1 Внутренние структуры данных

Базовые структуры данных алгоритма Ежа: переменная, константа и отрицательное условие.

- Поскольку множество переменных типа строка не меняется, то логично, что переменная представляется просто своим именем: например, x представляется как `(Var 'X')`.
- Константы могут являться результатами сжатия блоков, поэтому логично представлять их парами: имя и индекс. Например, `('D' 1)`.
- Индексы дополнительно имеют расшифровки: мультимножества компонент их длины. Например, если `('D' 1)` — константа, хранящая блок $D^{i_1+i_2+2}$, где D — константа исходного уравнения, то расшифровка будет:

  `('D' 1) is (('D' 0) (i1 1)(i2 1)(const 2))`.
- Отрицательное условие накладывается на переменную и имеет форму КНФ, где каждый литерал — это отрицательное утверждение, чем не может кончаться или начинаться переменная, или утверждение о непустоте значения переменной.

  Например,

```
(OR
    (not ('D' 1) ends (Var 'X'))
    (not ('A' 2) starts (Var 'Y'))
)
```

2 Базовые операции интерактивного режима

- Команда `(PairComp C1 C2)`. Применить сжатие пар:

  `<PairComp (/*Const1*/) (/*Const2*/) (/*EquationData*/)>`.

  Константы не должны совпадать. Результатом сжатия пар станет нумерованный набор новых уравнений с условиями (таким образом, сжатие пар преобразует одно уравнение к множеству уравнений).

- Команда (`BlockComp C`). Применить сжатие блоков:

  `<BlockComp (/*Const*/) (/*EquationData*/)>`.

  Аналогично, результат — нумерованный набор новых уравнений с условиями.

- Команда (`Pick i`). Выбрать из нумерованного списка уравнение, соответствующее номеру: `<Pick /*Number*/ /*EquationSet*/>`.

  Действие позволяет перейти от множества уравнений к единственному.

- Команда (`Subst i1 (/* Multiset */)`). Осуществить подстановку мультимножества компонент, представленного в команде, вместо компоненты `i1`.

3 Возможные результаты вычисления шага

- Если в результате вычислений получилось уравнение вида $\varepsilon = \varepsilon$ или уравнение, которое сводится к нему после вычёркивания всех переменных, тогда объявить, что решение найдено.

- Если в результате вычислений получилось уравнение, содержащее только переменные, причём хотя бы одна переменная имеет условие непустоты, тогда объявить, что на ветке нет минимального решения (остались только неявные сжатия).

- Если команда противоречит условиям (например, требуется повторно сжимать блоки, с которых ничего начинаться не может, или требуется сжать блок, содержащий константу, не встречающуюся в уравнении) — выдать сообщение о некорректном шаге.

- В противном случае результатом вычислений станет одно или несколько состояний (т.е. уравнений в паре с условиями).

4 Построение новых констант в BlockComp и PairComp

Вопрос о том, ограничено ли множество используемых в процессе преобразования констант некоторой разумной (полиномиально зависящей от размера уравнения) величиной, пока открыт. Будем считать, что мы такое ограничение не ставим. Поэтому в дальнейшем нет смысла связывать новые имена констант со старыми, а лучше порождать их по очереди.

Чтобы код был написан полностью в функциональном стиле, достаточно хранить в общем состоянии последний используемый индекс блоков и последнюю использованную новую константу. Далее порождать на их основе очередную константу, например, пользуясь алфавитным сдвигом и комбинацией встроенных функций `<Explode s.X>` – `<Implode e.X>` и `<Numb e.N>` – `<Symb s.N>`.

- Функция `<Explode s.X>` разбирает многобуквенную константу на символы и возвращает строку: `Term -> 'Term'`.
- Функция `<Implode e.X>` собирает символы в многобуквенную константу (не любая последовательность символов может ей стать, выбирается самая длинная подпоследовательность значения `e.X`): `'Term 123' -> Term '123'`.
- Функция `<Symb s.X>` разбирает число на символы и возвращает строку: `123 -> '123'`. Функция `<Numb e.X>` действует обратным образом.

Простейший код для порождения новых индексов блоков всего лишь увеличивает индекс константы. Константой здесь считаем литерал (многобуквенный терм без кавычек), который начинается на букву, а следом за ней обязательно идёт цифра. Например, `i19` (не путаем со строкой `'i19'`).

```
GetNewIndex {
 s.OldIndex
 , <Explode s.OldIndex> : s.Letter e.Digits
 , <Symb <Add <Numb e.Digits> 1>> : e.NewNumber
  = <Implode s.Letter e.NewNumber>;
}
```

Улучшением этого базового варианта может быть вариант, который меняет сначала букву (выбирая в строке возможных вариантов ближайшую справа от текущей), а если не задействованных букв уже не осталось — то цифру, возвращая указатель на букву в начало алфавита.

```
/* В этом варианте есть тонкость: начальным буквам стоит
   присвоить индексы с нулём, а порождённым - начиная с
   единицы, чтобы не путаться. Реализацию этой тонкости
   оставим за кадром ТЗ. */
GetNewIndexUpdated {
 s.OldIndex
 , <Explode s.OldIndex> : s.Letter e.Digits
 , <CharAplhabet> : {
    e.A1 s.Letter s.NextLetter e.A2 /* Можно выбрать следующую
   по очереди букву */
        = <Implode s.NextLetter e.Digits>;
    s.FirstLetter e.A1 s.Letter /* Буква уже последняя ---
   переходим на первую и увеличиваем число */
    , <Symb <Add <Numb e.Digits> 1>> : e.NewNumber
        = <Implode s.FirstLetter e.NewNumber>;
  };
}
```

Заметим, что эти же приёмы можно использовать для порождения не только индексов блоков, но и букв — то есть пар вида `(s.Name s.Number)`. Всё, что нужно будет поменять — это не делать первый вызов функции `<Explode s.OldIndex>` с последующей сборкой символов в число (и обратное ему преобразование), а просто сдвигать букву `s.Name` и увеличивать числовой индекс `s.Number`.

5 Пример

Покажем пример работы цепочки операций на следующем примере.

Исходное уравнение:

```
(AreEqual
 ((Var 'X') ('A' 0) ('A' 0))
 (('B' 0) (Var 'Y'))
```

```
)
(/* Блок условий пуст */)
(/* Блок уравнений на компоненты пуст */)
```

- Применяем `(BlockComp ('A' 0))`. Получаем набор уравнений:

  1. Случай, когда все переменные коллапсируют в блоки:

     ```
     (AreEqual
      (('A' 1))
      (('B' 0) ('A' 2))
     )
     (/* Блок условий пуст */)
     (
      (('A' 1) is ('A' (i1 1) (const 2)))
      (('A' 2) is ('A' (i2 1) (const 0)))
     )
     ```

  2. В блок коллапсирует только `X`:

     ```
     (AreEqual
      (('A' 1))
      (('B' 0) ('A' 2) (Var 'Y') ('A' 3))
     )
     (
       (OR (not empty (Var 'Y')))
       (OR (not ('A' 1) ends (Var 'Y'))
       (OR (not ('A' 2) ends (Var 'Y'))
       (OR (not ('A' 3) ends (Var 'Y'))
       (OR (not ('A' 1) starts (Var 'Y'))
       (OR (not ('A' 2) starts (Var 'Y'))
       (OR (not ('A' 3) starts (Var 'Y'))
     )
     (
      (('A' 1) is (('A' 0) (i1 1) (const 2)))
      (('A' 2) is (('A' 0) (i2 1) (const 0)))
      (('A' 3) is (('A' 0) (i3 1) (const 0)))
     )
     ```

  3. В блок коллапсирует только `Y`:

```
(AreEqual
 (('A' 1) (Var 'X') ('A' 2))
 (('B' 0) ('A' 3))
)
(
  (OR (not empty (Var 'X')))
  (OR (not ('A' 1) ends (Var 'X'))
  (OR (not ('A' 2) ends (Var 'X'))
  (OR (not ('A' 3) ends (Var 'X'))
  (OR (not ('A' 1) starts (Var 'X'))
  (OR (not ('A' 2) starts (Var 'X'))
  (OR (not ('A' 3) starts (Var 'X'))
)
(
 (('A' 1) is (('A' 0) (i1 1) (const 0)))
 (('A' 2) is (('A' 0) (i2 1) (const 2)))
 (('A' 3) is (('A' 0) (i3 1) (const 0)))
)
```

4. Ничего не коллапсирует:

```
(AreEqual
 (('A' 1) (Var 'X') ('A' 2))
 (('B' 0) ('A' 3) (Var 'Y') ('A' 4))
)
(
  (OR (not empty (Var 'X')))
  (OR (not empty (Var 'Y')))
  (OR (not ('A' 1) ends (Var 'X'))
  (OR (not ('A' 2) ends (Var 'X'))
  (OR (not ('A' 3) ends (Var 'X'))
  (OR (not ('A' 4) ends (Var 'X'))
  (OR (not ('A' 1) starts (Var 'X'))
  (OR (not ('A' 2) starts (Var 'X'))
  (OR (not ('A' 3) starts (Var 'X'))
  (OR (not ('A' 4) starts (Var 'X'))
  (OR (not ('A' 1) ends (Var 'Y'))
  (OR (not ('A' 2) ends (Var 'Y'))
  (OR (not ('A' 3) ends (Var 'Y'))
```

```
    (OR (not ('A' 4) ends (Var 'Y'))
    (OR (not ('A' 1) starts (Var 'Y'))
    (OR (not ('A' 2) starts (Var 'Y'))
    (OR (not ('A' 3) starts (Var 'Y'))
    (OR (not ('A' 4) starts (Var 'Y'))
  )
   (('A' 1) is (('A' 0) (i1 1) (const 0)))
   (('A' 2) is (('A' 0) (i2 1) (const 2)))
   (('A' 3) is (('A' 0) (i3 1) (const 0)))
   (('A' 4) is (('A' 0) (i4 1) (const 0)))
  )
```

- Выбираем четвёртый случай: `(Pick 4)`. Осуществляем ряд подстановок индексов:

  ```
  (Subst i1 ((i3 1) (const 0)))
  ```

  ```
  (Subst i4 ((i2 1) (const 2)))
  ```

  Интерактивный режим сам сократит одинаковые префиксы и удалит ненужные зависимости, ведь констант `('A' k)` после этого в уравнении не останется:

  ```
  (AreEqual
   ((Var 'X'))
   (('B' 0) (Var 'Y'))
  )
  (/* Тут ничего нет, кроме условий на непустоту:
      все упоминания констант, связанных с 'A',
      исчезли из уравнения, и нет никаких связей этих
      констант ни с чем, кроме как с собой, в блоке
      индексов, значит, и условия на них уже не нужны */
    (OR (not empty (Var 'X')))
    (OR (not empty (Var 'Y')))
   )
  (/* Тут тоже ничего нет --- всё сократилось */)
  ```

- Сжимаем блоки `('B' 0)`. Опять получаем четыре случая.

  1. Случай, когда все переменные коллапсируют в блоки:

```
(AreEqual
 (('B' 1))
 (('B' 2))
)
(/* Блок условий пуст */)
(
 (('B' 1) is ('B' (i1 1) (const 0)))
 (('B' 2) is ('B' (i2 1) (const 1)))
)
```

2. В блок коллапсирует только `X`:

```
(AreEqual
 (('B' 1))
 (('B' 2) (Var 'Y') ('B' 3))
)
(
  (OR (not empty (Var 'Y')))
  (OR (not ('B' 1) ends (Var 'Y'))
  (OR (not ('B' 2) ends (Var 'Y'))
  (OR (not ('B' 3) ends (Var 'Y'))
  (OR (not ('B' 1) starts (Var 'Y'))
  (OR (not ('B' 2) starts (Var 'Y'))
  (OR (not ('B' 3) starts (Var 'Y'))
)
(
 (('B' 1) is (('B' 0) (i1 1) (const 0)))
 (('B' 2) is (('B' 0) (i2 1) (const 1)))
 (('B' 3) is (('B' 0) (i3 1) (const 0)))
)
```

3. В блок коллапсирует только `Y`:

```
(AreEqual
 (('B' 1) (Var 'X') ('B' 2))
 (('B' 3))
)
(
  (OR (not empty (Var 'X')))
  (OR (not ('B' 1) ends (Var 'X'))
  (OR (not ('B' 2) ends (Var 'X'))
```

```
  (OR (not ('B' 3) ends (Var 'X'))
  (OR (not ('B' 1) starts (Var 'X'))
  (OR (not ('B' 2) starts (Var 'X'))
  (OR (not ('B' 3) starts (Var 'X'))
)
(
 (('B' 1) is (('B' 0) (i1 1) (const 0)))
 (('B' 2) is (('B' 0) (i2 1) (const 0)))
 (('B' 3) is (('B' 0) (i3 1) (const 1)))
)
```

4. Ничего не коллапсирует:

```
(AreEqual
 (('B' 1) (Var 'X') ('B' 2))
 (('B' 3) (Var 'Y') ('B' 4))
)
(
  (OR (not empty (Var 'X')))
  (OR (not empty (Var 'Y')))
  (OR (not ('B' 1) ends (Var 'X'))
  (OR (not ('B' 2) ends (Var 'X'))
  (OR (not ('B' 3) ends (Var 'X'))
  (OR (not ('B' 4) ends (Var 'X'))
  (OR (not ('B' 1) starts (Var 'X'))
  (OR (not ('B' 2) starts (Var 'X'))
  (OR (not ('B' 3) starts (Var 'X'))
  (OR (not ('B' 4) starts (Var 'X'))
  (OR (not ('B' 1) ends (Var 'Y'))
  (OR (not ('B' 2) ends (Var 'Y'))
  (OR (not ('B' 3) ends (Var 'Y'))
  (OR (not ('B' 4) ends (Var 'Y'))
  (OR (not ('B' 1) starts (Var 'Y'))
  (OR (not ('B' 2) starts (Var 'Y'))
  (OR (not ('B' 3) starts (Var 'Y'))
  (OR (not ('B' 4) starts (Var 'Y'))
)
 (('B' 1) is (('B' 0) (i1 1) (const 0)))
 (('B' 2) is (('B' 0) (i2 1) (const 0)))
```

```
 (('B' 3) is (('B' 0) (i3 1) (const 1)))
 (('B' 4) is (('B' 0) (i4 1) (const 0)))
)
```

- Пусть далее рассматривается опять ветка 4: `(Pick 4)`, и аналогичные рассмотренным ранее подстановки:

  `(Subst i1 ((i3 1) (const 1)))`

  `(Subst i2 ((i2 1) (const 0)))`

  После такого преобразования останется только уравнение

  `(AreEqual ((Var 'X')) ((Var 'Y')))`

  с условиями непустоты, и интерактивный режим должен сообщить, что минимальное решение на этой ветке не существует (т.к. остались только переменные).

- Если бы рассматривалась первая ветка, то следующая подстановка:

  `(Subst i1 ((i2 1) (const 1)))`

  привела бы к сообщению о нахождении решения.