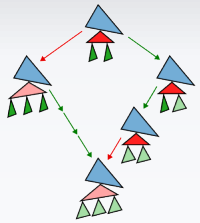
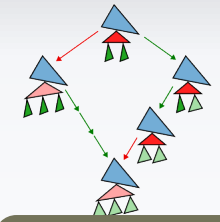


Лабораторная работа 5

Утилитарная часть: шпаргалка к экзамену по ТФЯ

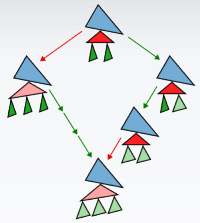
Выбрать один вопрос из базовой или дополнительной группы и написать по нему тезисную шпаргалку не меньше, чем на полстраницы и не больше, чем на две страницы (не больше, чем одну, если шпаргалка без картинок). Формат шпаргалок — по выбору группы, но у всех одинаковый и такой, чтобы вы могли их объединить. Вопросы должны быть у всех разные.



Лабораторная работа 5

Генератор отчётов по лабе в LaTeX (нечётный вариант)

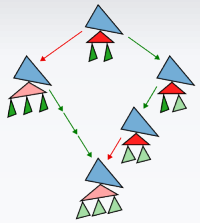
1. Выбрать язык (и среду) программирования, программы которого будут анализироваться генератором отчётов. Определить специальный тип комментария: псевдокод для генератора, — который будет извлекать для отчёта выделенные блоки программы. Комментарии этого специального типа должны быть двух видов: открывающий и закрывающий, причём каждый открывающий комментарий должен начинаться с ключей KEY={*id листинга*} и NAME={*имя листинга*}.
2. Генератор отчётов по лабе принимает на вход ваш проект на целевом ЯП, снабжённый псевдокодовыми комментариями, и дополнительный файл lab_report.bmstu, содержащий данные для шапки и некоторые другие разделы отчёта, описанные в простой грамматике.
3. По этим двум компонентам необходимо породить исходный файл отчёта в LaTeX.



Лабораторная работа 5

Переформатирование РБНФ (чётный вариант)

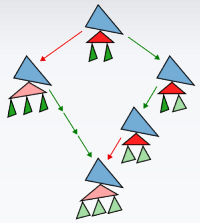
1. Алгоритм требует два «стилевых» файла. Файл current_syntax.txt описывает синтаксис РБНФ, которую нужно переформатировать, файл my_syntax.txt содержит описание желаемого синтаксиса РБНФ.
2. Дана контекстно-свободная грамматика G, записанная в синтаксисе current_syntax. Необходимо автоматически переписать её в форму my_syntax.
3. Поскольку исходный синтаксис может быть сколь угодно дурацким (не предполагается, что он удовлетворяет LR- или LL-свойствам), лучше всего применять общие алгоритмы разбора слова (т.е. данной грамматики G) и сообщать о найденных неоднозначностях (см. подробное описание алгоритма).



(По выбору)

Задача со звёздочкой: наивный TRS-Prolog

- Prolog-программы — это набор правил переписывания одного из двух видов: индуктивное и базисное, и запрос вида
  ?-⟨терм⟩.
  ⟨терм⟩ :- ⟨терм⟩(, ⟨терм⟩)*.
  ⟨терм⟩.
  Здесь ⟨терм⟩ — это конструктор (имя в [a-z]$^+$) на переменных (имена в [A-Z][A-z]*) и структурах языка (в swi-прологе структурами могут быть и trs).
- При вызове ?-⟨терм⟩ интерпретатор строит список всех возможных подстановок в терм (ground-термов, см.ниже), приводящих к успеху унификации, либо просто проверяет, удовлетворяет ли терм спецификации (если он в универсуме Эрбрана).
- Задача — построить аналогичный интерпретатор, в котором ⟨терм⟩ записан в произвольной сигнатуре TRS.

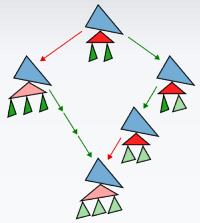


Синтаксис lab_report.bmstu

В файле-заготовке для отчёта по лабораторной работе должны быть следующие разделы:

ключевое слово	переносы	семантика
TITLE { ... }	forced	название лабораторной работы
AUTHOR { ... }	none	фио студента
REVIEWER { ... }	none	фио преподавателя
DEPT { ... }	none	кафедра и группа
START_TASK	free	начало блока описания задачи
END_TASK	free	конец блока описания задачи
START_BODY	free	начало блока описания решения
END_BODY	free	конец блока описания решения
START_TESTS	free	начало блока описания тестов
END_TESTS	free	конец блока описания тестов

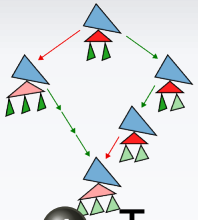
По желанию можно добавить раздел выводов (ключевое слово START_CONCLUSION–END_CONCLUSION).



Заголовок LaTeX-исходника

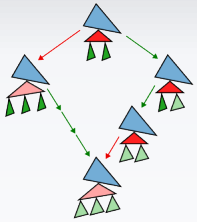
- Данные преамбулы (размещается в исходнике до \begin{document}), кроме данных титульника (см. ниже), должны храниться в отдельном вспомогательном файле и просто приписываться в начало порождаемого документа.
- Пример преамбулы приведён в листинге ниже. Это не гостовский формат отчёта, но первое приближение к нему. Не возбраняется взять и другую преамбулу.

```
\documentclass[14pt,russian]{scrartcl}
\usepackage{geometry}
\geometry{a4paper,tmargin=2cm,bmargin=2.8cm,lmargin=3cm,rmargin=1cm}
\usepackage{tempora}
\usepackage{cmap}
\usepackage[T2A]{fontenc}
\usepackage[utf8]{inputenc}
\usepackage[english, main=russian]{babel}
\linespread{1.3}
```



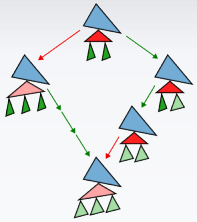
Оформление документа

1. Титульник можно оформлять в свободной форме. Если есть желание подогнать под какую-нибудь специальную форму, обращайтесь в письме. Простейший способ оформления: переложить работу на LaTeX, т.е. название лабораторной поместить в поле \title{...} заголовка, ваше фио и группу (через перенос, а именно комбинацию \\) — в поле \author{...}. Туда же можно и имя проверяющего. После чего просто вызываете \maketitle сразу после \begin{document}.
2. Листинги можно оформлять двумя способами. Первый, пакетно-независимый: \begin{verbatim} *сам листинг* \end{verbatim}. Второй, пакетно-зависимый: подключить listings или minted и поместить листинг внутрь соответствующего окружения.
3. Все листинги должны быть погружены в окружение \begin{figure}[htb] ...\end{figure}. [htb] — это подсказка транслятору LaTeX, где лучше размещать объект. Можете использовать и другую. Сразу после \begin{figure}[htb] добавляем команду \footnotesize, чтобы сделать листинги компактнее. Перед \end{figure} добавляем команду \caption{...}. Внутрь caption переносим имя листинга.
4. Разделы начинаются тегами \section{*название раздела*} и имеют стандартные имена, вшитые в генератор, например: TASK — Постановка задачи; BODY — Реализация; TESTS — Тестирование.



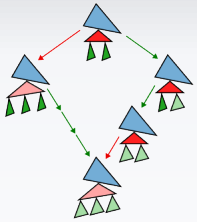
Оформление блоков текста

1. Спецсимволы LaTeX в блоках текста (не листингах) должны экранироваться. Как минимум, это относится к спецсимволам \, №, _, $, &, %, {, } (почти все они экранируются обратным слешем, кроме \— он \textbackslash, и №— он \textnumero).
2. Слова латиницей должны автоматически распознаваться как лексемы, относящиеся к программному коду, и погружаться в моноширинное окружение: \texttt{...}.
3. Исключения из предыдущего пункта — это лексемы-названия программных систем, которые должны быть помечены префиксом $SYSTEM_ в файле lab_report.bmstu, и погружаться в окружение \textsc{...}.



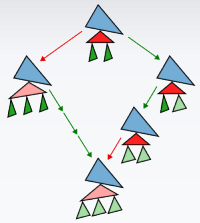
Извлечение листингов

1. Для листингов должна быть создана таблица имён, ассоциирующая каждый листинг с уникальным ключом. Если у нескольких листингов оказались одинаковые ключи — вывести сообщение об ошибке порождения отчёта с указанием именей модулей, где обнаружились одинаковые ключи, и имени этих ключей.
2. Место размещения листинга в файле отчёта определяется ключом. А именно, в соответствующем месте в тексте lab_report.bmstu должно быть объявлено включение листинга: $IMPORT{*ключ листинга*}.
3. В листинги извлекаются куски исходников, выделенные парой открывающий – закрывающий псевдокод. При этом открывающий и закрывающий псевдокомментарии могут содержать также текст, поясняющий листинг, который автоматически перенесётся в отчёт (и будет отформатирован наравне с текстом блоков из lab_report.bmstu). Текст из открывающего псевдокомментария помещается до листинга, из закрывающего — после листинга.
4. Если в строке исходника из блока, переносящегося в отчёт, оказалось больше 85 символов — вывести сообщение об ошибке порождения отчёта с указанием имени модуля, где обнаружилась строка, и собственно строки.



Комментарии к алгоритму

- Структуры, которые требуется распознавать в комментариях исходников и в lab_report, почти регулярны. В комментариях регулярность обеспечивается свойствами самого ЯП, гарантирующего, что комментарий является лексемой. А вот в lab_report таких ограничений нет — внутри блока TITLE могут появиться фигурные скобки, а ключевые слова для титульника могут идти в перепутанном порядке (например, DEPT первым).
- Можете попробовать автоматически сгенерировать отчёт об автоматической генерации отчётов (+1 балл).
- Можете также вносить любые дополнительные фишки в генератор. Креативность оценится до +2 баллов.

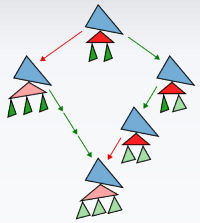


Описание синтаксисов РБНФ

По умолчанию считается, что общее описание синтаксиса записи РБНФ описывается следующей грамматикой G^{Meta}. Здесь все BEGIN_, END_, а также SEP_R, SEP_A, LPAREN, RPAREN — это строки-константы. Ниже синим выделены все нетерминалы, которые разворачиваются не в константы.

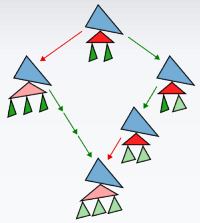
сущность		правая часть в грамматике G^{Meta}
RULE	::=	BEGIN_RULE NTERM SEP_R EXP END_RULE
EXP	::=	ALT | ITER EXP | NTERM EXP | CONST EXP
		| LPAREN EXP RPAREN EXP | ε
NTERM	::=	BEGIN_NTERM NNAME END_NTERM
ALT	::=	BEGIN_ALT EXP SEP_A NEXTALT
NEXTALT	::=	EXP SEP_A NEXTALT | EXP END_ALT
ITER	::=	BEGIN_ITER EXP END_ITER
CONST	::=	BEGIN_CONST CNAME END_CONST

Спецификация синтаксиса РБНФ состоит в указании всех строковых констант из грамматики выше, а также регулярных выражений, описывающих нетерминалы NNAME и CNAME.



Неоднозначность

- В рамках этой задачи неоднозначность никак не разрешается: нет ни приоритетов, ни жадности операторов. Поэтому если маркеры операций отсутствуют либо заданы пересекающимся образом, может возникнуть много разборов одного и того же слова. Из всех разборов при неоднозначности отбрасываются те, которые порождают вырождение EXP в пустое слово.
- Например, если синтаксис итерации такой: ITER ::= EXP* (эндмаркер звёздочка, маркера начала нет), тогда выражение вида ABA* можно понять как (ABA)*, A(BA)*, AB(A)*.
- Если синтаксис у альтернативы есть ALT::= EXP|NEXTALT, NEXTALT::= EXP|NEXTALT | EXP, тогда слово вида A|BBB|A можно распознать как (A|B)B(B|A), (A|BB)(B|A), (A|B)(BB|A), (A|BBB|A).
- Если разбор порождает вырождение EXP в пустое слово при раскрытии итерации, такой разбор считается некорректным и должен быть отброшен.



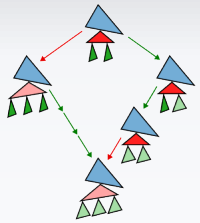
Синтаксис current_syntax и my_syntax

Файлы-описания синтаксиса РБНФ должны содержать расшифровки константных нетерминалов (чёрные на предыдущем слайде); а также регулярные выражения для нетерминалов NNAME и CNAME в следующем простейшем синтаксисе. Ниже красным выделены элементы входного языка.

⟨regexp⟩ ::= ⟨elem_reg⟩ ⟨regexp⟩ | (⟨elem_reg⟩)+⟨regexp⟩
| (⟨elem_reg⟩)*⟨regexp⟩ | ε

⟨elem_reg⟩ ::= [A-Z] | [a-z] | [0-9] | символ | . | !blank!

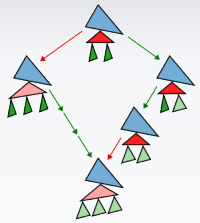
Под точкой подразумевается любой непробельный символ, символ — любой непробельный, кроме квадратных скобок, точки и восклицательного знака, !blank! — указание на пробельный символ (он может использоваться и при описании синтаксиса констант). Если расшифровки нетерминала-константы в файле описания синтаксиса нет, значит, эта константа предполагается равной значению по умолчанию, то есть (для LPAREN,) для RPAREN, = для SEP_R, | для SEP_A, * для END_ITER и пустому слову иначе.



Примеры current_syntax и my_syntax

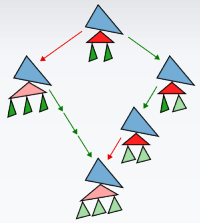
current_syntax	my_syntax
CNAME = (.)*	NNAME = [A-Z]
NNAME = (.)+	CNAME = (.)+
SEP_A = OR	BEGIN_ALT = [
SEP_R = ->	END_ALT =]
END_ITER = *	SEP_R = ::=
BEGIN_NTERM = <	END_RULE = ;
END_NTERM = >	BEGIN_ITER = [
BEGIN_CONST = ’	END_ITER =]*
	BEGIN_CONST = "
	END_CONST = "

Здесь есть возможная проблема перевода из левого синтаксиса в правый: регулярные языки для NNAME и CNAME в my_syntax более узкие, чем в current_syntax. Это может привести к ошибке, а может никак не проявиться при разборе конкретной грамматики (см. следующий слайд).



Пробелы

Пробел (разделитель) считается токеном, не переопределимым в синтаксических грамматиках. Поэтому, если пробельный символ не входит в определение ни одного из токенов в current_syntax, тогда он однозначно обрывает распознавание текущего токена. По умолчанию считается, что имена нетерминалов и терминальных символов определены так, что они содержат другие символы, кроме пробелов. Для ускорения работы алгоритма можно проверить, входят ли пробелы в first- и last-множества нетерминалов. Если нет, тогда несколько подряд идущих пробелов всегда можно заменить единственным ещё перед разбором грамматики.



Пример преобразования

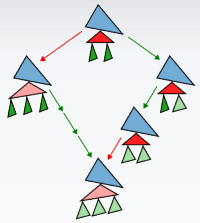
В данном случае РБНФ — это *слова* в вышеописанных грамматиках — вариантах G^{Meta}. Например, следующая РБНФ G^{input} разбирается в current_syntax однозначно с точностью до расстановки альтернатив (пробелы в правых частях здесь существенны: они не матчатся с точкой и обрывают распознавание констант).

```
<S>   ->   <R>*
<R>   ->   'id '-> <E>
<E>   ->   (<E> '| <E>) OR (<E>*) OR 'id
```

Её возможный вид после преобразования в my_syntax:

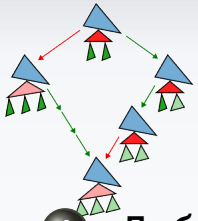
```
S   ::=   [R]*;
R   ::=   "id" "->" E;
E   ::=   [(E "|" E) | ([E]*) | "id"];
```

Проблем с именами нетерминалов и констант не возникло, поскольку в G^{input} они соответствовали допустимым именам



Расстановка альтернатив

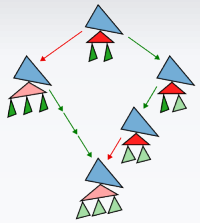
Несколько подряд идущих альтернатив, если у них нет чётко выделенных маркеров начала и конца, могут разбираться многими способами (их число экспоненциально от числа альтернатив). При этом семантически все эти разборы будут описывать единственную грамматику. Поэтому для упрощения и ускорения работы алгоритма можно всегда предполагать, что предпочитается разбор с самым длинным раскрытием альтернативы в NEXTALT.



Алгоритм преобразования

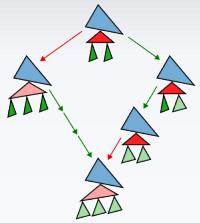
1. Добавляем в грамматику $\mathrm{G}^{\mathrm{Meta}}$ на слайде 10 правила для расшифровки константных разделителей и языков описания нетерминалов и констант. Последние записаны в виде регулярных выражений, поэтому их придётся перевести в форму правил. Получаем грамматику $\mathrm{G}^{\mathrm{Meta}}(\mathrm{current\_syntax})$.
2. Устраняем ε-правила в грамматике $\mathrm{G}^{\mathrm{Meta}}(\mathrm{current\_syntax})$. Если при этом появилось правило с левой рекурсией, сообщаем об ошибке и завершаем работу. При любом преобразовании правил сохраняем имена главных нетерминалов (NTERM, RULE, etc).
3. Порождаем все возможные деревья разбора входной грамматики $\mathrm{G}^{\mathrm{input}}$ в $\mathrm{G}^{\mathrm{Meta}}(\mathrm{current\_syntax})$ (например, нисходящим разбором).
4. Путём замены поддеревьев для главных нетерминалов (NTERM, RULE, etc) порождаем деревья разбора в грамматике $\mathrm{G}^{\mathrm{Meta}}(\mathrm{my\_syntax})$. При этом может возникнуть конфликт при переводе имён нетерминалов и констант, если окажется, что имя, распознаваемое грамматикой $\mathrm{G}^{\mathrm{Meta}}(\mathrm{current\_syntax})$ как нетерминал (константа), не входит в соответствующий язык грамматики $\mathrm{G}^{\mathrm{Meta}}(\mathrm{my\_syntax})$. Если такой конфликт возник, нужно выдать сообщение об ошибке.
5. Если превращение прошло без конфликтов, выводим все возможные версии грамматик $\mathrm{G}^{\mathrm{input}}$ в языке $\mathrm{G}^{\mathrm{Meta}}(\mathrm{my\_syntax})$.

Выражаю благодарность А.Д. Белоусову (выпуск 2023) за замечания и уточнения по задаче.



Комментарии к алгоритму

- Лучше не зашивать грамматику G^{Meta} глубоко в код преобразователя, а держать в виде модуля. Тогда при случае вы можете заменить её на другую, чтобы, например, контекстно-свободным образом переформатировать диаграммы.
- Нисходящий разбор — самый простой вариант алгоритма разбора. Можно воспользоваться и CYK либо KT. Тогда придётся устранять и цепные правила, а при переводе в CNF только вводим новые нетерминалы, а старые не переименовываем. (+2 балла).
- Можно провести предварительный анализ лексем уточненной грамматики $\mathrm{G}^{Meta}(current\_syntax)$. Если некоторый разделитель не распознается как токен, вывести сообщение об этом — скорее всего, на нём будет неоднозначность (+1 балл). Предварительный анализ не блокирует разбор, а только генерирует предупреждения.

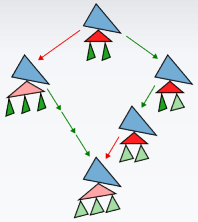


Prolog нормального человека

- Пусть натуральные числа заданы списками, последний элемент которых — константа z (соответствует нулю), которую может предварять некоторое количество констант s. Рассмотрим следующую пролог-программу:

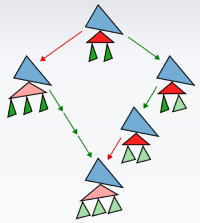
  sum([z],Y,Y).
  sum([s|X],Y,[s|Z]) :- sum(X,Y,Z).

  Она описывает предикат sum(X,Y,Z), истинный на таких списках, в которых Z — это X+Y (на самом деле нет — это верно только если значение Y есть унарное число).
- Запрос ?-sum(X,Y,[s,s,z]) к такой программе выведет список из трёх возможных пар – значений для переменных X и Y, при которых предикат sum(X,Y,[s,s,z]) выполняется.
- Запрос ?-sum([s,s,z],X,X) по здравому размышлению должен зацикливаться, но в современных интерпретаторах пролога он порождает решение (неподвижную точку уравнения n+X=X): X — это бесконечный список [s,s,s,...].
- Запрос ?-sum([s,s,z],Y,Z) выведет пару {Y, [s, s | Y]}.
- Запрос ?-sum(X,X,X) выведет нулевое решение и зациклится.



TRS-Prolog

- А теперь зададим те же натуральные числа не списками, а термами, получающимися из нульместного конструктора z навешиванием унарных конструкторов s:
  sum(z,Y,Y).
  sum(s(X),Y,s(X)) :- sum(X,Y,Z).
- Запрос ?-sum(X,Y,s(s(z))) к такой программе также должен вывести список из трёх возможных пар – значений для переменных X и Y, при которых предикат sum(X,Y,s(s(z))) выполняется. Запрос ?-sum(s(s(z)),Y,Z) выведет пару {Y=Y, Z=s(s(Y))}.
- Запрос ?-sum(s(s(z)),X,X) должен привести к сообщению о циклической унификации.
- Запрос ?-sum(X,X,X) зациклится полностью. Хорошо, если до этого он всё-таки найдёт нулевое решение.
- Запрос ?-sum(sum(X,X,Z),Z,s(s(z))) может выдать false либо зациклиться в зависимости от выбора вызова в стеке для исполнения. В нормальном прологе он будет иметь вид ?-sum(W,Z,[s,s,z]), sum(X,X,Z), W==Z и выдаст false.



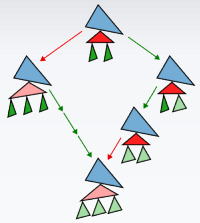
Синтаксис TRS-Prolog-a

Красным выделены элементы входного языка.

⟨fact⟩	::=	⟨term⟩.
⟨clause⟩	::=	⟨term⟩:-⟨term⟩(, ⟨term⟩)*.
⟨term⟩	::=	⟨constructor⟩(⟨arg⟩(, ⟨arg⟩)*)
⟨arg⟩	::=	⟨term⟩ | ⟨var⟩ | ⟨constructor⟩
⟨constructor⟩	::=	[a-z]$^+$([0-9]|[a-z])*
⟨var⟩	::=	[A-Z]$^+$([0-9]|[A-z])*

Сигнатура должна быть корректной: у каждого конструктора только одна местность (если в программе есть sum от трёх аргументов, появление sum с другим количеством аргументов должно приводить к синтаксической ошибке).

Интерпретатор должен выделить так называемые «ground» конструкторы из сигнатуры — такие, которые никогда не являются внешними конструкторами в левых частях правил.



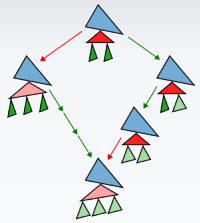
Конфигурации

Состояния при исполнении Prolog-программы — это конфигурации.

Они состоят из стека вызовов и набора равенств между термами.

- Если в равенстве встретились термы с одинаковым внешним конструктором, тогда это равенство заменяется на набор равенств между его аргументами. Например, <sum(s(X),Y,Z) = sum(W,s(X),sum(Y,V,W))> породит систему <s(X) = W>, <Y = s(X)>,<Z = sum (Y,V,W)>.
- Если в равенстве встретились термы с разными внешними конструкторами из класса «ground» — ветвь разбора обрывается. Например <s(sum(X,Y,Z))=z>.
- Если встретилось равенство вида «переменная = терм», тогда делается подстановка терма вместо переменной всюду в конфигурацию, если только это равенство не рекурсивно. Если рекурсивно — также обрываем ветвь разбора.

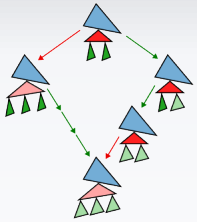
Короче, делается полная унификация (с поправкой на различие между базовыми и функциональными конструкторами).



Исполнение

Переход из конфигурации в множество дочерних происходит следующим образом.

- Самый внешний функциональный (не ground) конструктор активного вызова в стеке унифицируется со всеми предложениями программы с таким же внешним конструктором левой части, не останавливаясь на первом успехе.
- Если удалось унифицироваться с фактом, и в стеке больше ничего нет, причём все уравнения результатной конфигурации содержат только ground-конструкторы — решение найдено, развертка этой ветви завершается.
- Если удалось унифицироваться с фактом, но уравнения конфигурации содержат вызовы функции — переместить их в стек (или не трогать, если в стеке ещё что-то есть).
- Если удалось унифицироваться с индуктивным переходом — совершить его, назначив один из термов в правой части перехода активным вызовом.



Комментарии к алгоритму

- Унификация делает шаг интерпретации, и она же делается при анализе конфигураций. Это самый главный алгоритм TRS-Prolog'а.
- Результатом интерпретации получается не путь вычислений, а дерево таких путей (так же, как и в нормальном прологе). Каждый путь в этом дереве либо завершается нахождением решения, либо завершается обрывом ветки (противоречием), либо потенциально бесконечен. Чтобы исключить бесконечные пути, можно установить максимум на глубину развертки пути (например, 100 шагов).
- Выбор вызова для очередного шага развёртки не жёсткий и может быть сделан произвольно (даже недетерминированно).
- В левых частях правил могут быть вложенные функциональные вызовы (в отличие от нормального пролога).
- Это задача повышенной сложности: за её решение даётся 10 баллов вместо 8.