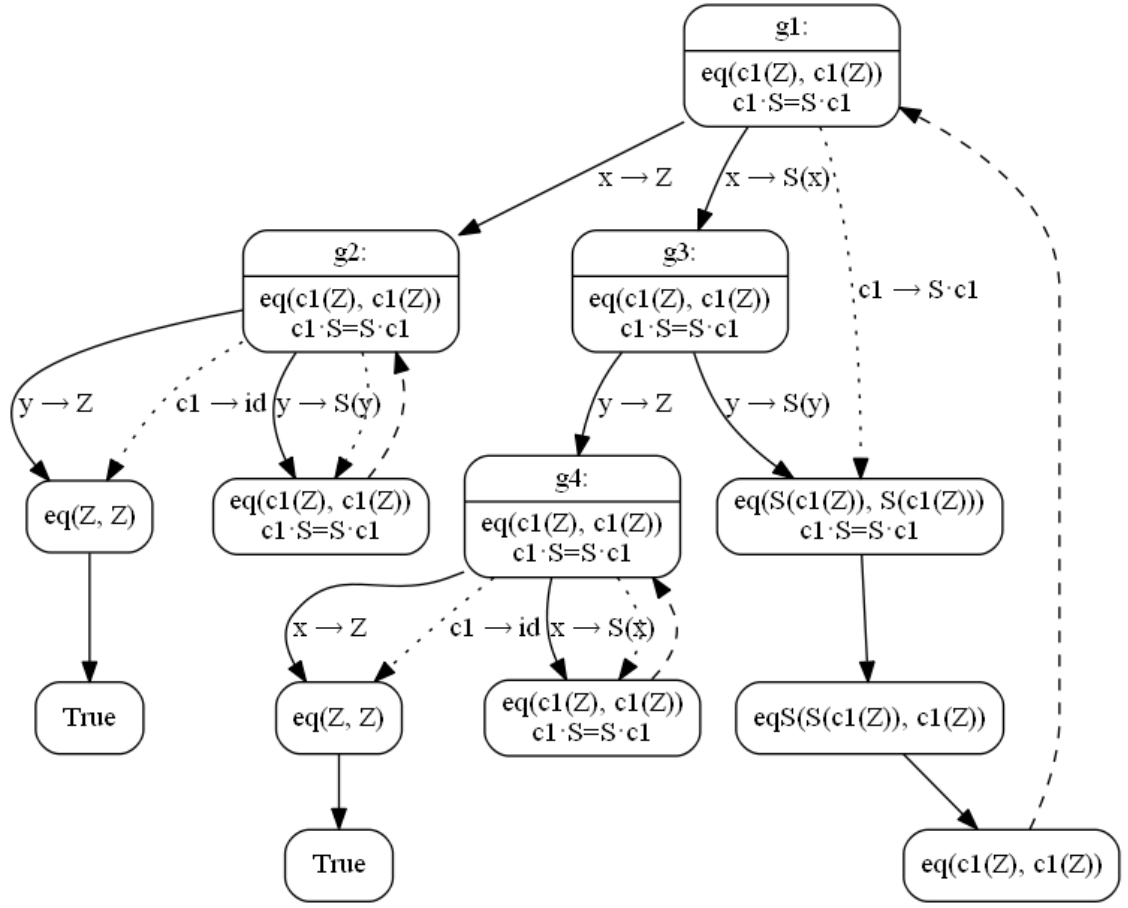
Верификация: x + y = y + x



Верификация: x + y = y + x



Верификация: x + y = y + x



Верификация: x + y = y + x

- Остаточная программа:

```
g1(x, y)

where
g1(Z, y) = g2(y);
g1(S(x), y) = g3(y, x);

g2(Z) = True;
g2(S(y)) = g2(y);

g3(Z, x) = g4(x);
g3(S(y), x) = g1(x, y);

g4(Z) = True;
g4(S(x)) = g4(x);
```