メモ

例題の定義

0.1 アルゴリズム簡略版

定義 1 (検索されうる引数)**.** atom($arg_0, \ldots, arg_n$) を Head に含み, かつ, いずれかの辺の式が $arg_i, 0 \leq i \leq n$ を含むような, 比較を行う二項演算子 op を Guard に含むようなルールが存在する場合, atom の第 i 引数は, **op によって検索されうる**（i th argument of atom can be searched by op）.

定義 2 (labeled rule)**.** これ以降, LMNtal ルールにおいて, Head :- Guard — Body. もしくは $\text{Head}_1 \backslash \text{Head}_2$:- Guard — Body. のように記述された全てのルールについて, Head に出現するある 1 つのアトム（atom($arg_0, \ldots, arg_n$)）にラベル付けしたルール (labeled rule) をラベル付きルール（もしくは単にルール, rule）と呼ぶ. ラベル付けされたアトムをターゲットアトム（target_atom） と呼び, Head に出現する全てのアトムについてラベル付きルールが生成される.

定義 3 (typed functor)**.** アトム名と引数の型の組を typed functor と呼ぶ.

typed functor	:-	Name(Args)
Name	:-	String
Args	:-	Args,Args | Type
Type	:-	Int | Float | String | Unary | HL | Link

Algorithm 1 全体のアルゴリズム

Require: A はルールのヘッドに出現するアトムの typed functor の集合
Require: Rule はラベル付きルールの集合
procedure 全体
  while $\exists tf \in A$, D_{tf}_p is not empty **do**
    Exec_Insert(tf) ▷ Body に一度も出現しないアトムは while の外にくくりだせる
  end while
end procedure

Algorithm 2 全体のアルゴリズム特殊化

procedure 全体
  Exec_Insert(start(HL))
  Exec_Insert(distance(Int,HL,HL,Int))
  while $\exists tf \in$ {between(Int,HL,Int)}, D_{tf}_p is not empty **do**
    Exec_Insert(tf)
  end while
end procedure

Algorithm 3 Insertion Loop

```
Require: tf : 入力, typed functor
Require: Rule : tf にラベルがついた labeled rule の集合
  procedure EXEC_INSERT(tf)
    while D_tf_p is not empty do
      target ← D_tf_p.pop_front()
      for all rule ∈ Rule do
        ret ← Exec_Rule(rule, target)
        if ret == Status_Continue then                ▷ If target atom is deleted
          flag ← true
          break
        end if
      end for
      if flag then
        continue
      end if
      D_tf_al.push_back(target)
    end while
  end procedure
```

Algorithm 4 Insertion Loop 特殊化：start(HL)

```
Require: tf = start(HL)
Require: Rule = {start }
  procedure EXEC_INSERT(tf)
    while D_tf_p is not empty do
      target ← D_tf_p.front()
      D_tf_p.pop_front()
      for all rule ∈ Rule do                          ▷ start ルール
        ret ← Exec_Rule(rule, target)
        if ret == Status_Continue then                ▷ If target atom is deleted
          Flag ← true
          berak
        end if
      end for
      if Flag then
        continue
      end if
      D_tf_al.push_back(target)
    end while
  end procedure
```

Algorithm 5 Insertion Loop 特殊化：between(Int,HL,Int)

```
Require: tf = between(Int,HL,Int)
Require: Rule = {inconsistent, intersect, propagate_forward, propagate_backward }
  procedure EXEC_INSERT(tf)
    while D_tf_p is not empty do
      target ← D_tf_p.front()
      D_tf_p.pop_front()
      for all rule ∈ Rule do
        ret ← Exec_Rule(rule, target)
        if ret == Status_Continue then                    ▷ If target atom is deleted
          Flag ← true
          berak
        end if
      end for
      if Flag then
        continue
      end if
      D_tf_al.push_back(target)
    end while
  end procedure
```

Algorithm 6 Insertion Loop 特殊化：distance(Int,HL,HL,Int)

```
Require: tf = distance(Int,HL,HL,Int)
Require: Rule = {propagate_forward, propagate_backward }
  procedure EXEC_INSERT(tf)
    while D_tf_p is not empty do
      target ← D_tf_p.front()
      D_tf_p.pop_front()
      for all rule ∈ Rule do
        ret ← Exec_Rule(rule, target)
        if ret == Status_Continue then                    ▷ If target atom is deleted
          Flag ← true
          berak
        end if
      end for
      if Flag then
        continue
      end if
      D_tf_al.push_back(target)
    end while
  end procedure
```

Algorithm 7 Rule Execution, Head size = 0

```
Require: rule : target にラベルが付いたルール. H, target :- G | B.
Require: target : ターゲットアトムの typed functor
  procedure EXEC_RULE(rule, target)
    case |H| = 0
    if G is true then
      for all atom ∈ B do
        tf ← typed functor of atom
        if D_tf_p exists then
          D_tf_p.push_back(atom)
        else
          print atom
        end if
      end for
      if target is to be deleted by rule then
        return Status_Continue
      end if
    end if
    return Status_Not_Continue
  end procedure
```

Algorithm 8 Rule Execution, Head size = 1

```
Require: rule : target にラベルが付いたルール. H, target :- G | B.
Require: target : ターゲットアトムの typed functor
  procedure EXEC_RULE(rule, target)
    case |H| = 1
    tmp_atom ∈ Head
    tf ← typed functor of tmp_atom
    while true do
      if rule で, {target} について tf が ith arg = j で検索可能 then
        pair_0 ← IS_tf_vec_i[j].next()          ▷ .next() fetches deque elements from front to end
      else
        pair_0 ← D_tf_al.next()
      end if
      if pair_0 の取得に失敗 then                ▷ deque iteration is complete
        break
      end if
      if G is true then
        for all tf ∈ B do
          if D_tf_p exists then
            D_tf_p.push_back(atom)
          else
            print atom
          end if
        end for
        if target is to be deleted by rule then
          return Status_Continue
        end if
      end if
    end while
    return Status_Not_Continue
  end procedure
```

Algorithm 9 Rule Execution 特殊化：start

```
Require: rule = start
Require: target = start(!X)
Require: Head =
Require: Body = between(0,!X,0)
  procedure EXEC_RULE(rule, target)
    if true then                                                    ▷ Guard doesn't exist
      for all atom ∈ Body do
        atom が連結グラフならできるだけ書き換える
        if between_p exists then                                    ▷ between_p is deque for between(Int,HL,Int)
          between_p.push_back(atom)
        else                                                        ▷ never reach here
          dump_text ← dump_text + atom + "."
        end if
      end for
      if target is deleted by this rule then                        ▷ yes
        return Status_Continue
      end if
    end if
    return Status_Not_Continue
  end procedure
```

Algorithm 10 Rule Execution 特殊化２：start

```
Require: rule = start
Require: target = start(!H)
  procedure EXEC_RULE(rule, target)
    start_p.push_back(between(0,!X,0))
    return Status_Continue
  end procedure
```

Algorithm 11 Rule Execution 特殊化：inconsistent

```
Require: rule = inconsistent
Require: target = between(A,!X,B)
Require: Head =
Require: Body = fail(A,!X,B)
  procedure EXEC_RULE(rule, target)
    if A>B then
      for all atom ∈ Body do
        if fail_p exists then                                       ▷ fail_p doesn't exist
          fail_p.push_back(atom)
        else                                                        ▷ always reach here
          dump_text ← dump_text + atom + "."
        end if
      end for
      if target is deleted by this rule then                        ▷ yes
        return Status_Continue
      end if
    end if
    return Status_Not_Continue
  end procedure
```

Algorithm 12 Rule Execution 特殊化 2 ：inconsistent

```
Require: rule = inconsistent
Require: target = between(A,!X,B)
  procedure EXEC_RULE(rule, target)
    if A>B then
      dump_text ← dump_text + fail(A,!X,B) + "."
      return Status_Continue
    end if
    return Status_Not_Continue
  end procedure
```

Algorithm 13 Rule Execution 特殊化：intersect

```
Require: rule = intersect
Require: target = between(A,!Y,B)      ▷ between(A,!Y,B), between(C,!Y,D) をすべて入れ替えたものも同様
Require: Head = between(C,!Y,D)
Require: Body = between(max(A,C),!Y,min(B,D))
  procedure EXEC_RULE(rule, target)
    while true do
      if true then                    ▷ between(C,!Y,D) が ith arg = j, i=2, j=id=HLID of !Y で検索可能
        pair_0 ← "between"_vec_2[id].next()      ▷ .next() fetches deque elements from front to end
      else
        pair_0 ← "pair_0"_al.next()
      end if
      if pair_0 の取得に失敗 then                                          ▷ deque iteration is complete
        break
      end if
      if true then
        for all atom ∈ Body do
          atom が連結グラフならできるだけ書き換える
          if "atom"_p exists then                          ▷ = if atom is between(Int,HL,Int)
            "atom"_p.push_back(atom)                                  ▷ 常に between_p.push
          else                                          ▷ max, min が between の引数に残った場合
            dump_text ← dump_text + atom + "."
          end if
        end for
        if true then
          return Status_Continue
        end if
      end if
    end while
    return Status_Not_Continue
  end procedure
```

Algorithm 14 Rule Execution 特殊化 2 ：intersect

```
Require: rule = intersect
Require: target = between(A,!Y,B)
Require: Head = between(C,!Y,D)
Require: Body = between(max(A,C),!Y,min(B,D))
  procedure EXEC_RULE(rule, target)
    while true do
      pair_0 ← "between"_vec_2[id].next()        ▷ .next() fetches deque elements from front to end
      if pair_0 の取得に失敗 then                  ▷ deque iteration is complete
        break
      end if
      if A>C then
        L0 ← A
      else if A≤C then
        L0 ← C
      else
        L0 ← max(A,C)
      end if
      if B>D then
        L2 ← D
      else if B≤D then
        L2 ← B
      else
        L2 ← min(B,D)
      end if
      atom ← between(L0,!Y,L2)
      if "atom"_p exists then                    ▷ = if atom is between(Int,HL,Int)
        between_p.push_back(atom)
      else                                       ▷ max, min が between の引数に残った場合
        dump_text ← dump_text + atom + "."
      end if
      return Status_Continue
    end while
    return Status_Not_Continue
  end procedure
```

Algorithm 15 Rule Execution 特殊化：propagation_forward

```
Require: rule = propagation_forward
Require: target = between(A,!Y,B)
Require: Head = distance(C,!Y,!Z,D)
Require: Body = between(AC,!Y,BD)
Require: hist : tuple(A,!Y,B,C,!Z,D) を key とする hash table
  procedure EXEC_RULE(rule, target)
    while true do
      if true then                          ▷ distance(C,!Y,!Z,D) が ith arg = j, i=2, j=id=HLID of !Y で検索可能
        pair_0 ← distance_vec_2[id].next()          ▷ .next() fetches deque elements from front to end
      else
        pair_0 ← "pair_0"_al.next()
      end if
      if pair_0 の取得に失敗 then                                          ▷ deque iteration is complete
        break
      end if
      if hist.find((A,!Y,B,C,!Z,D)) == false then
        hist.insert((A,!Y,B,C,!Z,D))
        AC ← A + C
        BD ← B + D
        for all atom ∈ Body do
          if "atom"_p exists then
            "atom"_p.push_back(atom)
          else                                                          ▷ never reach here
            dump_text ← dump_text + atom + "."
          end if
        end for
        if false then
          return Status_Continue
        end if
      end if
    end while
    return Status_Not_Continue
  end procedure
```

Algorithm 16 Rule Execution 特殊化２：propagation_forward

```
Require: rule = propagation_forward
Require: target = between(A,!Y,B)
Require: Head = distance(C,!Y,!Z,D)
Require: Body = between(AC,!Y,BD)
Require: hist : tuple(A,!Y,B,C,!Z,D) を key とする hash table
  procedure EXEC_RULE(rule, target)
    while true do
      pair_0 ← distance_vec_2[ID of !Y].next()      ▷ .next() fetches deque elements from front to end
      if pair_0 の取得に失敗 then                                          ▷ deque iteration is complete
        break
      end if
      if hist.find((A,!Y,B,C,!Z,D)) == false then
        hist.insert((A,!Y,B,C,!Z,D))
        AC ← A + C
        BD ← B + D
        between_p.push_back(between(AC,!Z,BD))
      end if
    end while
    return Status_Not_Continue
  end procedure
```

Algorithm 17 Rule Execution 特殊化 2 ：propagation_forward

```
Require: rule = propagation_forward
Require: target = distance(C,!Y,!Z,D)                    ▷ target が入れ替わっても同様
Require: Head = between(A,!Y,B)
Require: Body = between(AC,!Y,BD)
Require: hist : tuple(A,!Y,B,C,!Z,D) を key とする hash table
  procedure EXEC_RULE(rule, target)
    while true do
      pair_0 ← between_vec_2[Id of !Y].next()    ▷ .next() fetches deque elements from front to end
      if pair_0 の取得に失敗 then                              ▷ deque iteration is complete
        break
      end if
      if hist.find((A,!Y,B,C,!Z,D)) == false then
        hist.insert((A,!Y,B,C,!Z,D))
        AC ← A + C
        BD ← B + D
        between_p.push_back(between(AC,!Z,BD))
      end if
    end while
    return Status_Not_Continue
  end procedure
```

0.2 例題の定義

0.2.1 LMNtal プログラム

プログラムには，初期状態として，1 つのハイパーリンクを引数として持つアトム start_1 1 つと，整数，ハイパーリンク，ハイパーリンク，整数の組を引数として持つアトム distance_4 がいくつか与えられる．

```
1  start@@
2  start(!X) :- between(0,!X,0).
3
4  max@@ R = max(A,B) :- A > B | R = A.
5  max@@ R = max(A,B) :- A =< B | R = B.
6  min@@ R = min(A,B) :- A > B | R = B.
7  min@@ R = min(A,B) :- A =< B | R = A.
8
9  inconsistent@@
10 between(A,!X,B) :- A > B | fail(A,!X,B).
11
12 intersect@@
13 between(A,!Y,B), between(C,!Y,D) :- between(max(A,C),!Y,min(B,D)).
14
15 propagate_forward@@
16 between(A,!Y,B), distance(C,!Y,!Z,D) \
17     :- AC=A+C, BD=B+D, uniq(A,!Y,B,C,!Z,D)
18     | between(AC,!Z,BD).
19
20 propagate_backward@@
21 between(A,!Y,B), distance(C,!Z,!Y,D) \
22     :- AC=A-D, BD=B-C, uniq(A,!Y,B,C,!Z,D)
23     | between(AC,!Z,BD).
```

0.2.2 各アトムとプログラムの数学としての意味

プログラム内には，between_3, distance_4 が，主要なアトムとして出現し，引数に含まれるハイパーリンクは，整数上の変数を意味する．between_3 の引数は，整数 a ，ハイパーリンク H，整数 b の組であり，$a \leq H \leq b$ を意味する．distance_4 の引数は，整数 a, ハイパーリンク H_1，ハイパーリンク H_2，整数 b の組であり， $a \leq H_2 - H_1 \leq b$ を意味する．

$min \leq H_1 \leq max$ かつ $a \leq H_2 - H_1 \leq b$ であるとき, $a+min \leq H_2 \leq b+max$ がわかる．また, $min \leq H_2 \leq max$ かつ $a \leq H_2 - H_1 \leq b$ であるとき, $max-b \leq H_1 \leq min-a$ がわかる．

ある変数の集合 V を考える．今，$0 \leq H_0 \leq 0$ where $H_0 \in V$ を表す `between_3` アトムが 1 つと，$a \leq H_j - H_i \leq b$ where $H_i \in V, H_j \in V, a \in \mathbb{Z}, b \in \mathbb{Z}$ を表す `distance4` アトムが複数与えられるとする．

このとき，計算可能な全ての変数 $H \in V$ について，$min \leq H \leq max$ を計算する．

0.2.3 このプログラムの実用例題的な意味（問題文）

あなたは部署の依存関係のあるイベントの時間管理を任されました．あなたには，複数あるイベントの内 2 つのイベントについて，Event_j が必ず Event_i の a 分後から b 分後までの間に実行されるという関係が複数知らされます．また，どのイベントが最初に行われるべきかということが知らされます．最初に行われるべきイベントの時間をイベントの開始後 0 分後 から 0 分後であるとすると，各イベントは開始後何分後から何分後に実行されるでしょうか．

0.3 アルゴリズムメモ

定義 4 (検索されうる引数)**.** atom($arg_0, \ldots, arg_n$) を Head に含み，かつ，いずれかの辺の式が $arg_i, 0 \leq i \leq n$ を含むような, 比較を行う二項演算子 op を Guard に含むようなルールが存在する場合，atom の第 i 引数は，**op によって検索されうる**（i th argument of atom can be searched by op）.

定義 5 (labeled rule)**.** これ以降，LMNtal ルールにおいて，Head :- Guard — Body. もしくは $\text{Head}_1 \backslash \text{Head}_2$:- Guard — Body. のように記述された全てのルールについて，Head に出現するある 1 つのアトム（atom($arg_0, \ldots, arg_n$)）にラベル付けしたルール (labeled rule) をラベル付きルール（もしくは単にルール, rule）と呼ぶ．ラベル付けされたアトムをターゲットアトム（target_atom） と呼び，Head に出現する全てのアトムについてラベル付きルールが生成される.

定義 6 (quated atom)**.** アトム atom($arg_0, \ldots, arg\_n$) について，”atom” もしくは ”atom($arg_0, \ldots, arg\_n$)” と表記したとき，そのアトムの名前を表す文字列を意味する

example

LMNtal ルール `a(A,B),b(C,D) :- A=C, B<D | .` が存在するとき，ラベル付けされたルール `label:a(A,B), b(C,D) :- A=C, B<D | .`, `a(A,B), label:b(C,D) :- A=C, B<D | .` が存在する.

主に HyperLink （id を表すような増減しない整数値）が同一であることを確かめるケースを考えるためここでは ”=” によって検索されうる引数について考える.
他の演算子によって検索されうる引数については今後の課題とする（が，残っているのは int の比較だと思うので，普通に 2 分探索木を考えている）. 例題がないので考えるモチベーションがないだけである.

例

図 0.3 のプログラムにおいて，ルールのヘッドに存在するアトムは,

- start(HyperLink)
- max(Int, Int, Link)
- min(Int, Int, Link)
- between(Int, HyperLink, Int)

Algorithm 18 全体のアルゴ memo

```
Require:                                                    ▷ "atom" ∈ set of all atoms(functors)
 1: "atom"_p is a list of "atom"s that have not been inserted into atomlist yet.
 2: "atom"_al is a list of "atom"s that have already been inserted and can be used as part of the Head of
    rules.
 3: "atom"_vec_i is a vector (or array) of vector (or list) of pointer to "atoms"s where "atom"_vec_i[j] is
    vector of pointer to the atom that HL_id(or Int_id) of i_th argument is j    ▷ Such vectors are exist if
    and only if i th argument of "atom" can be searched by "="
Ensure: All "atom"_p are empty.
 4: procedure 全体の処理
 5:     for all atom ∈ set_non_increasable_atoms do
 6:         if !("atom"_p is empty) then
 7:             target_atom ← "atom"_p.front()
 8:             "atom"_p_pop_front()
 9:             for all Rule ← Rules_Head_include_target_atom do
10:                 if Exec_Rule(Rule, target_atom) == Status_Continue then
11:                     Continue
12:                 end if
13:                 break
14:             end for
15:             "atom"_al.push_back(target_atom)
16:             for all search_vec: "atom"_vec_0 . . . "atom"_vec_n do
17:                 search_vec.push_back("atom"_al.last())        ▷ .last() gets iterator to last element of list
                                                                    or vector
18:             end for
19:         end if
20:     end for
21:     while !(atom_p is empty for all atoms) do
22:         for all atom ∈ set_increasable_atoms do
23:             if !(atom_p is empty) then
24:                 target_atom ← atom_p.front()
25:                 atom_p.pop_front()
26:                 for all Rule ← Rules_Head_include_target_atom do
27:                     if Exec_Rule(Rule, target_atom) == Status_Continue then
28:                         Continue
29:                     end if
30:                     break
31:                 end for
32:                 "atom"_al.push_back(target_atom)
33:                 for all search_vec: "atom"_vec_0 . . . "atom"_vec_n do
34:                     search_vec.push_back("atom"_al.last())    ▷ .last() gets iterator to last element of list
                                                                    or vector
35:                 end for
36:             end if
37:         end for
38:     end while
39: end procedure
```

- distance(Int, HyperLink, HyperLink, Int)

である．尚，intersect ルールにおいて生成される `max(Int, Int, Link)`, `min(Int, Int, Link)` は生成時に `Link=Int` に書き換える．そのため，初期状態にこれらが含まれない場合は，max, min についてのアトムリスト等は生成されない．ここでは簡単のため，max, min は初期状態に含まれないものとする．また，"=" によって検索されうる引数は以下である．(HyperLink が等しいことを確かめるオペレータは "=" であるとする)

```
Require:                                                    ▷ these information are known at compile time
 1: "atom"_vec_i is a vector (or array) of vector (or list) of pointer to "atoms"s where "atom"_vec_i[j] is
    vector of pointer to the atom that HL_id(or Int_id) of i_th argument is j    ▷ Such vectors are exist if
    and only if i th argument of "atom" can be searched by "="
 2: input rule is a labeled rule
 3: procedure EXEC_RULE(rule, target_atom)
 4:     set S = target_atom
 5:     set Head ← Head of rule
 6:     atom_0(arg_1, ..., arg_n) ∈ Head
 7:     Head = Head \ {atom_0(arg_1, ..., arg_n)}
 8:     ...
 9:     atom_l(arg_1, ..., arg_m) ∈ Head
10:     Head = Head \ {atom_l(arg_1, ..., arg_m)}                          ▷ Head must be ∅ here
11:     loop
12:         ... (l)
13:         loop
14:             if arg_i can be searched by "=" and ∀link ∈ L(∃atm ∈ S(∃a ∈ args of atm(link = a))) where
                  0 ≤ i ≤ m and L is set of links appears as operands of the "=" except for arg_i
    then
15:                 expr ← transform the expression into the form arg_i "=" others
16:                 j ← result of the right side of expr
17:                 if atom_vec_i[j] is empty then
18:                     return Status_Not_Continue
19:                 else
20:                     pair_atom_l ← atom_vec_i[j].next()        ▷ .next() if it is called first time, gets
                                                                    first element; otherwise, gets next ele-
                                                                    ment(including the next of the last el-
                                                                    ement)
21:                     if pair_atom_l = atom_vec_i[j].end() then ▷ .end() means the next of the last element
22:                         break
23:                     end if
24:                 end if
25:             else
26:                 pair_atom_l ← atom_atomlist.next()
27:             end if
28:             if Guard is true then
29:                 for all body_atom ∈ Body of rule do
30:                     "body_atom"_p.push_back(body_atom)
31:                     delete atoms
32:                     if target_atom is deleted then
33:                         return Status_Continue
34:                     end if
35:                 end for
36:             end if
37:         end loop
38:     end loop
39:     return Status_Not_Continue
40: end procedure
```

- intersect 他より， between(Int, HyperLink, Int) の第 2 引数
- propagate_forward より，distance(Int, HyperLink, HyperLink, Int) の第 2 引数
- propagate_backward より, distance(Int, HyperLink, HyperLink, Int) の第 3 引数

以上の事から，疑似コード 18 において用意されるデータ構造は以下である.

- start_p
- start_al

```
1  start@@
2  start(!X) :- between(0,!X,0).
3
4  max@@ R = max(A,B) :- A > B | R = A.
5  max@@ R = max(A,B) :- A =< B | R = B.
6  min@@ R = min(A,B) :- A > B | R = B.
7  min@@ R = min(A,B) :- A =< B | R = A.
8
9  inconsistent@@
10 between(A,!X,B) :- A > B | fail(A,!X,B).
11
12 intersect@@
13 between(A,!Y,B), between(C,!Y,D) :- between(max(A,C),!Y,min(B,D)).
14
15 propagate_forward@@
16 between(A,!Y,B), distance(C,!Y,!Z,D) \
17     :- AC=A+C, BD=B+D, uniq(A,!Y,B,C,!Z,D)
18     | between(AC,!Z,BD).
19
20 propagate_backward@@
21 between(A,!Y,B), distance(C,!Z,!Y,D) \
22     :- AC=A-D, BD=B-C, uniq(A,!Y,B,C,!Z,D)
23     | between(AC,!Z,BD).
```

- between_p
- between_al
- between_vec_2
- distance_p
- distance_al
- distance_vec_2
- distance_vec_3

全体のアルゴリズム**??**において，line 5 の for loop で処理されるアトムは `start(HyperLink)` と `distance(Int, HyperLink, HyperLink, Int)` であり，それに伴い実行されるルールは以下である．

- ターゲットアトムが `start(!X)` である start ルール
- ターゲットアトムが `distance(C,!Y,!Z,D)` である，propagation_forward ルール
- ターゲットアトムが `distance(C,!Z,!Y,D)` である，propagation_backward ルール

その他のアトム，つまり `between(Int, HyperLink, Int)` (注: max, min は考えない

ことを思い出してください) については，line 21 の while loop 内で処理される．具体的な残りのルールは以下である

- ターゲットアトムが `between(A,!X,B)` である inconsistent ルール
- ターゲットアトムが `between(A,!Y,B)` である intersect ルール
- ターゲットアトムが `between(C,!Y,D)` である intersect ルール
- ターゲットアトムが `between(A,!Y,B)` である propagation_forward ルール
- ターゲットアトムが `between(A,!Y,B)` である propagation_backward ルール

ここで，exec_rule0.3 について，ターゲットアトムが `between(A,!Y,B)` であったときの propagation_forward ルールについての例を挙げる．まず，ターゲットアトム以外にヘッドに含まれるアトムは `distance(C,!Y,!Z,D)` ただ 1 つであるため， line 6-10 において $l = 0$ である．そのため，line 11 以降の loop は 1 重である．次に line 14 の if 文について，`between(A,!Y,B)` をターゲットアトムとして持っているため，`distance(C,!Y,!Z,D)` 側の `!Y` は arg_2 から検索可能で，かつ ”=” のオペランドとして出現するリンク，つまり `between(A,!Y,B)` 側の `!Y` がわかっているため，条件式は `true` を返す．その後，

- line 15: expr ← `!Y` の HyperLink ID
- line 16: j ← expr

となり, distance_vec_2[j] のベクタを得る．要素数が 0 の場合 line 17-18 によって return しルール実行が終了する．要素がある場合, line 20 によって先頭から順に要素を得て, line 28 でガード条件を処理する．その後、line 29-35 で between_p に `between(AC,!Z,BD)` を push する．なお，ターゲットアトムが削除されないため，line 32-33 は条件が false なので実行されない．loop を繰り返し，line 20 の処理で要素を最後まで辿ったら line 22-23 によりループを抜け，line 39 により Status_Not_Continue を返して終了．なお， Status_Not_Continue を受け取った アルゴリズム 18 line は，ターゲットアトム `between(A,!Y,B)` を between_al に push し，処理を続ける．Status_Continue はターゲットアトムが削除される場合に返され，アトムリストに push することなく処理を続ける．

- ”atom”_p is deque
- ”atom”_al is deque
- ”atom”_vec_i is Index structure of deque of pointer to element of ”atom”_al

Algorithm 19 例題に特殊化したアルゴ memo

```
 1: procedure 全体
 2:     while true do
 3:         if start_p is empty then
 4:             break
 5:         end if
 6:         target ← start_p.front()
 7:         ret ← Exec start(target)
 8:         if ret == Stat_Continue then
 9:             continue
10:         end if
11:         start_al.push_back(target)                          ▷ never reach here
12:     end while
13:     while true do
14:         if distance_p is empty then
15:             break
16:         end if
17:         target ← distance_p.front()
18:         distance_p.pop_front()
19:         ret ← Exec propagate_forward(target)
20:         if ret == Stat_Continue then
21:             continue                                        ▷ never reach here
22:         end if
23:         ret ← Exec propagate_backward(target)
24:         if ret == Stat_Continue then
25:             continue                                        ▷ never reach here
26:         end if
27:         distance_al.push_back(target)
28:         idx_2nd ← ID of 2nd argument of target
29:         idx_3rd ← ID of 3rd argument of target
30:         distance_vec_2[idx_2nd].push_back(&distance_al.back())
31:         distance_vec_3[idx_3rd].push_back(&distance_al.back())
32:     end while
33:     while true do
34:         if between_p is empty then
35:             break
36:         end if
37:         target ← between_p.front()
38:         between_p.pop_front()
39:         ret ← Exec inconsistent(target)
40:         if ret == Status_Continue then
41:             continue                                        ▷ if the rule fired
42:         end if
43:         ret ← Exec intersect(target)
44:         if ret == Status_Continue then
45:             continue                                        ▷ if the rule fired
46:         end if
47:         ret ← Exec propagate_forward(target)
48:         if ret == Status_Continue then
49:             continue                                        ▷ never reach here
50:         end if
51:         ret ← Exec propagate_backward(target)
52:         if ret == Status_Continue then
53:             continue                                        ▷ never reach here
54:         end if
55:         between_al.push_back(target)
56:         idx_2nd ← ID of 2nd argument of target
57:         between_vec_2[idx_2nd].push_back(&between_al.back())
58:     end while
59: end procedure
```

Algorithm 20 全体のアルゴリズム簡略版

```
Require: A はルールのヘッドに出現するアトムの集合
Require: Rule はラベル付きルールの集合
  procedure 全体
    while ∃a ∈ A, "a"_p is not empty do
      for all atom ∈ A do
        Exec_Insert(atom)                    ▷ Body に一度も出現しないアトムは while の外にくくりだせる
      end for
    end while
  end procedure
```

Algorithm 21 全体のアルゴリズム特殊化

```
  procedure 全体
    Exec_Insert(start(HL))
    Exec_Insert(distance(Int,HL,HL,Int))
    while ∃deq ∈ {between_p}, deq is not empty do
      Exec_Insert(between(Int,HL,Int))
    end while
  end procedure
```

Algorithm 22 Insertion Loop

```
Require: atom : 入力, アトム名と引数の型の組
Require: Rule : atom がターゲットアトムであるルールの集合
  procedure EXEC_INSERT(atom)
    while "atom"_p is not empty do        ▷ recall: "a" は変数 a に格納されたアトムのアトム名の文字列表現
      target ← "atom"_p.front()
      "atom"_p.pop_front()
      for all rule ∈ Rule do
        ret ← Exec_Rule(rule, target)
        if ret == Status_Continue then                    ▷ If target atom is deleted
          Flag ← true
          berak
        end if
      end for
      if Flag then
        continue
      end if
      "atom"_al.push_back(target)
    end while
  end procedure
```

Algorithm 23 Insertion Loop 特殊化：start(HL)

```
Require: atom = start(HL)
Require: Rule = {start }
  procedure EXEC_INSERT(atom)
    while start_p is not empty do
      target ← start_p.front()
      start_p.pop_front()
      for all rule ∈ Rule do                                  ▷ start ルール
        ret ← Exec_Rule(rule, target)
        if ret == Status_Continue then                        ▷ If target atom is deleted
          Flag ← true
          berak
        end if
      end for
      if Flag then
        continue
      end if
      start_al.push_back(target)
    end while
  end procedure
```

Algorithm 24 Insertion Loop 特殊化：between(Int,HL,Int)

```
Require: atom = between(Int,HL,Int)
Require: Rule = {inconsistent, intersect, propagate_forward, propagate_backward }
  procedure EXEC_INSERT(atom)
    while between_p is not empty do
      target ← between_p.front()
      between_p.pop_front()
      for all rule ∈ Rule do
        ret ← Exec_Rule(rule, target)
        if ret == Status_Continue then                        ▷ If target atom is deleted
          Flag ← true
          berak
        end if
      end for
      if Flag then
        continue
      end if
      between_al.push_back(target)
    end while
  end procedure
```

Algorithm 25 Insertion Loop 特殊化：distance(Int,HL,HL,Int)

```
Require: atom = distance(Int,HL,HL,Int)
Require: Rule = {propagate_forward, propagate_backward }
  procedure EXEC_INSERT(atom)
    while distance_p is not empty do
      target ← distance_p.front()
      distance_p.pop_front()
      for all rule ∈ Rule do
        ret ← Exec_Rule(rule, target)
        if ret == Status_Continue then                    ▷ If target atom is deleted
          Flag ← true
          berak
        end if
      end for
      if Flag then
        continue
      end if
      distance_al.push_back(target)
    end while
  end procedure
```

Algorithm 26 Rule Execution, Head size = 1

```
Require: rule : target をターゲットアトムとして使用できるルール
Require: target : ターゲットアトム
Require: Head : rule の Head に含まれるターゲットアトム以外のアトム
Require: Rody : rule の Body に含まれるアトム
  procedure EXEC_RULE(rule, target)
    case |Head| = 0
    if Guard of rule is true then
      for all atom ∈ Rody do
        if "atom"_p exists then
          "atom"_p.push_back(atom)
        else
          dump_text ← dump_text + atom + "."
        end if
      end for
      if target is deleted by this rule then
        return Status_Continue
      end if
    end if
    return Status_Not_Continue
  end procedure
```

Algorithm 27 Rule Execution, Head size = 2

Require: rule : target をターゲットアトムとして使用できるルール
Require: target : ターゲットアトム
Require: Head : rule の Head に含まれるターゲットアトム以外のアトム
Require: Body : rule の Body に含まれるアトム
Require: hist : uniq の引数（tuple）が key である hash table ▷ uniq が Guard に存在する場合

```
procedure EXEC_RULE(rule, target)
    case |Head| = 1
    while true do
        if pair atom が ith arg = j で検索可能 then
            pair_0 ← "pair_0"_vec_i[j].next()        ▷ .next() fetches deque elements from front to end
        else
            pair_0 ← "pair_0"_al.next()
        end if
        if pair_0 の取得に失敗 then                    ▷ deque iteration is complete
            break
        end if
        if Guard of rule is true then
            for all atom ∈ Body do
                if "atom"_p exists then
                    "atom"_p.push_back(atom)
                else
                    dump_text ← dump_text + atom + "."
                end if
            end for
            if target is deleted by this rule then
                return Status_Continue
            end if
        end if
    end while
    return Status_Not_Continue
end procedure
```

Algorithm 28 Rule Execution 特殊化：start

Require: rule = start
Require: target = start(!X)
Require: Head =
Require: Body = between(0,!X,0)

```
procedure EXEC_RULE(rule, target)
    if true then                                          ▷ Guard doesn't exist
        for all atom ∈ Body do
            atom が連結グラフならできるだけ書き換える
            if between_p exists then                      ▷ between_p is deque for between(Int,HL,Int)
                between_p.push_back(atom)
            else                                          ▷ never reach here
                dump_text ← dump_text + atom + "."
            end if
        end for
        if target is deleted by this rule then            ▷ yes
            return Status_Continue
        end if
    end if
    return Status_Not_Continue
end procedure
```

Algorithm 29 Rule Execution 特殊化 2 ：start

```
Require: rule = start
Require: target = start(!H)
  procedure EXEC_RULE(rule, target)
    start_p.push_back(between(0,!X,0))
    return Status_Continue
  end procedure
```

Algorithm 30 Rule Execution 特殊化：inconsistent

```
Require: rule = inconsistent
Require: target = between(A,!X,B)
Require: Head =
Require: Body = fail(A,!X,B)
  procedure EXEC_RULE(rule, target)
    if A>B then
      for all atom ∈ Body do
        if fail_p exists then                          ▷ fail_p doesn't exist
          fail_p.push_back(atom)
        else                                           ▷ always reach here
          dump_text ← dump_text + atom + "."
        end if
      end for
      if target is deleted by this rule then           ▷ yes
        return Status_Continue
      end if
    end if
    return Status_Not_Continue
  end procedure
```

Algorithm 31 Rule Execution 特殊化 2 ：inconsistent

```
Require: rule = inconsistent
Require: target = between(A,!X,B)
  procedure EXEC_RULE(rule, target)
    if A>B then
      dump_text ← dump_text + fail(A,!X,B) + "."
      return Status_Continue
    end if
    return Status_Not_Continue
  end procedure
```

Algorithm 32 Rule Execution 特殊化：intersect

```
Require: rule = intersect
Require: target = between(A,!Y,B)        ▷ between(A,!Y,B), between(C,!Y,D) をすべて入れ替えたものも同様
Require: Head = between(C,!Y,D)
Require: Body = between(max(A,C),!Y,min(B,D))
  procedure EXEC_RULE(rule, target)
    while true do
      if true then                          ▷ between(C,!Y,D) が ith arg = j, i=2, j=id=HLID of !Y で検索可能
        pair_0 ← "between"_vec_2[id].next()        ▷ .next() fetches deque elements from front to end
      else
        pair_0 ← "pair_0"_al.next()
      end if
      if pair_0 の取得に失敗 then                                          ▷ deque iteration is complete
        break
      end if
      if true then
        for all atom ∈ Body do
          atom が連結グラフならできるだけ書き換える
          if "atom"_p exists then                                  ▷ = if atom is between(Int,HL,Int)
            "atom"_p.push_back(atom)                                          ▷ 常に between_p.push
          else                                                ▷ max, min が between の引数に残った場合
            dump_text ← dump_text + atom + "."
          end if
        end for
        if true then
          return Status_Continue
        end if
      end if
    end while
    return Status_Not_Continue
  end procedure
```

Algorithm 33 Rule Execution 特殊化 2 ：intersect

```
Require: rule = intersect
Require: target = between(A,!Y,B)
Require: Head = between(C,!Y,D)
Require: Body = between(max(A,C),!Y,min(B,D))
  procedure EXEC_RULE(rule, target)
    while true do
      pair_0 ← "between"_vec_2[id].next()        ▷ .next() fetches deque elements from front to end
      if pair_0 の取得に失敗 then                  ▷ deque iteration is complete
        break
      end if
      if A>C then
        L0 ← A
      else if A≤C then
        L0 ← C
      else
        L0 ← max(A,C)
      end if
      if B>D then
        L2 ← D
      else if B≤D then
        L2 ← B
      else
        L2 ← min(B,D)
      end if
      atom ← between(L0,!Y,L2)
      if "atom"_p exists then                    ▷ = if atom is between(Int,HL,Int)
        between_p.push_back(atom)
      else                                       ▷ max, min が between の引数に残った場合
        dump_text ← dump_text + atom + "."
      end if
      return Status_Continue
    end while
    return Status_Not_Continue
  end procedure
```

Algorithm 34 Rule Execution 特殊化：propagation_forward

```
Require: rule = propagation_forward
Require: target = between(A,!Y,B)
Require: Head = distance(C,!Y,!Z,D)
Require: Body = between(AC,!Y,BD)
Require: hist : tuple(A,!Y,B,C,!Z,D) を key とする hash table
  procedure EXEC_RULE(rule, target)
    while true do
      if true then                          ▷ distance(C,!Y,!Z,D) が ith arg = j, i=2, j=id=HLID of !Y で検索可能
        pair_0 ← distance_vec_2[id].next()          ▷ .next() fetches deque elements from front to end
      else
        pair_0 ← "pair_0"_al.next()
      end if
      if pair_0 の取得に失敗 then                                          ▷ deque iteration is complete
        break
      end if
      if hist.find((A,!Y,B,C,!Z,D)) == false then
        hist.insert((A,!Y,B,C,!Z,D))
        AC ← A + C
        BD ← B + D
        for all atom ∈ Body do
          if "atom"_p exists then
            "atom"_p.push_back(atom)
          else                                                          ▷ never reach here
            dump_text ← dump_text + atom + "."
          end if
        end for
        if false then
          return Status_Continue
        end if
      end if
    end while
    return Status_Not_Continue
  end procedure
```

Algorithm 35 Rule Execution 特殊化２：propagation_forward

```
Require: rule = propagation_forward
Require: target = between(A,!Y,B)
Require: Head = distance(C,!Y,!Z,D)
Require: Body = between(AC,!Y,BD)
Require: hist : tuple(A,!Y,B,C,!Z,D) を key とする hash table
  procedure EXEC_RULE(rule, target)
    while true do
      pair_0 ← distance_vec_2[ID of !Y].next()      ▷ .next() fetches deque elements from front to end
      if pair_0 の取得に失敗 then                                          ▷ deque iteration is complete
        break
      end if
      if hist.find((A,!Y,B,C,!Z,D)) == false then
        hist.insert((A,!Y,B,C,!Z,D))
        AC ← A + C
        BD ← B + D
        between_p.push_back(between(AC,!Z,BD))
      end if
    end while
    return Status_Not_Continue
  end procedure
```

Algorithm 36 Rule Execution 特殊化 2 ：propagation_forward

```
Require: rule = propagation_forward
Require: target = distance(C,!Y,!Z,D)                          ▷ target が入れ替わっても同様
Require: Head = between(A,!Y,B)
Require: Body = between(AC,!Y,BD)
Require: hist : tuple(A,!Y,B,C,!Z,D) を key とする hash table
  procedure EXEC_RULE(rule, target)
    while true do
      pair_0 ← between_vec_2[Id of !Y].next()      ▷ .next() fetches deque elements from front to end
      if pair_0 の取得に失敗 then                                  ▷ deque iteration is complete
        break
      end if
      if hist.find((A,!Y,B,C,!Z,D)) == false then
        hist.insert((A,!Y,B,C,!Z,D))
        AC ← A + C
        BD ← B + D
        between_p.push_back(between(AC,!Z,BD))
      end if
    end while
    return Status_Not_Continue
  end procedure
```