Say My Innate Capabilities in a Natural Language　3/3

30 September 2020

Kenzo Iwama

Number

[(←) (↓) (←) (↓)

“Write “ “ “. { ～～ } { } (next)

{ ・ }

]

[(←) (↓) (←) (↓)

“～～ as many as this“. { ✔ ・ } (next)

“1” { ・ ～～ } (next)

“2” { ・ ～～ } (next)

“ “ { ・ ～～ }

]

[(←) (↓) (←) (↓)

“Draw this number of ○” { ✔ ・ }

⇔ ["Draw ～” { ～～} { } (next) {～ }] (next)

“1” { ・ ○ ～～} (next)

“2” { ・ ○○ ～～} (next)

, , ,

“ “ { ・ , , , ○ ～～}

]

[(←) (↓) (←) (↓)

“Take that number of ～” { ✔ ・ ～～ , , , } (next)

“1” { ・ ✔ ～ , , , ～ } (next)

“2” { ・ ✔ ～ , , , ～～ } (next)

, , ,

“ “ { ・ ✔ ～ , , , , , , ～～ } (next)

]

[(←) (↓) (←) (↓)

“Jump this number of times” ✔ { • }

⇔ ["Jump” { ˆ } { }] (next)

“1” { • 1 ˆ } <same> (next)

“2” { • 2 ˆ } (next)

<same>

, , , { • • ^ } (next)

<same>

“ “ { • • ˆ }

<same>

]

[(←) (↓) (←) (↓)

“Choose a number”. { · · • · · · · , , , }

⇔ ["Choose ～” { , , , ～～ } { } (next) {～ , , , }]

(next)

“ “ { · · • · · · · , , , ～～ }

⇔ [" “ {· · • }]

]

[(←) (↓) (←) (↓)

“～～ number of ～” ✔ { • } (next)

⇔ [”～” { ～ }]

⇔ ["～～” { ～～} { }]

“1” { • ～ ～～ } <same> (next)

“2” { • ～ ～～ } (next)

<same>

, , , { • ～ ～～ } (next)

“ “ { • ～ ～～ } <same>

]

[(←) (↓) (←) (↓)

"∼∼ number" { ∼ }

⇔ ["∼∼" { ∼∼} { } (next) { } { }]

⇔ ["∼" {∼ }]

(next)

{ ∼ ∼∼ } {next}

{ • ∼ }

]

[(←) (↓) (←) (↓)

"∼∼ number" { ∼ }

⇔ ["∼∼" { ∼∼} { } (next) { } { }]

⇔ ["∼" {∼ }]

(next)

{ ∼ ∼∼ } {next}

{ 1 ∼ }

]

[(←) (↓) (←) (↓)

"∼∼ number" { ∼ }

⇔ ["∼∼" { ∼∼} { } (next) { } { }]

⇔ ["∼" {∼ }]

(next)

{ ∼ ∼∼ } {next}

{ 2 ∼ }

]

[(←) (↓) (→) (↓) “Count how many ～”. {～～ , , , }

(next)

“Write the number”.

⇔ ["Write ～” { ～～} (next) {～ }] (next)

“1” {✔～～ , , , } (next)

, , ,

“ ” { , , , ～～✔ } (next)

“ “ “ ～” { , , , ～～ } (next)

“Write ” “ “ { , , , ～～ ～～} (next)

⇔ [" “ { ・ }]

“ ” {～～ , , , ・ }]

[(←) (↓) (→) (↓) “Count the number of ～”. {～～ , , , } (next)

“1” {✔～～ , , , } (next)

, , ,

“ ” { , , , ～～✔ } (next)

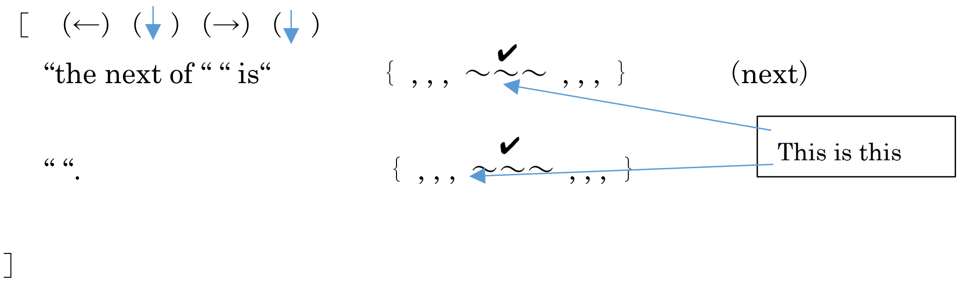
“The number of ～ is “ “ ” {～～ , , , }

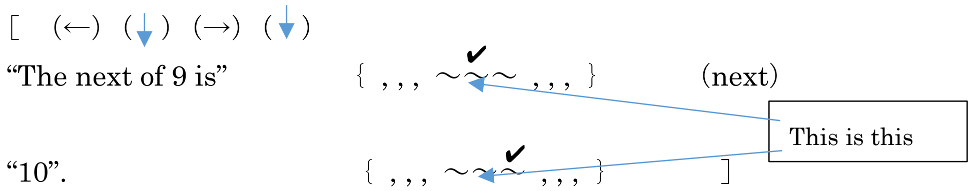
“Write the number” {～～ , , , ～～ } (next)

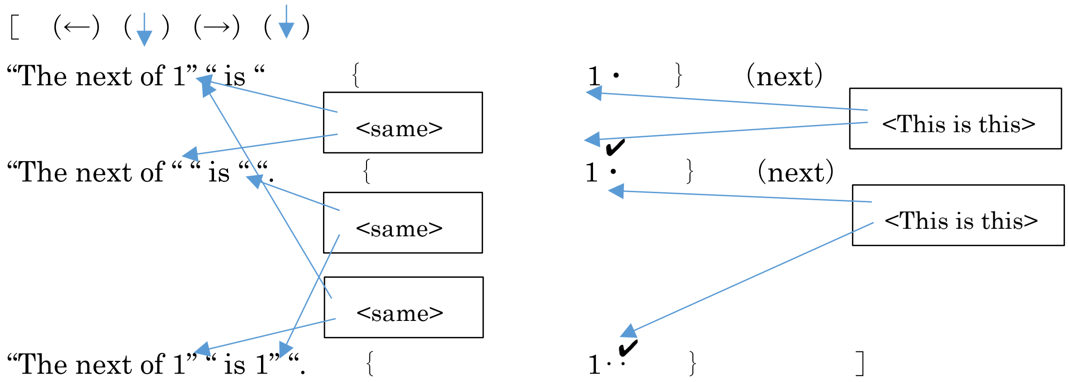
{～～ , , , ・ }

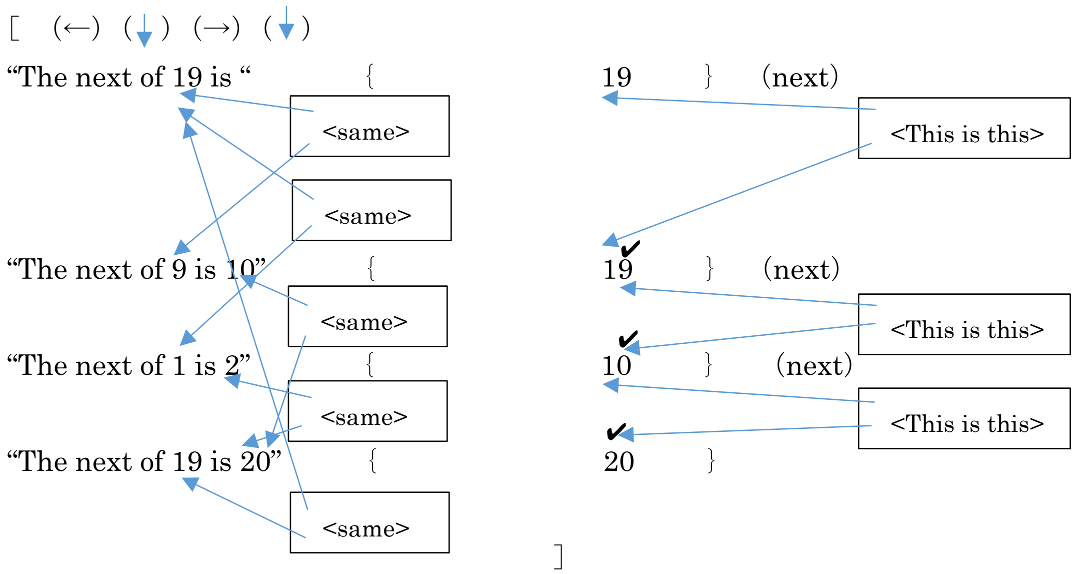
]

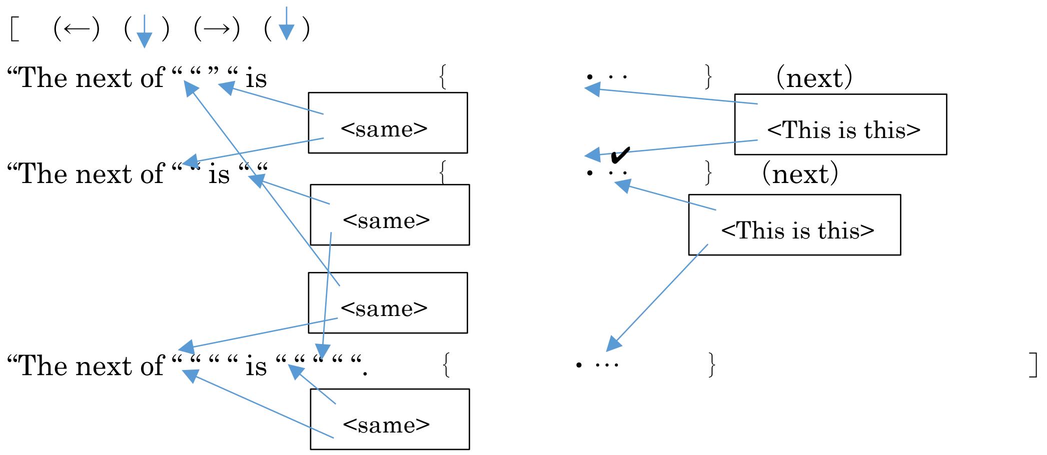
“～～ number ～” becomes a representative string bound to the representative of “～～ number ～” “1” {～ 1 } , “～～number ～” “2” {～ 2 } and the like. The program has “ “ { ・ } from the begibbibg and turns out to describe the representative of “1” {～ 1 } , “2” {～ 2 } and the like.


[(←) (↓) (→) (↓)
“the next of “ “ is“
{ , , , ∼∼∼ , , , }
(next)
This is this
“ “.
{ , , , ∼∼∼ , , , }
]


[(←) (↓) (→) (↓)
“The next of 9 is”
{ , , , ∼∼∼ , , , }
(next)
This is this
“10”.
{ , , , ∼∼∼ , , , }
]


[(←) (↓) (→) (↓)
“The next of 1” “ is “
{
1 ·
}
(next)
<same>
<This is this>
“The next of “ “ is “ “.
{
1 ·
}
(next)
<same>
<This is this>
<same>
“The next of 1” “ is 1” “.
{
1 · ·
}
]


[(←) (↓) (→) (↓)
“The next of 19 is “
{
19
}
(next)
<same>
<This is this>
<same>
“The next of 9 is 10”
{
19
}
(next)
<same>
<This is this>
“The next of 1 is 2”
{
10
}
(next)
<same>
<This is this>
“The next of 19 is 20”
{
20
}
<same>
]



[(←) (↓) (→) (↓)

“The next of “ “ ” “ is { • · · } (next)

<same>

<This is this>

“The next of “ “ is “ “ { • · · } (next)

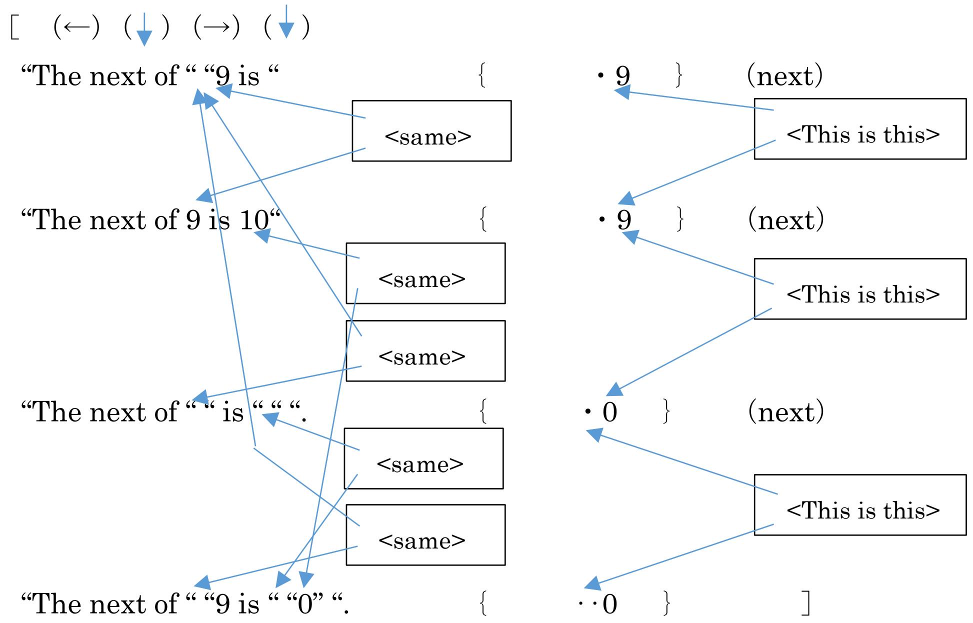
<same>

<This is this>

<same>

“The next of “ “ “ “ is “ “ “ “ “. { • · · · }]

<same>



[(←) (↓) (→) (↓)

“The next of “ “9 is “ { • 9 } (next)

<same>

<This is this>

“The next of 9 is 10“ { • 9 } (next)

<same>

<This is this>

<same>

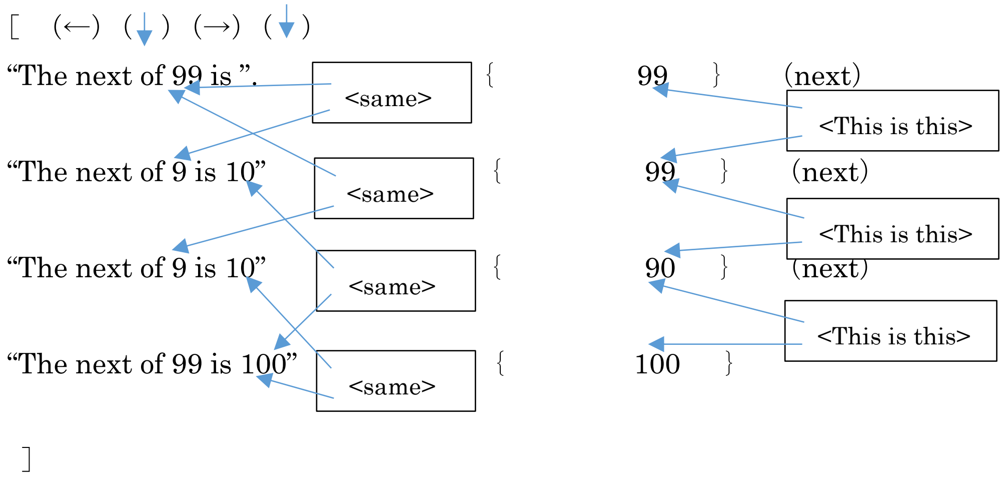
“The next of “ “ is “ “ “. { • 0 } (next)

<same>

<This is this>

<same>

“The next of “ “9 is “ “0” “. { · · 0 }]



[(←) (↓) (→) (↓)

“The next of 99 is ”. <same> { 99 } (next)

<This is this>

“The next of 9 is 10” <same> { 99 } (next)

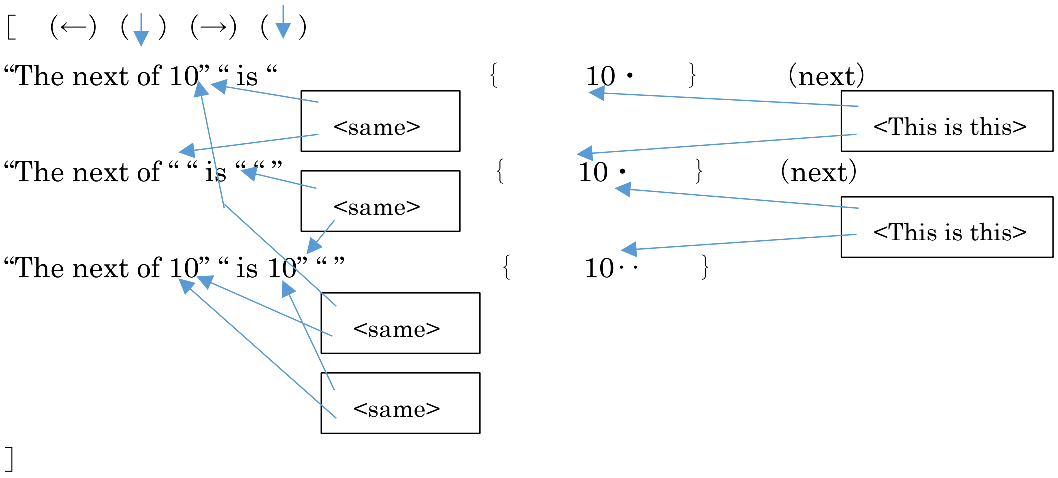
<This is this>

“The next of 9 is 10” <same> { 90 } (next)

<This is this>

“The next of 99 is 100” <same> { 100 }

]



[(←) (↓) (→) (↓)

“The next of 10” “ is “ { 10 • } (next)

<same> <This is this>

“The next of “ “ is “ “ ” { 10 • } (next)

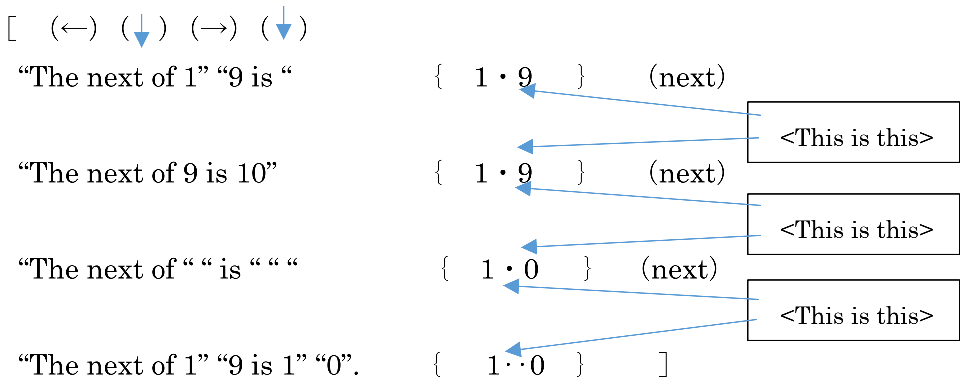
<same> <This is this>

“The next of 10” “ is 10” “ ” { 10 · · }

<same>

<same>

]



[(←) (↓) (→) (↓)

“The next of 1” “9 is “ { 1 • 9 } (next)

<This is this>

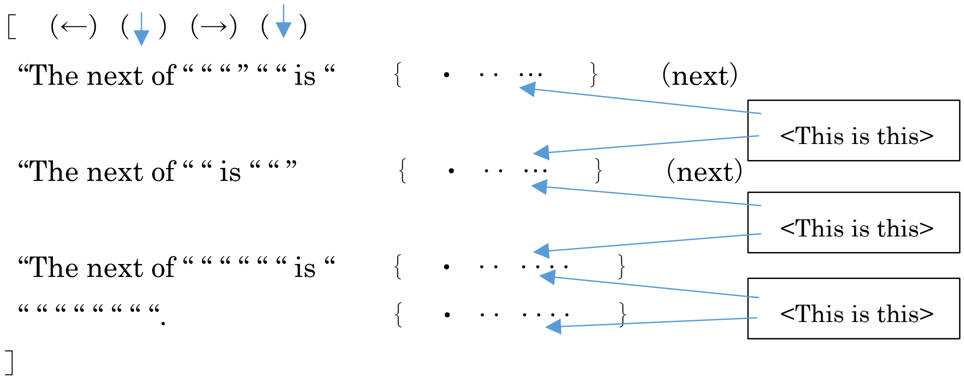
“The next of 9 is 10” { 1 • 9 } (next)

<This is this>

“The next of “ “ is “ “ “ { 1 • 0 } (next)

<This is this>

“The next of 1” “9 is 1” “0”. { 1 · · 0 }]



[(←) (↓) (→) (↓)

“The next of “ “ “ ” “ “ is “ { • · · · · · } (next)

<This is this>

“The next of “ “ is “ “ ” { • · · · · · } (next)

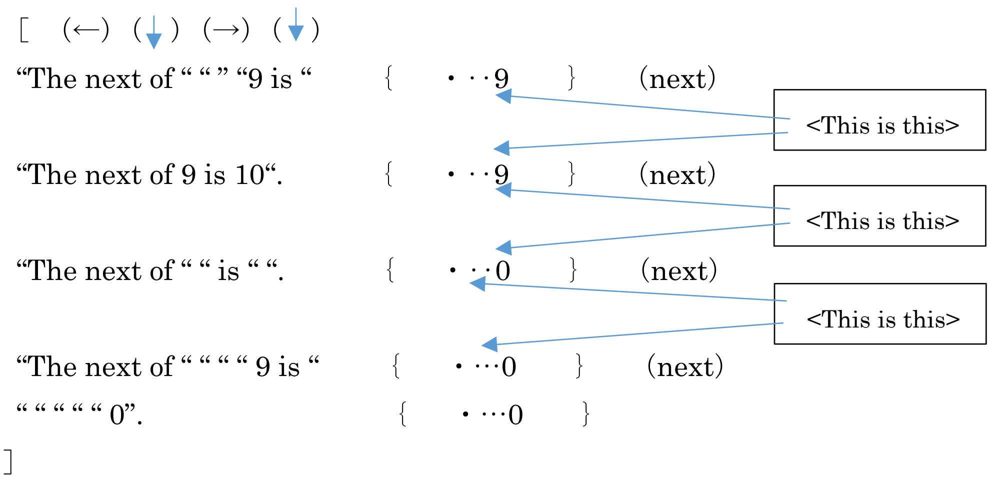
<This is this>

“The next of “ “ “ “ “ “ is “ { • · · · · · · }

<This is this>

“ “ “ “ “ “ “ “. { • · · · · · · }

]



[(←) (↓) (→) (↓)

“The next of “ “ ” “9 is “ { • · · 9 } (next)

<This is this>

“The next of 9 is 10“. { • · · 9 } (next)

<This is this>

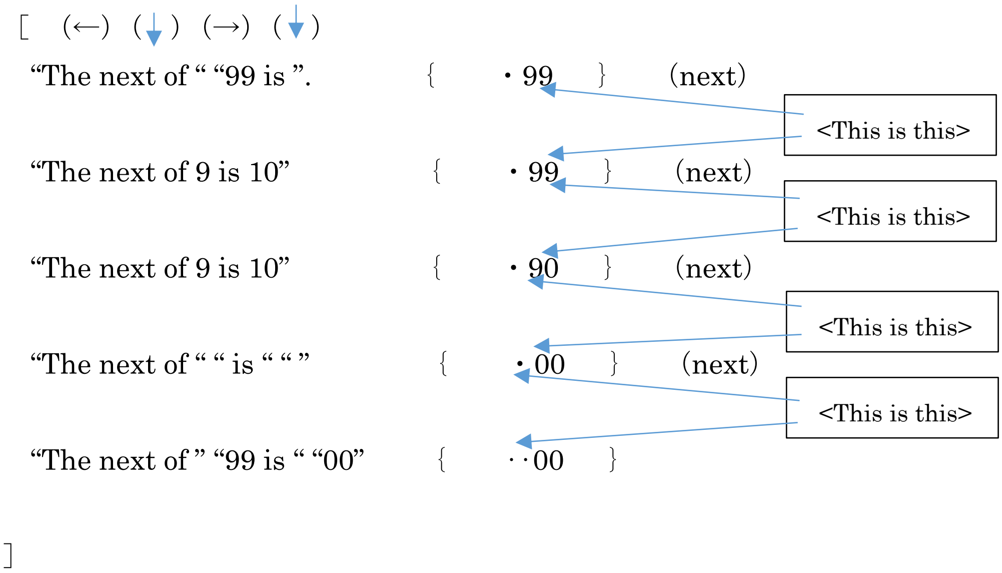
“The next of “ “ is “ “. { • · · 0 } (next)

<This is this>

“The next of “ “ “ “ “ 9 is “ { • ···0 } (next)

“ “ “ “ “ “ 0”. { • ···0 }

]



[(←) (↓) (→) (↓)

“The next of “ “99 is ”. { • 99 } (next)

<This is this>

“The next of 9 is 10” { • 99 } (next)

<This is this>

“The next of 9 is 10” { • 90 } (next)

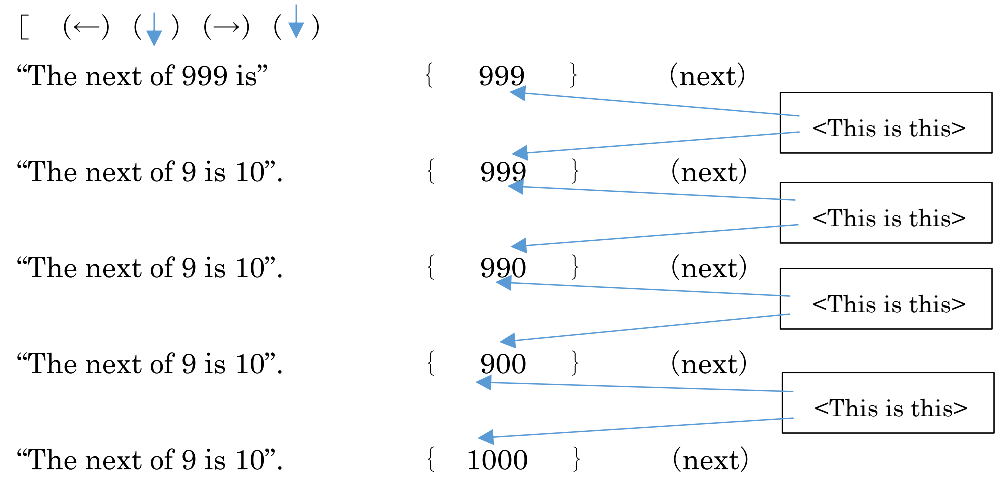
<This is this>

“The next of “ “ is “ “ ” { • 00 } (next)

<This is this>

“The next of ” “99 is “ “00” { · · 00 }

]



[(←) (↓) (→) (↓)

“The next of 999 is” { 999 } (next)

<This is this>

“The next of 9 is 10”. { 999 } (next)

<This is this>

“The next of 9 is 10”. { 990 } (next)

<This is this>

“The next of 9 is 10”. { 900 } (next)

<This is this>

“The next of 9 is 10”. { 1000 } (next)

“The next of 999 is “ { 1000 }
“1000“.]

[(←) (↓) (→) (↓)
“The next of “ “ , , , “ “ is “ { • , , , · · } (next)

<This is this>

“The next of “ “ is “ “ { • , , , · · } (next)

<This is this>

“The next of “ “ , , , “ “ is” { • , , , ··· }
“ “ “ , , , “ “ “.]

The set of the above regularity includes the next of a 3 digit number from 100 to 998.

[(←) (↓) (→) (↓)
“The next of “ “ 999 is “ { • 999 } (next)

<This is this>

“The next of 9 is 10” { • 999 } (next)

<This is this>

“The next of 9 is 10” { • 990 } (next)

<This is this>

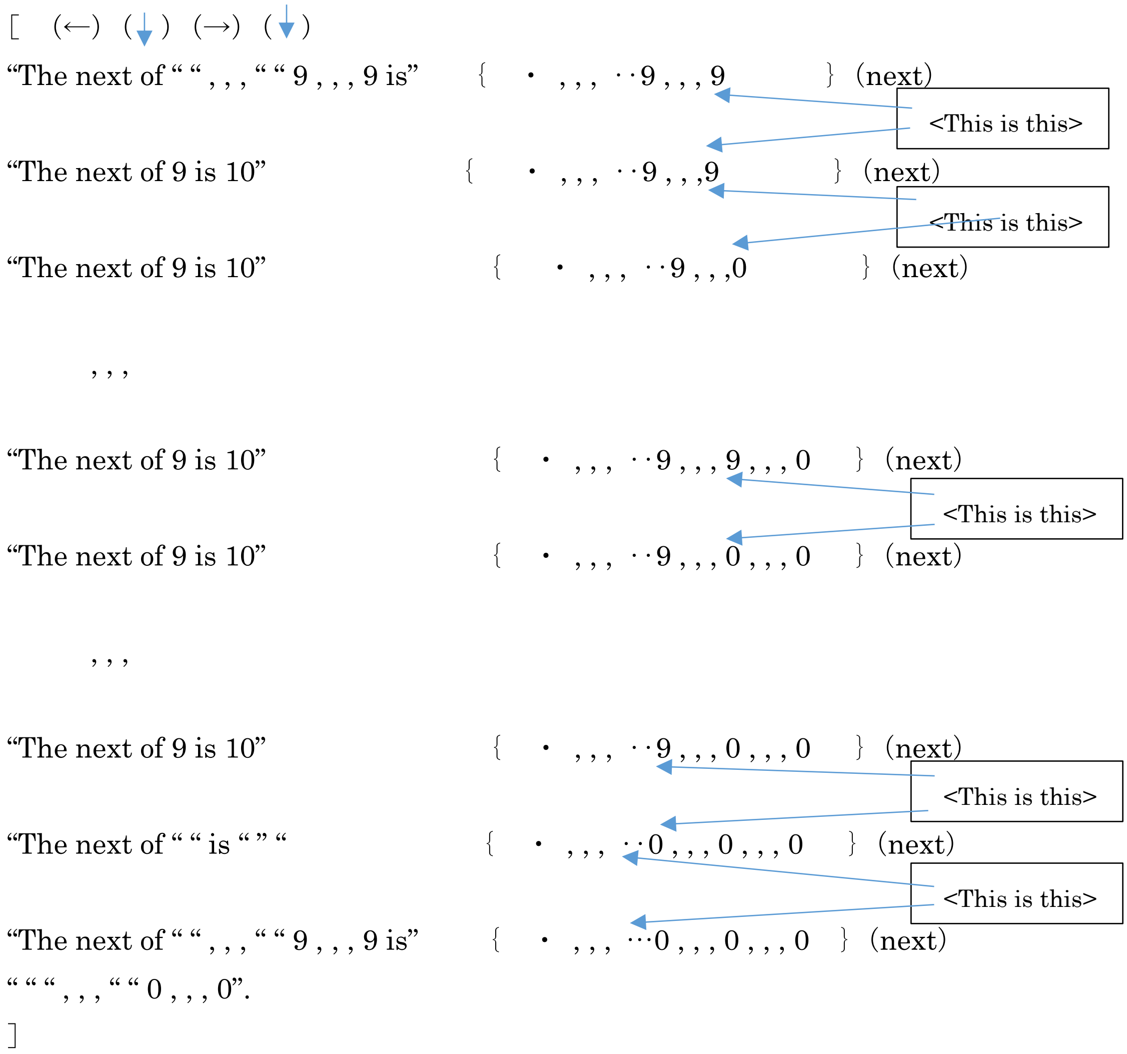
“The next of 9 is 10” { • 900 } (next)

<This is this>

“The next of “ “ is “ “ { • 000 } (next)

<This is this>

“The next of “ “999 is “ “000“. { · · 000 }]



[(←) (↓) (→) (↓)

“The next of “ “ , , , “ “ 9 , , , 9 is” { ・ , , , · · 9 , , , 9 } (next)

<This is this>

“The next of 9 is 10” { ・ , , , · · 9 , , ,9 } (next)

<This is this>

“The next of 9 is 10” { ・ , , , · · 9 , , ,0 } (next)

, , ,

“The next of 9 is 10” { ・ , , , · · 9 , , , 9 , , , 0 } (next)

<This is this>

“The next of 9 is 10” { ・ , , , · · 9 , , , 0 , , , 0 } (next)

, , ,

“The next of 9 is 10” { ・ , , , · · 9 , , , 0 , , , 0 } (next)

<This is this>

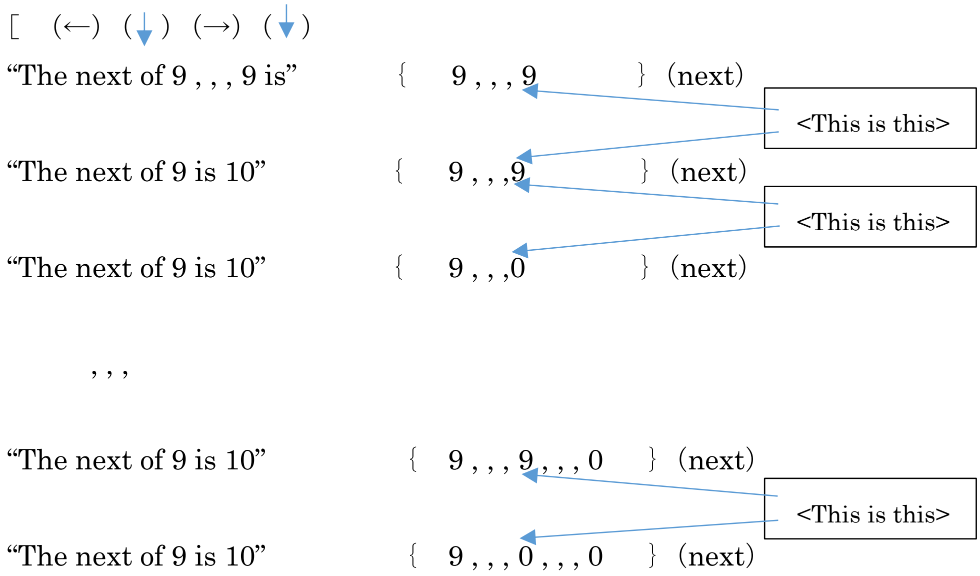
“The next of “ “ is “ ” “ { ・ , , , · · 0 , , , 0 , , , 0 } (next)

<This is this>

“The next of “ “ , , , “ “ 9 , , , 9 is” { ・ , , , ···0 , , , 0 , , , 0 } (next)

“ “ “ , , , “ “ 0 , , , 0”.

]



[(←) (↓) (→) (↓)

“The next of 9 , , , 9 is” { 9 , , , 9 } (next)

<This is this>

“The next of 9 is 10” { 9 , , ,9 } (next)

<This is this>

“The next of 9 is 10” { 9 , , ,0 } (next)

, , ,

“The next of 9 is 10” { 9 , , , 9 , , , 0 } (next)

<This is this>

“The next of 9 is 10” { 9 , , , 0 , , , 0 } (next)

, , ,

“The next of 9 is 10” { 9 , , , 0 , , , 0 } (next)

<This is this>

“The next of 9 is 10” { 0 , , , 0 , , , 0 } (next)

<This is this>

“The next of 9 is 10 , , , 0” { 10 , , , 0 , , , 0 } (next)

]

A representative of its member such as “The next of 7 is 8“ is “The next of { • } (↓) (→) is { • } (↓) (→) ”, and the representative links “The next of 5 is 6“, “The next of 6 is 7“, “The next of 7 is 8“, and others.

[(←) (↓) (→) (↓) “This number is 1 greater than “ { ✔• •• }
“that number”. { • ✔•• }
“The next of that number is “ “ “. { • •• ✔• }
“ “ “ is this number”. { ✔• •• • }]

[(←) (↓) (→) (↓) “This number is 2 greater than “ { ✔• 2 •• }
“that number”. { • 2 ✔•• }
“The next of that number is “ “ “. { • 2 •• ✔••• }
“The next of “ “ is “ “ “. { • 2 •• ••• ✔• }
“ “ “ is this number”. { ✔• 2 •• ••• • }]

[(←) (↓) (→) (↓)
"This number is “ { ✔• •• … }
“ “ “ greater than “ { • ✔•• … }
“that number”. { • •• ✔… }
“1 greater is “ “ “. { • 1 •• … ✔• • }
“2 greater is “ “ “. { • 2 •• … • • ✔••• }

, , ,

“ “ “ greater is “ “ “ { • •• •• ··· • • • •• , , , ✓• }

“this number”. { ✓• •• •• ··· • • • •• , , , • }]

[(←) (↓) (→) (↓)

"This number is “ { ✓• •• ··· }

“ “ “ less than “ { • ✓•• ··· }

“that number”. { • •• ✓··· }

“The next of this number is” { ✓• 1 •• ··· }

“ “. { • ✓•·· 1 •• ··· }

“The next of this number is” { • ✓•·· 2 •• ··· }

“ “. { • ✓···· 2 •• ··· }

<same>

“The next of this number is” { • ✓•··· •• •• ··· }

“ “. { • ✓··· •• •• ··· }

"This number is “ { ✓• ··· •• •• ··· }

“ “ “ less than “ { • ··· ✓•• •• ··· }

“that number”. { • ··· •• •• ✓··· }

]

[(←) (↓) (→) (↓)

“There are two groups of ○”. { ○○ , , , ○○ , , , }

“Which one is greater than the other. (next)

“1” { ✓○○ , , , ○○ , , , }

“2“ { ○✓○ , , , ○○ , , , }

, , ,

“ “ { , , , ○✓○ ○○ , , , }

“1” { ○○ , , , ✓○○ , , , }

“2” { ○○ , , , ○✓○ , , , }

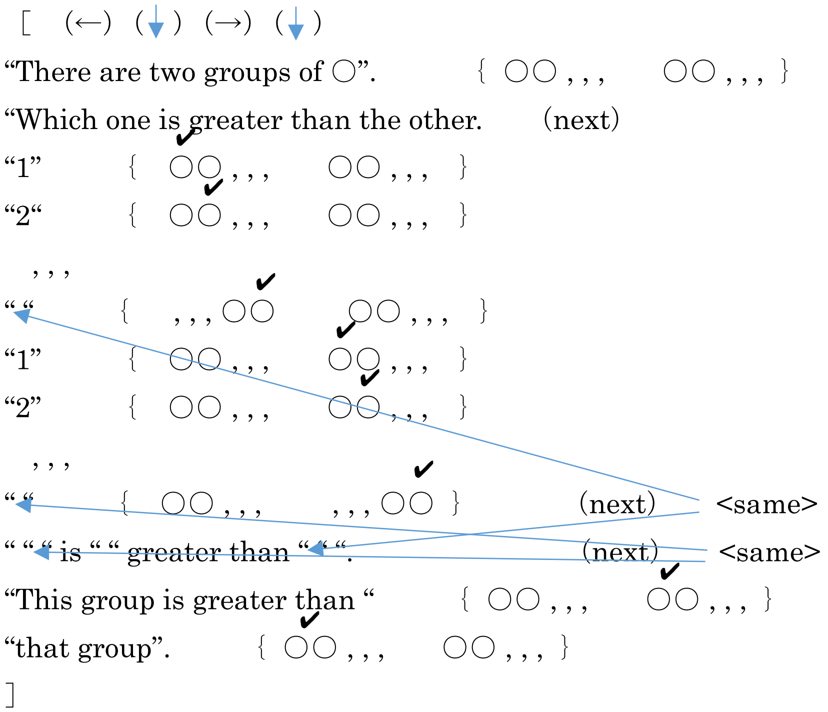
, , ,

“ “ { ○○ , , , , , , ○✓○ } (next) <same>

“ “ “ is “ “ greater than “ “ “. (next) <same>

“This group is greater than “ ✓{ ○✓○ , , , ○○ , , , } "that group". { ○○ , , , ○○ , , , }

]



[(←) (↓) (→) (↓)

"There are two groups of ○". { ○○ , , , ○○ , , , }

"Which one is greater than the other. (next)

"1" { ✓○○ , , , ○○ , , , }

"2" { ○✓○ , , , ○○ , , , }

, , ,

" " { , , , ○✓○ ○○ , , , }

"1" { ○○ , , , ✓○○ , , , }

"2" { ○○ , , , ○✓○ , , , }

, , ,

" " { ○○ , , , , , , ○✓○ } (next) <same>

" " " is " " greater than " " ". (next) <same>

"This group is greater than " { ○○ , , , ✓○○ , , , }

"that group". { ✓○○ , , , ○○ , , , }

]

[(←) (↓) (→) (↓)

“There is a number”.

⇔ [(←) (↓) (→) (↓) ”∼∼ number ∼“ { ∼ }

⇔ ["∼∼” { ∼∼} { }]

⇔ [”∼” {∼ }]

(next)

{ ∼ ∼∼ } (next)

{ • ∼ }

]

⇔ ["There is ∼” { ∼ }]

{ • }

]

[(←) (↓) (→) (↓)

“There are “ “ numbers”. { • •• , , , }

⇔ [(←) ( ) (→) ( ) ”∼∼ number ∼“ { ∼ }

⇔ ["∼∼” { ∼∼} { }]

⇔ [”∼” {∼ }]

(next)

{ ∼ ∼∼ } (next)

{ • ∼ }

]

⇔ ["There is ∼” { ∼ }]

“1” { ✔• •• , , , }

, , ,

“ “ { , , , ✔… }

]

When the program retrieves the above and tries to apply them, it finds cases where it only uses { • } or { • •• , , , } and puts away ⇔ with strings that follow.

Addition.

[(←) (↓) (→) (↓)
✔
{○ ○ } ”here is 1 ○” (next)
✔
{○ ○ } ”here is 1 ○” (next)
{○ ○ } ”put them together” (next)
{○○ } (next)
✔
{○○ } “here are 2 ○” (next)
{○○ } “1 and 1 is 2”]

[(←) (↓) (→) (↓)
✔
{○ 1 ○ 1 } ”here is 1 ○” (next)
✔
{○ 1 ○ 1 } ”here is 1 ○” (next)
{○ 1 ○ 1} ”put them together” (next)
{○○ 1 1 } (next)
✔
{○○ 1 1 } “here are 2 ○” (next)
{○○ 1 1 1 and 1 is 2 } “1 and 1 is 2”]

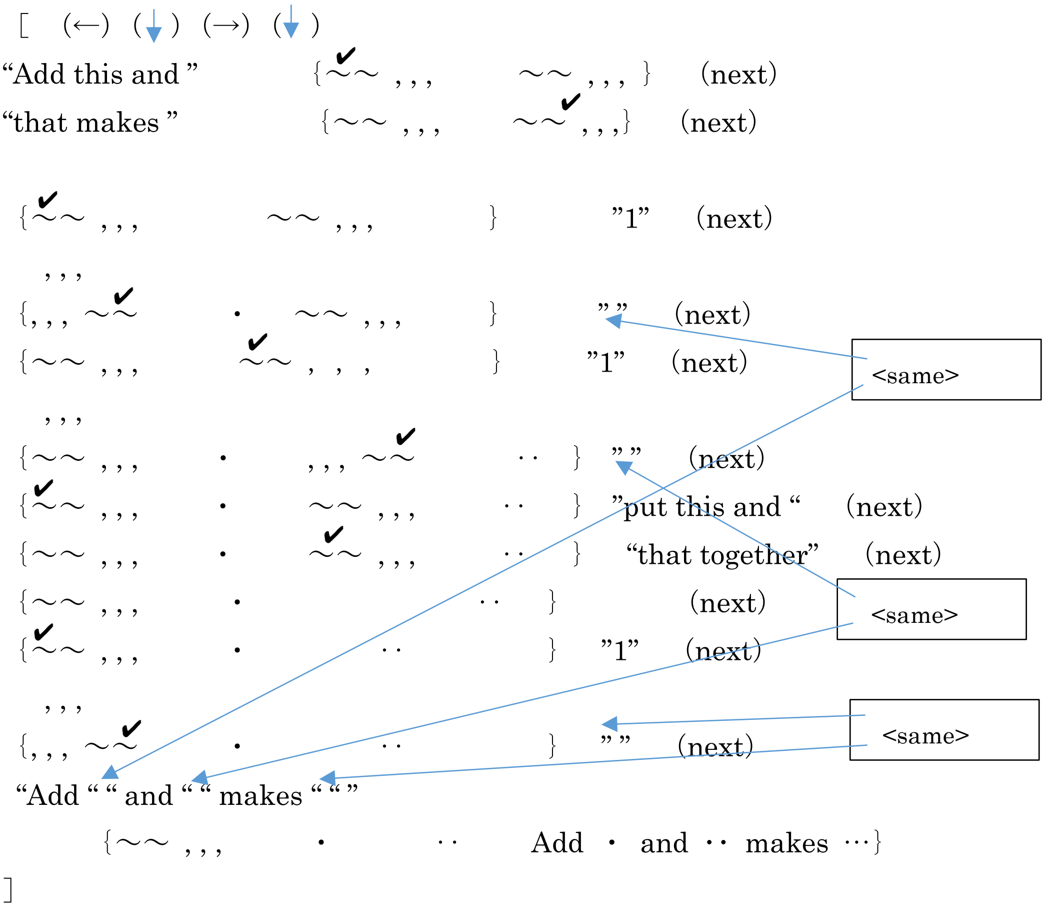
[(←) (↓) (→) (↓)
✔
{○○ 2 ○○ 2 } ”here are 2 ○” (next)
✔
{○○ 2 ○○ 2 } ”here are 2 ○” (next)
{○○ 2 ○○ 2} ”put them together” (next)
{○○○○ 2 2 } (next)
✔
{○○○○ 2 2 } “here are 4 ○” (next)
{○○○○ 2 2 2 and 2 is 4 } “2 and 2 is 4”]

[(←) (↓) (→) (↓)
✔
{○○ 2 ○○○ 3 } ”here are 2 ○” (next)
✔
{○○ 2 ○○○ 3 } ”here are 3 ○” (next)
{○○ 2 ○○○ 3 } ”put them together” (next)
{○○○○○ 2 3 } (next)
✔
{○○○○○ 2 3 } “1” (next)

// omit the intermediate steps (or strings)

{○○○○✔○   2   3   }   “5”   (next)

{○○○○○   2   3   2 and 3 is 5 }   “2 and 3 is 5”   ]



[(←)   (↓)   (→)   (↓)

“Add this and ”   {✔∼∼ , , ,   ∼∼ , , , }   (next)

“that makes ”   {∼∼ , , ,   ∼∼✔ , , ,}   (next)

{✔∼∼ , , ,   ∼∼ , , ,   }   ”1”   (next)

, , ,

{, , , ∼✔∼   ・   ∼∼ , , ,   }   ” ”   (next)

{∼∼ , , ,   ✔∼∼ , , ,   }   ”1”   (next)

<same>

, , ,

{∼∼ , , ,   ・   , , , ∼✔∼   ・・ }   ” ”   (next)

{✔∼∼ , , ,   ・   ∼∼ , , ,   ・・ }   ”put this and “   (next)

{∼∼ , , ,   ・   ✔∼∼ , , ,   ・・ }   “that together”   (next)

{∼∼ , , ,   ・   ・・ }   (next)

<same>

{✔∼∼ , , ,   ・   ・・   }   ”1”   (next)

, , ,

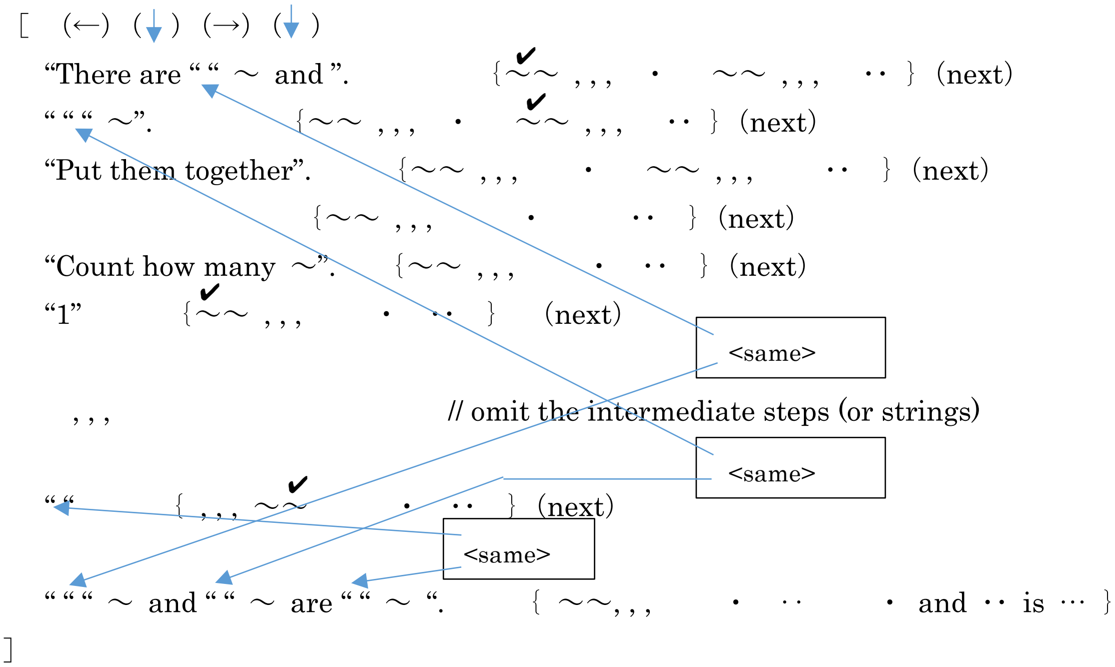
{, , , ∼✔∼   ・   ・・   }   ” ”   (next)

<same>

“Add “ “ and “ “ makes “ “ ”

{∼∼ , , ,   ・   ・・   Add ・ and ・・ makes ・・・}

]



[(←) (↓) (→) (↓))

“There are “ “ ～ and ”. {✔～～ , , , ・ ～～ , , , ・・ } (next)

“ “ “ ～”. {～～ , , , ・ ✔～～ , , , ・・ } (next)

“Put them together”. {～～ , , , ・ ～～ , , , ・・ } (next)

{～～ , , , ・ ・・ } (next)

“Count how many ～”. {～～ , , , ・ ・・ } (next)

“1” {✔～～ , , , ・ ・・ } (next)

<same>

, , , // omit the intermediate steps (or strings)

<same>

“ “ { , , , ✔～～ ・ ・・ } (next)

<same>

“ “ “ ～ and “ “ ～ are “ “ ～ “. { ～～, , , ・ ・・ ・ and ・・ is ⋯ }

]

[(←) (↓) (→) (↓))

”There are “ “ and “ “ ”. { ・ ・・ } (next)

”Add this and “ { ✔・ ・・ } (next)

“that is ” { ・ ✔・・ } (next)

" ”. { ・ ・・ Add ・ and ・・ is ⋯. }

]

[(←) (↓) (→) (↓)

"Add " " and "   { ✔• •• }   (next)

" " " makes "   { ✔• •• }   (next)

" ".   { • •• Add • and •• makes ··· }   ]

[(←) (↓) (→) (↓)

"Add " " to "   { ✔• •• }   (next)

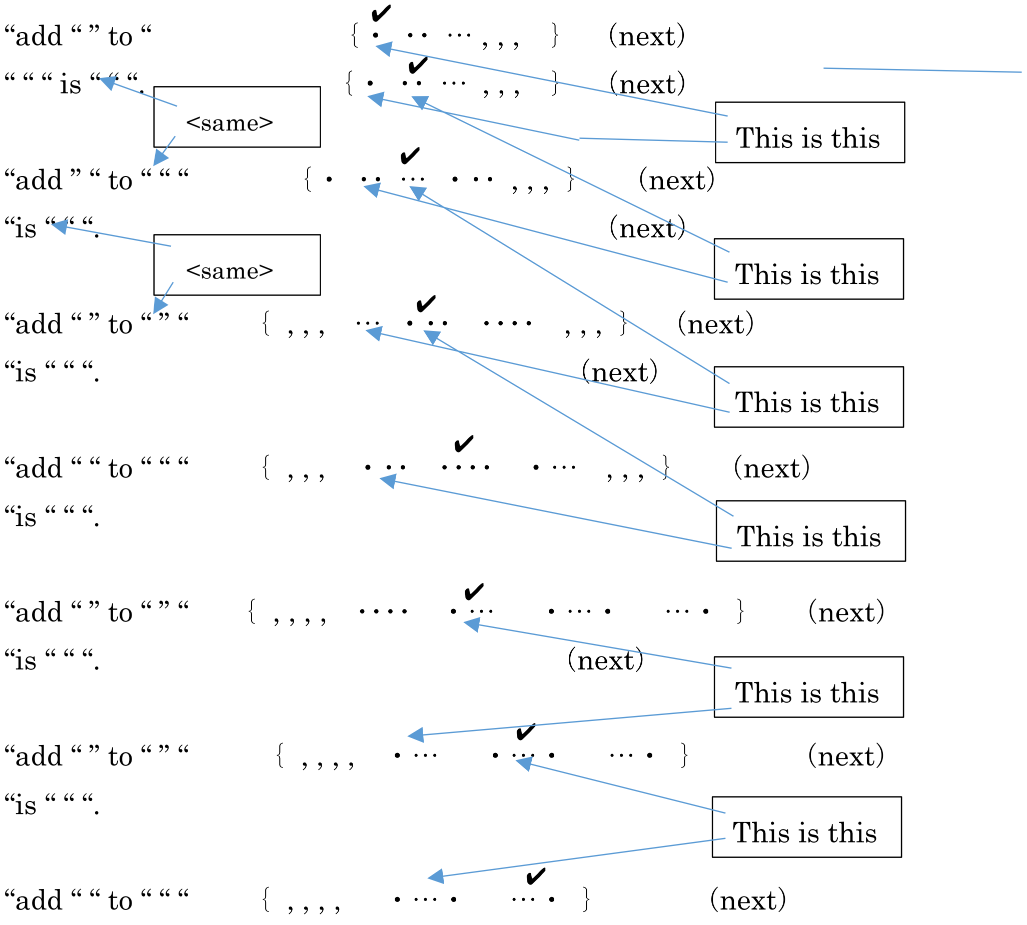
" " " is "   { • ✔•• }   (next)

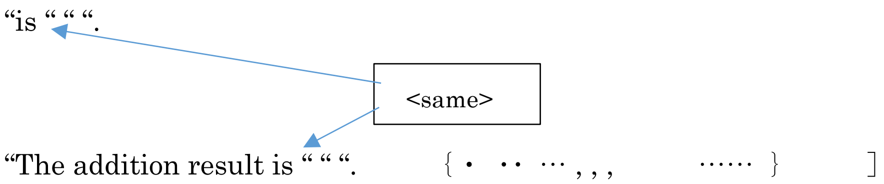
" ".   { • •• Add • to •• is ··· }   ]

[(←) (↓) (→) (↓)

"Add all the numbers in a list".   { • •• , , , }

"What is the addition result".   { • •• , , , }


"add " " to "   { ✔• •• ··· , , , }   (next)
" " " is " " ".   { • ✔•• ··· , , , }   (next)
<same>
This is this
"add " " to " " "   { • •• ✔··· • •• , , , }   (next)
"is " " ".   (next)
<same>
This is this
"add " " to " " "   { , , , ··· ✔• •• ···· , , , }   (next)
"is " " ".   (next)
This is this
"add " " to " " "   { , , , • •• ✔···· • ··· , , , }   (next)
"is " " ".
This is this
"add " " to " " "   { , , , , ···· ✔• ··· • ··· • ··· • }   (next)
"is " " ".   (next)
This is this
"add " " to " " "   { , , , , • ··· ✔• ··· • ··· • }   (next)
"is " " ".
This is this
"add " " to " " "   { , , , , • ··· • ✔··· • }   (next)

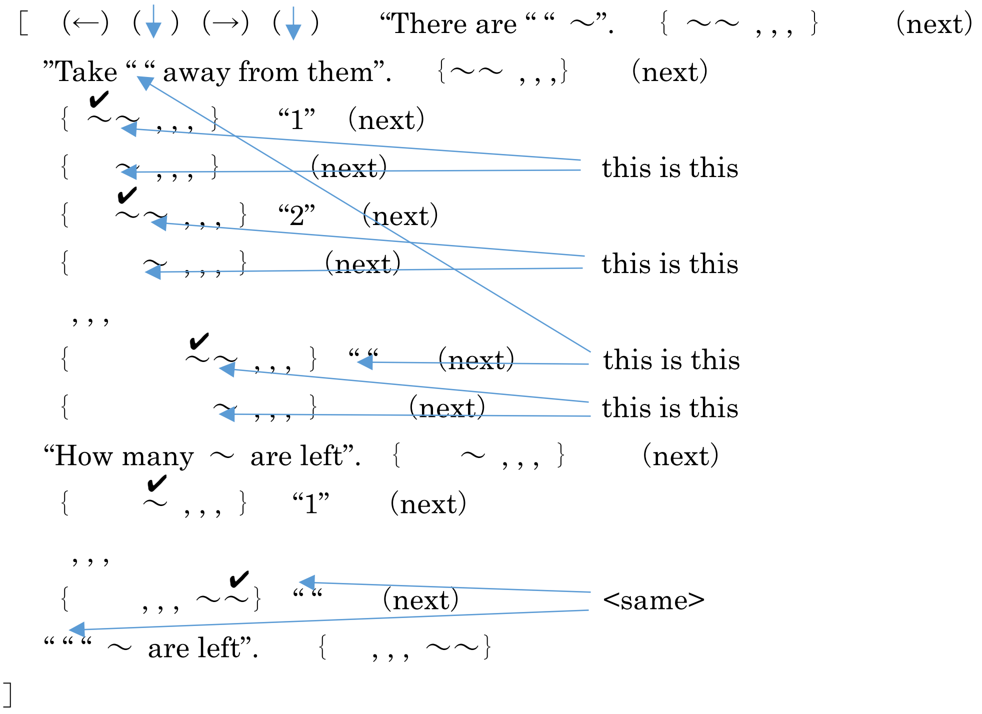

“is “ “ “.

<same>

“The addition result is “ “ “. { • • • ⋯ , , , ⋯⋯ }]

The program retrieves strings that match input strings with replacing sub strings by their representatives. But not further retrieve strings that match retrieved strings. For example, it gets “Add 5 and 4 is”, and retrieves “Add 5 and 4 is”, followed by “9”. But it does not retrieve strings of counting 5 objects, counting 4 objects, putting them together and counting 9 objects.

Subtraction.



[(←) (↓) (→) (↓) “There are “ “ ∼”. { ∼∼ , , , } (next)

”Take “ “ away from them”. {∼∼ , , ,} (next)

{ ✔∼∼ , , , } “1” (next)

{ ∼ , , , } (next) this is this

{ ✔∼∼ , , , } “2” (next)

{ ∼ , , , } (next) this is this

, , ,

{ ✔∼∼ , , , } “ “ (next) this is this

{ ∼ , , , } (next) this is this

“How many ∼ are left”. { ∼ , , , } (next)

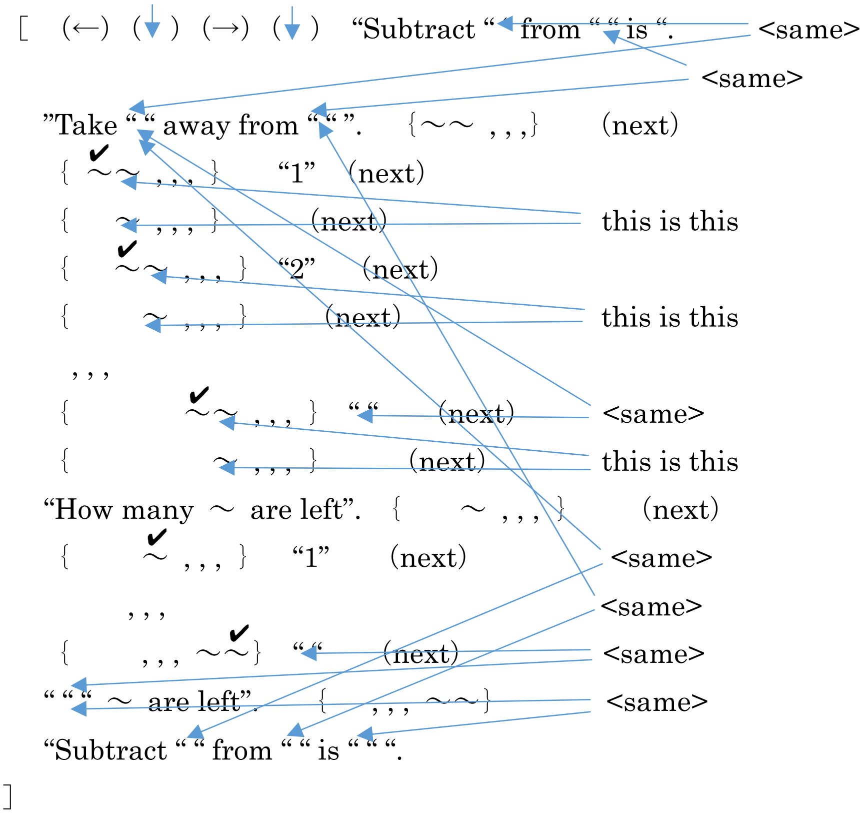
{ ✔∼ , , , } “1” (next)

, , ,

{ , , , ∼✔∼} “ “ (next) <same>

“ “ “ ∼ are left”. { , , , ∼∼}

]



[(←) (↓) (→) (↓) “Subtract “ “ from “ “ is “. <same>

<same>

”Take “ “ away from “ “ ”. {∼∼ , , ,} (next)

{ ✔∼∼ , , , } “1” (next)

{ ∼ , , , } (next) this is this

{ ✔∼∼ , , , } “2” (next)

{ ∼ , , , } (next) this is this

, , ,

{ ✔∼∼ , , , } “ “ (next) <same>

{ ∼ , , , } (next) this is this

“How many ∼ are left”. { ∼ , , , } (next)

{ ✔∼ , , , } “1” (next) <same>

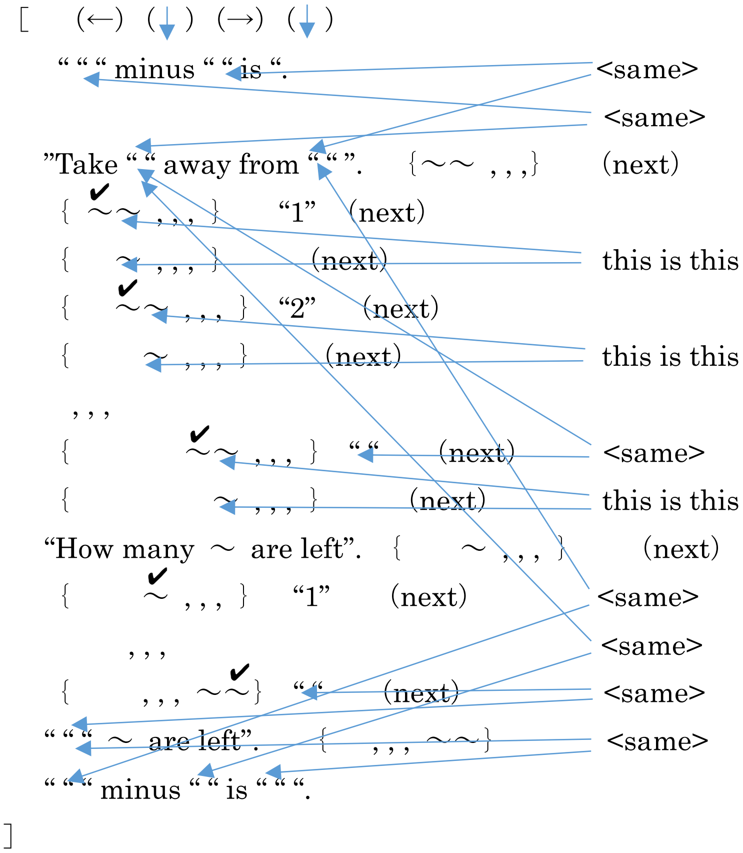
, , , <same>

{ , , , ∼✔∼} “ “ (next) <same>

“ “ “ ∼ are left”. { , , , ∼∼} <same>

“Subtract “ “ from “ “ is “ “ “.

]



[(←) (↓) (→) (↓)

“ “ “ minus “ “ is “. <same>

<same>

”Take “ “ away from “ “ ”. {～～ , , ,} (next)

{ ✔～～ , , , } “1” (next)

{ ～ , , , } (next) this is this

{ ✔～～ , , , } “2” (next)

{ ～ , , , } (next) this is this

, , ,

{ ✔～～ , , , } “ “ (next) <same>

{ ～ , , , } (next) this is this

“How many ～ are left”. { ～ , , , } (next)

{ ✔～ , , , } “1” (next) <same>

, , ,

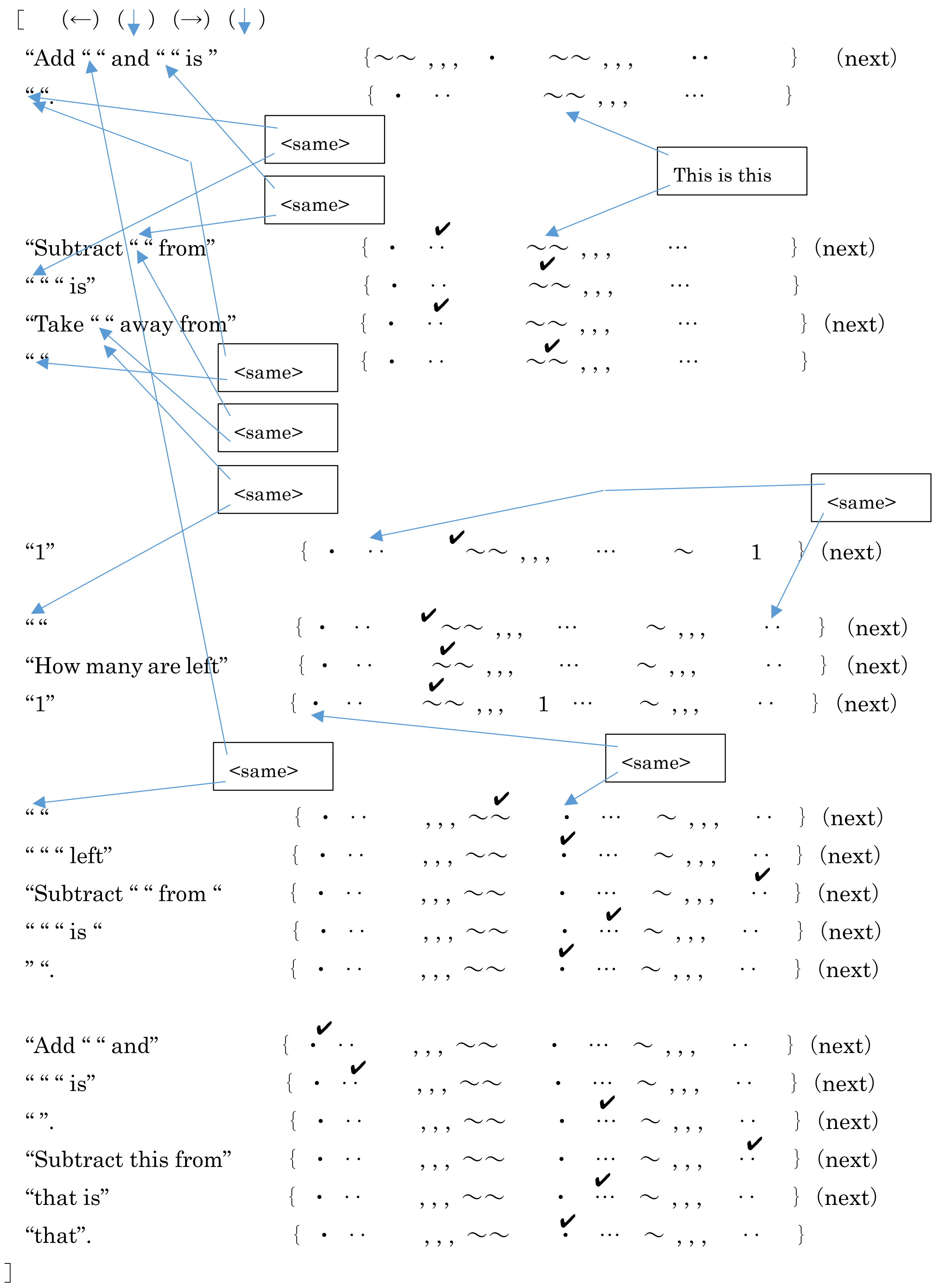
<same>

{ , , , ～✔～} “ “ (next) <same>

“ “ “ ～ are left”. { , , , ～～} <same>

“ “ “ minus “ “ is “ “ “.

]



[(←) (↓) (→) (↓)

"Add " " and " " is " {∼∼ , , , • ∼∼ , , , • • } (next)

" ". { • • • ∼∼ , , , ⋯ }

<same>

<same>

This is this

"Subtract " " from" { • ✔ • • ∼∼ ✔ , , , ⋯ } (next)

" " " is" { • • • ∼∼ , , , ⋯ }

"Take " " away from" { • ✔ • • ∼∼ , , , ⋯ } (next)

" " { • • • ✔ ∼∼ , , , ⋯ }

<same>

<same>

<same>

<same>

"1" { • • • ✔ ∼∼ , , , ⋯ ∼ 1 } (next)

" " { • • • ✔ ∼∼ , , , ⋯ ∼ , , , • • } (next)

"How many are left" { • • • ✔ ∼∼ , , , ⋯ ∼ , , , • • } (next)

"1" { • • • ✔ ∼∼ , , , 1 ⋯ ∼ , , , • • } (next)

<same>

<same>

" " { • • • , , , ∼∼ ✔ • ⋯ ∼ , , , • • } (next)

" " " left" { • • • , , , ∼∼ ✔ • ⋯ ∼ , , , • • } (next)

"Subtract " " from " { • • • , , , ∼∼ • ⋯ ∼ , , , ✔ • • } (next)

" " " is " { • • • , , , ∼∼ • ✔ ⋯ ∼ , , , • • } (next)

" ". { • • • , , , ∼∼ ✔ • ⋯ ∼ , , , • • } (next)

"Add " " and" { ✔ • • • , , , ∼∼ • ⋯ ∼ , , , • • } (next)

" " " is" { • ✔ • • , , , ∼∼ • ⋯ ∼ , , , • • } (next)

" ". { • • • , , , ∼∼ • ✔ ⋯ ∼ , , , • • } (next)

"Subtract this from" { • • • , , , ∼∼ • ⋯ ∼ , , , ✔ • • } (next)

"that is" { • • • , , , ∼∼ • ✔ ⋯ ∼ , , , • • } (next)

"that". { • • • , , , ∼∼ ✔ • ⋯ ∼ , , , • • }

]

[(←) (↓) (→) (↓)

“Add “ “ and” { ✓• ·· } (next)

“ “ “ is” { • ✓·· } (next)

“ ”. { • ·· ✓··· } (next)

“Subtract this from” { • ✓·· ··· } (next)

“that is” { • ·· ✓··· } (next)

“that”. { ✓• ·· ··· }

]

[(←) (↓) (→) (↓)

“Add “ “ and” { ✓• ·· } (next)

“ “ “ is” { • ✓·· } (next)

“ ”. { • ·· ✓··· } (next)

“Subtract this from” { ✓• ·· ··· } (next)

“that is” { • ·· ✓··· } (next)

“that”. { • ✓·· ··· }

]

[(←) (↓) (→) (↓)

“Subtract “ “ from” { ✓• ·· } (next)

“ “ “ is” { • ✓·· } (next)

“ ”. { • ·· ✓··· } (next)

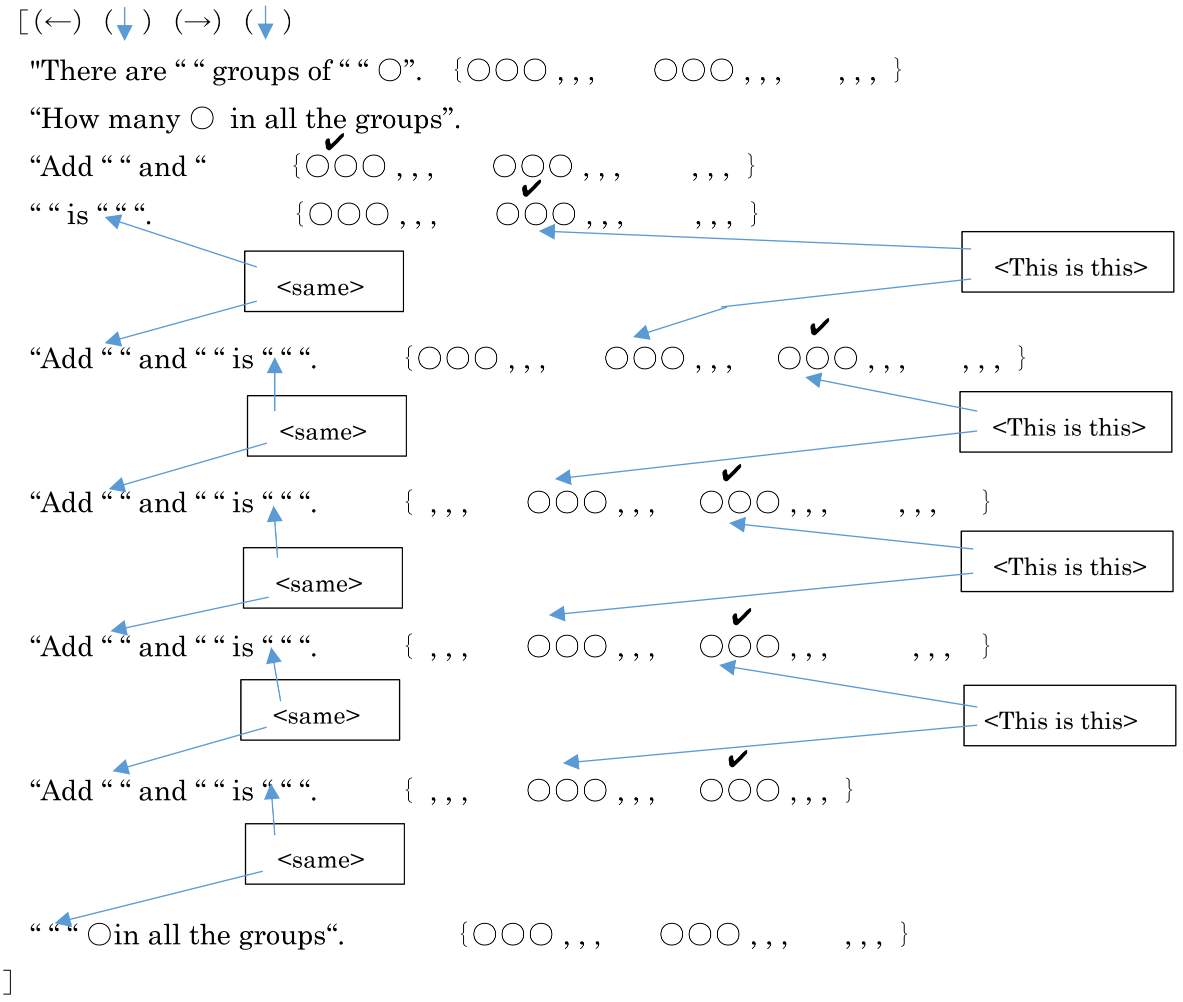
“Add this and” { ✓• ·· ··· } (next)

“that is” { • ·· ✓··· } (next)

“that”. { • ✓·· ··· }

]

Multiplication



[(←) (↓) (→) (↓)

"There are “ “ groups of “ “ ○”. {○○○ , , , ○○○ , , , , , , }

“How many ○ in all the groups”.

“Add “ “ and “ {✔○○○ , , , ○○○ , , , , , , }

“ “ is “ “ “. {○○○ , , , ✔○○○ , , , , , , }

<same>

<This is this>

“Add “ “ and “ “ is “ “ “. {○○○ , , , ○○○ , , , ✔○○○ , , , , , , }

<same>

<This is this>

“Add “ “ and “ “ is “ “ “. { , , , ○○○ , , , ✔○○○ , , , , , , }

<same>

<This is this>

“Add “ “ and “ “ is “ “ “. { , , , ○○○ , , , ✔○○○ , , , , , , }

<same>

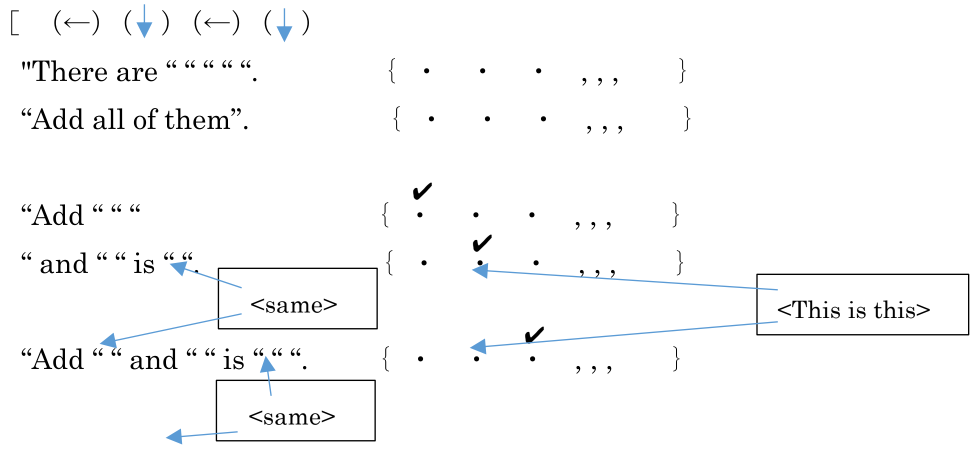
<This is this>

“Add “ “ and “ “ is “ “ “. { , , , ○○○ , , , ✔○○○ , , , }

<same>

“ “ “ ○in all the groups“. {○○○ , , , ○○○ , , , , , , }

]



[(←) (↓) (←) (↓)

"There are “ “ “ “ “ “. { · · · , , , }

“Add all of them”. { · · · , , , }

“Add “ “ “ { ✔· · · , , , }

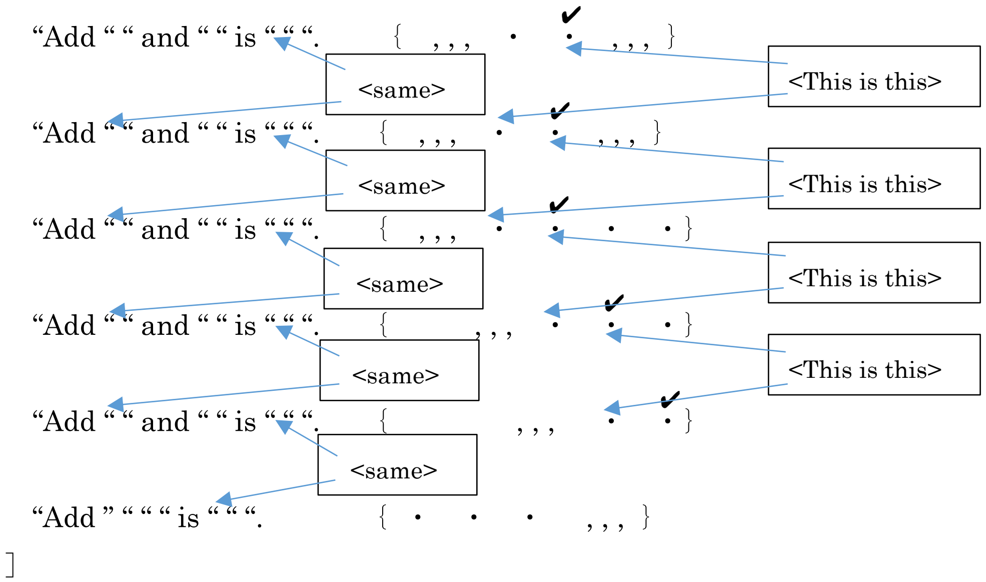
“ and “ “ is “ “. { · ✔· · , , , }

<same>

<This is this>

“Add “ “ and “ “ is “ “ “. { · · ✔· , , , }

<same>



“Add “ “ and “ “ is “ “ “. { , , , • ✔ • , , , }

<same> <This is this>

“Add “ “ and “ “ is “ “ “. { , , , • ✔ • , , , }

<same> <This is this>

“Add “ “ and “ “ is “ “ “. { , , , • ✔ • • • }

<same> <This is this>

“Add “ “ and “ “ is “ “ “. { , , , • ✔ • • }

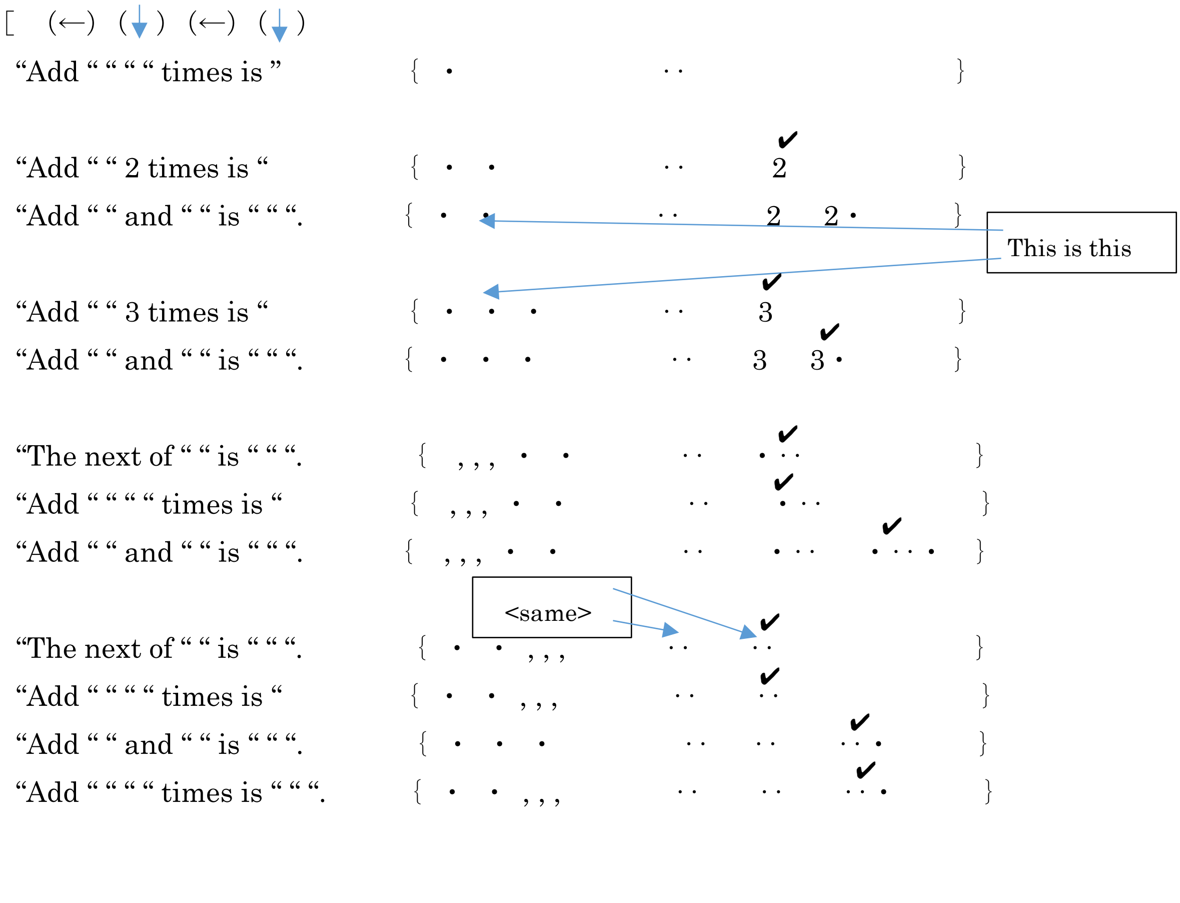
<same> <This is this>

“Add “ “ and “ “ is “ “ “. { , , , ✔ • • }

<same>

“Add ” “ “ “ is “ “ “. { • • • , , , }

]



[(←) (↓) (←) (↓)

“Add “ “ “ “ times is ” { • · · }

“Add “ “ 2 times is “ { • • · · ✔ 2 }

“Add “ “ and “ “ is “ “ “. { • • · · 2 2 • }

This is this

“Add “ “ 3 times is “ { • • • · · ✔ 3 }

“Add “ “ and “ “ is “ “ “. { • • • · · 3 ✔ 3 • }

“The next of “ “ is “ “ “. { , , , • • · · ✔ • · · }

“Add “ “ “ “ times is “ { , , , • • · · ✔ • · · }

“Add “ “ and “ “ is “ “ “. { , , , • • · · • · · ✔ • · · • }

<same>

“The next of “ “ is “ “ “. { • • , , , · · ✔ · · }

“Add “ “ “ “ times is “ { • • , , , · · ✔ · · }

“Add “ “ and “ “ is “ “ “. { • • • · · · · ✔ · · • }

“Add “ “ “ “ times is “ “ “. { • • , , , · · · · ✔ · · • }

]

［　(←)　(↓)　(←)　(↓)

“ “ “ times “ “ is”　　　{　•　　　　　‥　　　　　　}

“Add “ “ “ “ times is “　　　{　•　　　　　‥　　　　　　}

⇔　［ “Add “ “ “ “ times is “

// 　omit steps

］

“ “ “ times “ “ is “ “ ”.　　　{　•　　　　　‥　　　　…•　　}

］

［　(←)　(↓)　(←)　(↓)

“Multiply “ “ by “ “ is “　　　{　•　　　　　‥　　　　　　}

“Add “ “ “ “ times is “　　　{　•　　　　　‥　　　　　　}

⇔　［ “Add “ “ “ “ times is “

// 　omit steps

］

“Multiply “ “ by “ “ is “ “ “.　{　•　　　　　‥　　　　…•　　}

］

Division

[(←) (↓) (←) (↓)
“Divide 4 ○ into 2”. { ○○ , , , } (next)

{ ○○○○ } (next)

{ ✔○○○○ } (next)

{ ○○○ ✔○ } (next)

{ ✔○○○ ○ } (next)

{ ○○ ○ ✔○ } (next)

{ ✔○○ ○ } (next)

{ ○ ○✔○ ○ } (next)

{ ✔○ ○○ ○ } (next)

{ ○○ ○✔○ } (next)

“2 ○ each” { ○○ ○○ }
]

[(←) (↓) (←) (↓)
“Divide 6 ○ into 2”. {○○ , , , }

{○○ , , , }

{ ○○ , , , ○ }

{ ○○ , , , ○ ○ }

{ 〇〇〇 〇〇 〇 }

{ 〇〇 〇〇 〇〇 }

{ 〇 〇〇〇 〇〇 }

{ 〇〇〇 〇〇〇 }

“2 3 〇” { 〇〇〇 〇〇〇 }

“Divide 6 〇 into 2 is 3 〇”
]

[(←) (↓) (←) (↓)
“Divide 6 〇 into 3”. {〇〇 , , , }

{〇〇 , , , }

{ 〇〇 , , , 〇 }

{ 〇〇 , , , 〇 〇 }

{ 〇〇〇 〇 〇 〇 }

{ 〇〇 〇〇 〇 〇 }

{ 〇 〇〇 〇〇 〇 }

{ 〇〇 〇〇 〇〇 }

“3 2 〇” { 〇〇 〇〇 〇〇 }

“Divide 6 〇 into 3 is 2 〇”.

]

[(←) (↓) (←) (↓)

"Divide 7 ○ into 3". {○○ , , , }

{○○ , , , }

{ ○○ , , , ○ }

{ ○○ , , , ○ ○ }

{ ○○ , , , ○ ○ ○ }

{ ○○○ ○○ ○ ○ }

{ ○○ ○○ ○○ ○ }

{ ○ ○○ ○○ ○○ }

"3 2 ○" { ○ ○○ ○○ ○○ }

"1 ○ remainder" { ○ ○○ ○○ ○○ }

"Divide 6 ○ into 3 is 2 ○ and 1 ○ remainder".

]

[(←) (↓) (←) (↓)

Divide " " ○ into " "

" " {○○ , , , }

"1" {○○ , , , }

"1" { ○ , , , ○ }

"2" { ○ , , , ○ ○ }

, , ,
“ “ { ○○ , , , ○ ○ , , , ○ }

“2”

“1” { ○ , , , ○○ ○ , , , ○ }

“2” { ○ , , , ○○ ○ , , , ○ }

, , ,
“ “ { ○○ , , , ○○ ○○ , , , ○○ }

, , ,
“ “

“1” { ○ , , , , , ,○ , , , , , , , , , }

“2” { ○ , , , , , ,○ , , ,○ , , , , , , }

, , ,
“ “ { , , ,○ , , , ○ , , , , , ,○ }

✔
“Count how many in this” ○○ , , , ○○ , , , , , , ○○ , , ,
✔
“1” ○○ , , , ○○ , , , , , , ○○ , , ,
✔
“2” ○○ , , , ○○ , , , , , , ○○ , , ,

✔
“ “ , , , ○ ○○ , , , , , , ○○ , , ,

Divide “ “ ○ into “ “ is “ “ ○. { • •• ⋯ }

]

[(←) (↓) (←) (↓)

Divide “ “ ○ into “ “

“ “ {○○ , , , }

“1” {○○ , , , }

“1” { ○ , , , ○ }

“2” { ○ , , , ○ ○ }

, , ,
“ “ { ○○ , , , ○ ○ , , , ○ }

“2”

“1” { ○ , , , ○○ ○ , , , ○ }

“2” { ○ , , , ○○ ○ , , , ○ }

, , ,

“ “ { ○○ , , , ○○ ○○ , , , ○○ }

, , ,

“ “

“1” { ○ , , , , , ,○ , , , , , , , , , }

“2” { ○ , , , , , ,○ , , ,○ , , , , , , }

, , ,

“ “ { ○ , , , , , ,○ , , , ○ , , , , , ,○ }

“Count how many in this”

{ ○ , , , ✔○○ , , , ○○ , , , , , , ○○ , , , }

“1” ○ , , , ✔○○ , , , ○○ , , , , , , ○○ , , , }

“2” ○ , , , ○✔○ , , , ○○ , , , , , , ○○ , , , }

“ “ ○ , , , , , , ✔○ ○○ , , , , , , ○○ , , , }

“Count how many in this”

{ ✔○ , , , ○○ , , , ○○ , , , , , , ○○ , , , }

“1” { ✔○○ , , , ○○ , , , ○○ , , , , , , ○○ , , , }

“2” { ○✔○ , , , ○○ , , , ○○ , , , , , , ○○ , , , }

“ “ { , , , ✔○ , , , ○ ○○ , , , , , , ○○ , , , }

“Divide “ “ ○ into “ “ is “ “ ○ and” . { • •• ⋯ }

“ “ “ ○ remainder”. { • •• ⋯ ⋯ • }

]

[(←) (↓) (←) (↓)

" " times " " is ". { • ✔ • • }

{ ✔ • • • }

"1" { • • 1 • • }

"2" { • • 2 • 2 • • }

, , ,

" " { • ✔ • • , , , • ✔ … • • • • }

" " " • here" { • • , , , • … • • • • }

"Divide " " by " " is ". { • • , , , • • • • ✔ … • • • • }

{ • • , , , • • • • … ✔ • • • • }

" " " • here" { ✔ • • , , , • • • • … • • • • }

" ". { ✔ • • , , , • • • • … • • • • }

]

[(←) (↓) (←) (↓)

" " times " " is " ". { • • • … }

"Divide " " by " " is " " ". { • • • … … • • • }

"This and this are " { • • • … ✔ … • • • }

{ • • • … … ✔ • • • }

"the same as this and this". { • • • ✔ … … • • • }

{ • ✔ • • … … • • • }

"This is the same as this". { • • • … … • • ✔ • }

{ ✔ • • • … … • • • }

]

[(←) (↓) (←) (↓)

“ “ times “ “ is “ “. { • •• ··· }

“ “ “ plus “ “ is “ “. { • •• ··· •••• ······• }

“Divide “ “ by “ “ is “ “ “. { ··· • ····• ······ }

“and remainder is “ “ “. { ··· • ····• ······ • •• }

“This and this are “ { • •• ··· •••• ✔······• }

{ • ✔•• ··· •••• ······• }

“the same as this and this”. { ✔··· • ····• ······ • •• }

{ ··· • ✔····• ······ • •• }

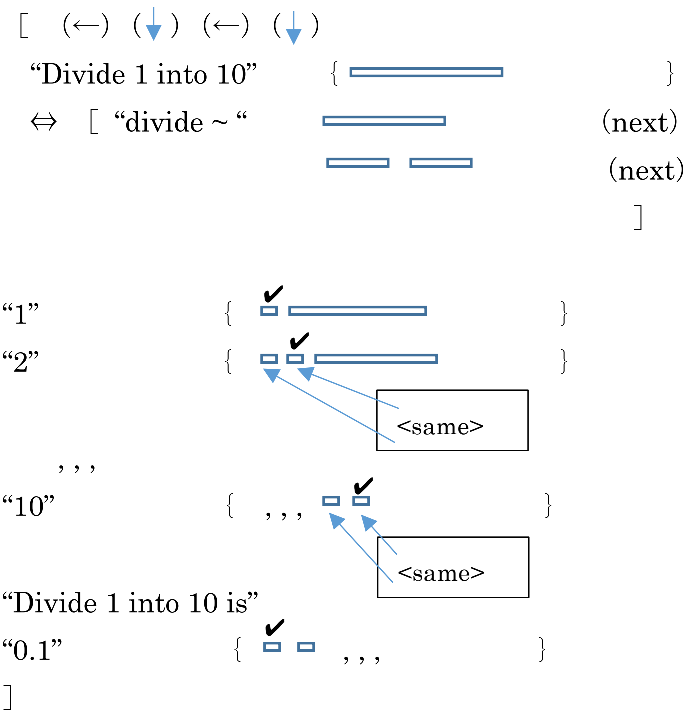
“This is the same as this”. { • •• ··· •••• ✔······• }

{ ··· • ····• ······ ✔•••• }

“This is the same as this”. { ✔• •• ··· •••• ······• }

{ ··· • ····• ✔······ •••• }

]



[(←) (↓) (←) (↓)

“Divide 1 into 10” { }

⇔ [“divide ~ “ (next)

(next)

]

“1” { ✔ }

“2” { ✔ }

<same>

, , ,

“10” { , , , ✔ }

<same>

“Divide 1 into 10 is”

“0.1” { ✔ , , , }

]

[(←) (↓) (←) (↓)

“This is 1”. { ▭ □ □ , , , □ }

“This is 0.1”. { ✓▭ ✓□ □ , , , □ }

“count how many 0.1 here“

{ ▭ ✓□ □ , , , □ }

“1” { ▭ ✓□ □ , , , □ }

“2” { ▭ □ ✓□ , , , □ }

, , ,

“ “ { ▭ □ □ , , , ✓□ }

“ “ “ 0.1 “ { ▭ ✓□ □ , , , □ }

]

[(←) (↓) (←) (↓)

“2 of 1 divided by 10 is” { □ □ , , , }

⇔ [“Divide 1 by 10 is” { ▭ }

“0.1” { □ □ , , , }]

“1” { ✓□ □ }

“2” { □ ✓□ }

“put them together” { ✓□□ }

“2 of 1 divided by 10 is”

“0.2” { ✓□□ }

]

[(←) (↓) (←) (↓)

“ “ “ of 1 divided into 10 is ” { □ □ , , , }

⇔ [“Divide 1 by 10 is” { ▭ }

“0.1” { □ □ , , , }]

“1” { ✓□ □ , , , }

“2” { □ ✓□ , , , }

, , ,

” “ { □ □ , , , ✓□ , , , }

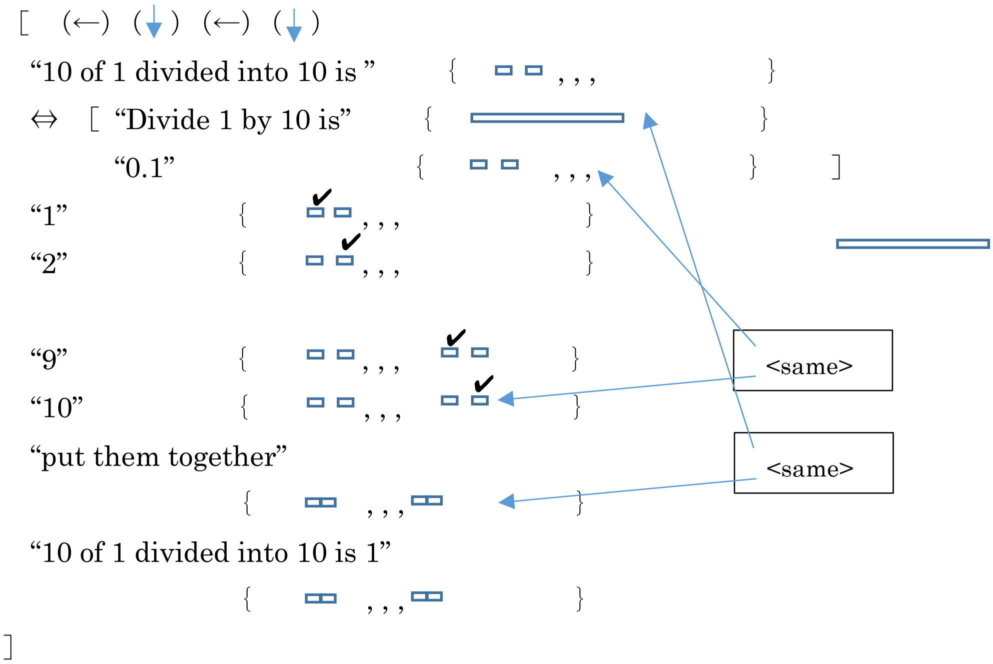
“put them together”

{ ✓□□ , , , □ }

“ “ “ of 1 divided into 10 is ”

“0.” “ ” { ✓□□□ , , , □ }

]



[(←) (↓) (←) (↓))

“10 of 1 divided into 10 is ” { □ □ , , , }

⇔ [“Divide 1 by 10 is” { ▭ }

“0.1” { □ □ , , , }]

“1” { ✓□ □ , , , }

“2” { □ ✓□ , , , }

“9” { □ □ , , , ✓□ □ }

“10” { □ □ , , , □ ✓□ }

“put them together”

{ □□ , , , □□ }

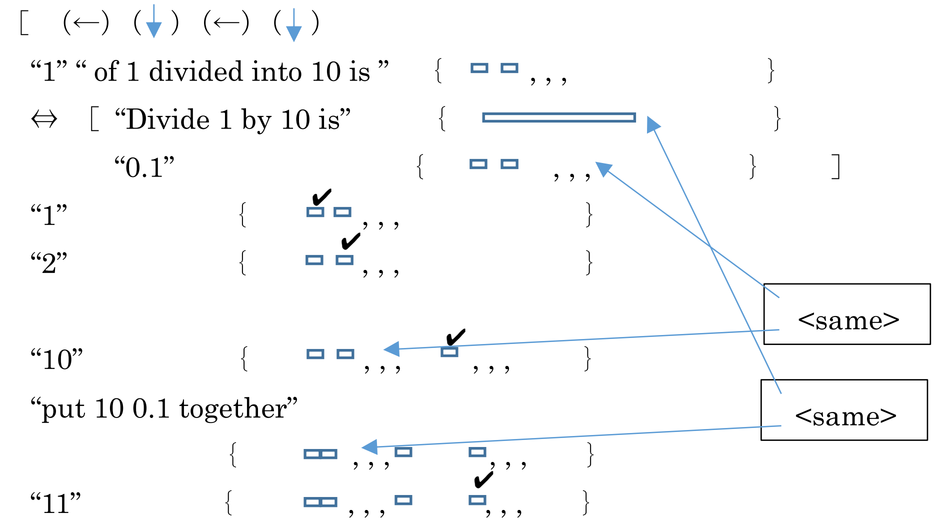
“10 of 1 divided into 10 is 1”

{ □□ , , , □□ }

]

<same>

<same>



[(←) (↓) (←) (↓))

“1” “ of 1 divided into 10 is ” { □ □ , , , }

⇔ [“Divide 1 by 10 is” { ▭ }

“0.1” { □ □ , , , }]

“1” { ✓□ □ , , , }

“2” { □ ✓□ , , , }

“10” { □ □ , , , ✓□ , , , }

“put 10 0.1 together”

{ □□ , , , □ □ , , , }

“11” { □□ , , , □ ✓□ , , , }

<same>

<same>

“1” ” “ { □□ , , , □ , , , ✓□ }

“put “ “ 0.1 together”

{ □□, , , □ □□, , , □ }

“1” “ of 1 divided into 10 is”

“10 of 1 divided into 10 and”

{ □□ ✓, , , □ □□, , , □ }

“ “ “ of 1 divided into 10 is”

{ □□, , , □ ✓□□, , , □ }

“1.” ” “

{ □□, , , □ □□, , , □ }

]

[(←) (↓) (←) (↓))

“This is 1”. { ✓▭ □ □ , , , □ □ □ , , , □ }

“This is 0.1”. { ▭ ✓□ □ , , , □ □ □ , , , □ }

“This is 0.1”. { ▭ □ □ , , , □ ✓□ □ , , , □ }

“count how many 0.1 here“

{ ▭ ✓□ □ , , , □ □ □ , , , □ }

“1” { ▭ ✓□ □ , , , □ □ □ , , , □ }

“2” { ▭ □ ✓□ , , , □ □ □ , , , □ }

, , ,

“ “ { ▭ □ □ , , , ✓□ □ □ , , , □ }

“count how many 0.1 here”

{ ▭ □ □ , , , □ ✓□ □ , , , □ }

“1” { ▭ □ □ , , , □ ✓□ □ , , , □ }

“2” { ▭ □ □ , , , □ □ ✓□ , , , □ }

, , ,

“ “ { ▭ □ □ , , , □ □ □ , , , ✓□ }

“put them together”

{ ▭ ✓□□ , , , □ }

“count how many 0.1”

“1” { ▭ ✓□□ , , , □ }

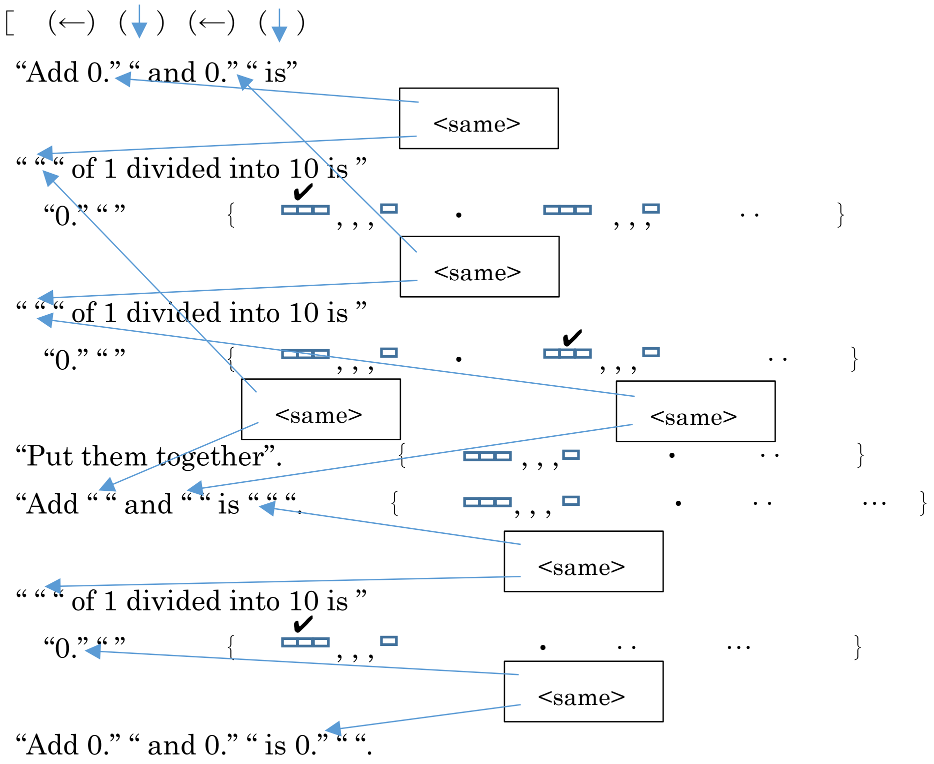
“2” { ▭ □✓□ , , , □ }

, , ,

“ “ { ▭ □□ , , , ✓□ . }

“ “ “ 0.1 “ { ▭ □□ , , , □ . }

]



[(←) (↓) (←) (↓)

“Add 0.” “ and 0.” “ is”

<same>

“ “ “ of 1 divided into 10 is ”

“0.” “ ” { ✔□□□ , , , □ • □□□ , , , □ . . }

<same>

“ “ “ of 1 divided into 10 is ”

“0.” “ ” { □□□ , , , □ • ✔□□□ , , , □ . . }

<same>

<same>

“Put them together”. { □□□ , , , □ • . . }

“Add “ “ and “ “ is “ “ “. { □□□, , , □ • . . … }

<same>

“ “ “ of 1 divided into 10 is ”

“0.” “ ” { ✔□□□ , , , □ • . . … }

<same>

“Add 0.” “ and 0.” “ is 0.” “ “.

]

[(←) (↓) (←) (↓)

“Add 0.” “ and 0.” “ is”

“ “ “ of 1 divided into 10 is ”

“0.” “ ” { ✔□□□ , , , □ • □□□ , , , □ . . }

“ “ “ of 1 divided into 10 is ”

“0.” “ ” { □□□ , , , □ • ✔□□□ , , , □ . . }

“Put them together”. { □□□ , , , □ • . . }

“Add “ “ and “ “ is 1“ “ “. { □□□ , , , □ • . . … }

“1” “ of 1 divided into 10 is”

“10 of 1 divided into 10 and”

{ □□, ✔, , □ □□, , , □ • . . … }

“ “ “ of 1 divided into 10 is”

{ □□, , , □ ✔□□, , , □ • . . … }

“1.” ” “

{ □□, , , □ □□, , , □ • · · … }

“Add 0.” “ and 0.” “ is 1.” “ “.

]

[(←) (↓) (←) (↓)

“Divide 0.1 into 10” { ▭ 0.1 }

“1” { ✔□ ▭ 0.1 }

“2” { □ ✔□ ▭ 0.1 }

, , ,

“10” { , , , □ ✔□ 0.1 }

“ Divide 0.1 into 10 is”

“0.01”. { ✔□ □ , , , 0.01 0.1 }

]

[(←) (↓) (←) (↓)

“ “ “ of 0.1 divided by 10 is” { □ □ , , , □ }

⇔ [“Divide 0.1 by 10 is” { ▭ }

“0.01” { □ □ , , , }]

⇔ [“ “ of ∼ { ✔∼∼ , , , • }]

“ 0.0” “ “. { 0.0 • □ □ , , , □ • }

]

[(←) (↓) (←) (↓)

“10 of 0.1 divided into 10 is ” { ▭ }

⇔ [“Divide 0.1 by 10 is” { ▭ }

“0.01” { □ □ , , , }]

“1” { ✔□ □ , , , }

“2” { □ ✔□ , , , }

“9” { □ □ , , , ✔□ □ }

✔

<same>

“10” { □ □ , , , □ □ }

“put them together” <same>

{ ◫ , , , ◫ }

“10 of 0.1 divided into 10 is 0.1”

{ ◫ , , , ◫ }

]

[(←) (↓) (←) (↓)

“100 of 0.1 divided into 10 is ” { □ □ , , , }

⇔ [“Divide 0.1 by 10 is” { ▭ }

“0.01” { □ □ , , , }]

“1” { ✓□ □ , , , }

“2” { □ ✓□ , , , }

, , ,

“10” { □ □ , , , ✓□ □ , , , } <same>

“Put them together”

{ ✓◫ , , , □ □ , , , }

“10 of 0.1 divided into 10 is 0.1”

{ ◫ , , , □ □ , , , }

“11”

{ ◫ , , , □ ✓□ , , , }

“20” { ◫ , , , □ □ , , , ✓□ , , , }

“put them together”

{ ◫ , , , □ ◫ , , , □ , , , }

“99” { , , , , , , ✓□ □ }

“100” { , , , , , , □ ✓□ }

“put them together”

{ , , , ◫ , , , □ }

“10 of 0.1 divided into 10 is 0.1”

{ ◫ , , , □ , , , ◫ , , , □ }

“There are 10 0.1”

{ ◫ , , , □ , , , ◫ , , , □ }

“10 0.1 is 1”

“100 of 0.1 divided into 10 is 1”

{ ▭▭ , , , ▭ , , , ▭▭ , , , ▭ }

]

[(←) (↓) (←) (↓))

“Add 0.0” “ and 0.0” “ is”

“ “ “ of 0.1 divided into 10 is ”

“0.0” “ ” { ✔▭▭▭ , , , ▭ • ▭▭▭ , , , ▭ · · }

“ “ “ of 1 divided into 10 is ”

“0.0” “ ” { ▭▭▭ , , , ▭ • ✔▭▭▭ , , , ▭ · · }

“Put them together”. { ▭▭▭ , , , ▭ • · · }

“Add “ “ and “ “ is “ “ “. { ▭▭▭ , , , ▭ • · · ··· }

“ “ “ of 0.1 divided into 10 is ”

“0.0” “ ” { ✔▭▭▭ , , , ▭ • · · ··· }

“Add 0.0” “ and 0.0” “ is 0.0” “ “.

]

[(←) (↓) (←) (↓))

“Divide 0.0 , , , 1 into 10” { 0.0 , , , 1 ▭ }

“1” { 0.0 , , , 1 ▭ ▭ }

<This is this>

“2” { 0.0 , , , 1 ▭ ▭ ▭ }

, , ,

“10” { 0.0 , , , 1 , , , ▭ ▭ }

“count how many 0 right of .” { 0.0 , , , 1 ▭ , , , }

“1” { 0.✔0 , , , 1 , , , }

<This is this>

“2” { 0.0✔0 , , , 1 , , , }

, , ,

“ “ { 0.0 , , ,✔01 , , , }

“1 larger number of “ “ is “ “ “

<This is this>

“add 0 right of the rightmost 0” { 0.0 , , , ✔001 ▭ , , , }

“ “ “ 0 right of .” { 0.0 , , , 001 , , , }

“Divide 0.0 , , , 1 into 10 is”

“0.0 , , , 01”

]

[(←) (↓) (←) (↓)

“10 of 0.0 , , , 1 divided into 10 is ” { ▭ }

⇔ [“Divide 0.0 , , , 1 by 10 is” { ▭ }

“0.0 , , , 01” { ▭ ▭ , , , }]

“1” { ✔▭ ▭ , , , }

“2” { ▭ ✔▭ , , , }

“9” { ▭ ▭ , , , ✔▭ ▭ }

“10” { ▭ ▭ , , , ▭ ✔▭ }

“put them together”

{ ▭▭ , , , ▭▭ }

“10 of 0.0 , , , 1 divided into 10 is 0.0 , , , 1”

{ ▭▭ , , , ▭▭ }

]

Find the number.

[(←) (↓) (→) (↓)

"There is one number". { • }

This is this

"Add " " to the number is ". { • ·· ··· Add ·· to • is }

" " { • ·· ··· Add ·· to • is ✓ ··· }

This is this

"Find the number". { ✓ • ·· ··· Add ·· to • is ··· }

This is this

" " " minus " " is ". { • ·· ··· Add ·· to • is ··· ··· minus ·· is }

" ". { • ·· ··· Add ·· to • is ··· ··· minus ·· is ✓ • }

<same>

"The number is " " ".]

[(←) (↓) (→) (↓)

"There is one number". { • }

"Multiply " " and the number is ". { ·· 1 ·· }

This is this

{ ·· ·· 2 • ·· }

, , ,

" ". { ·· ·· , , , • ··· Multiply ·· and • is ✓ ··· }

This is this

"Find the number". { ·· ·· , , , ✓ • ··· Multiply ·· and • is ··· }

<same>

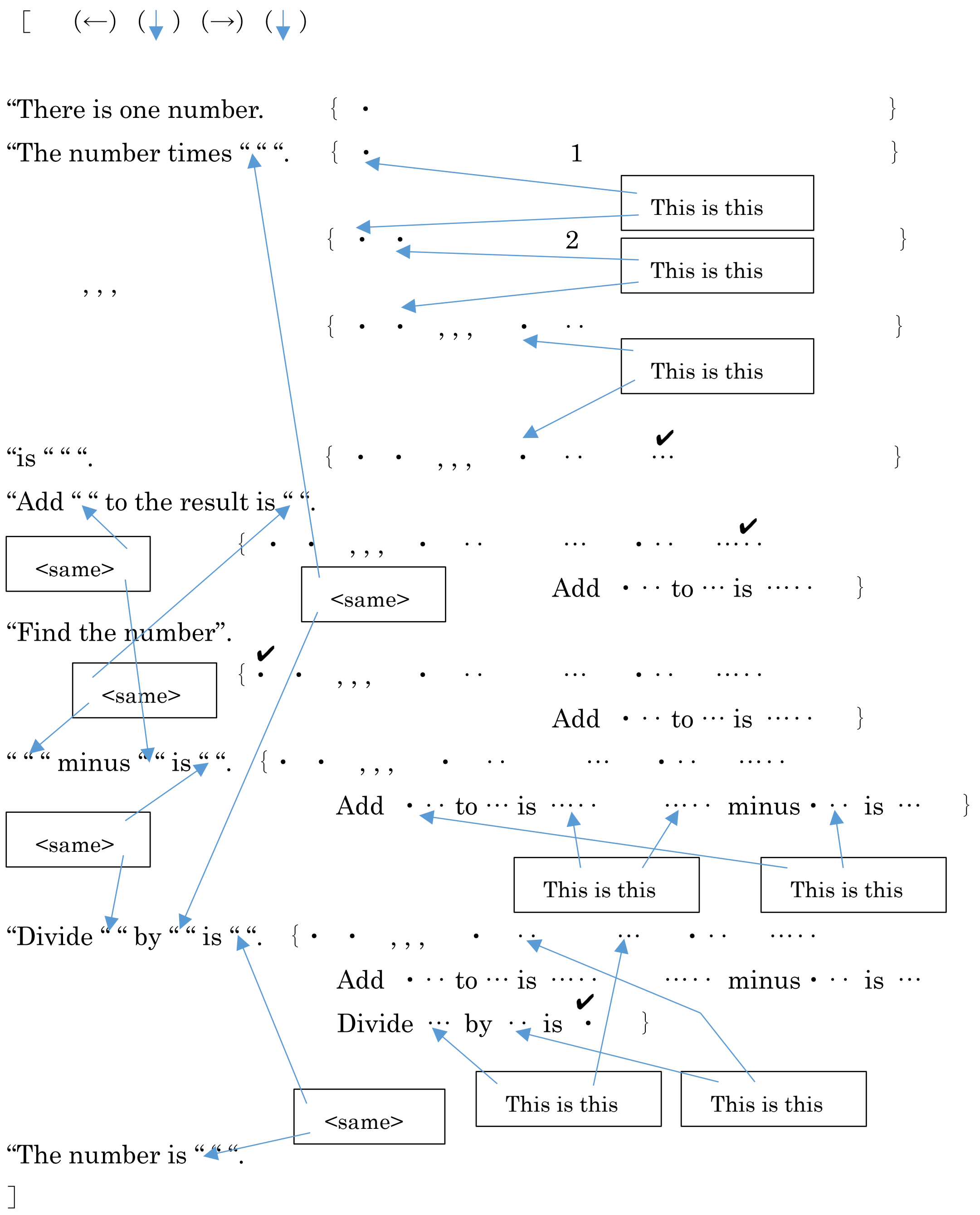
<same>

"Divide " " by " " is" . { ·· ·· , , , • ··· Multiply ·· and • is ···
Divide ··· by ·· is }

" ". { ·· ·· , , , • ··· Multiply ·· and • is ···
Divide ··· by ·· is ✓ • }

<same>

"The number is " " ".]



[(←) (↓) (→) (↓)

“There is one number. { • }

“The number times “ “ “. { • 1 }

This is this

{ • • 2 }

This is this

, , ,

{ • • , , , • · · }

This is this

“is “ “ “. { • • , , , • · · ✔ ··· }

“Add “ “ to the result is “ “.

<same>

{ • • , , , • · · ··· • · · ✔ ·····

<same>

Add • · · to ··· is ····· }

“Find the number”.

✔

{ • • , , , • · · ··· • · · ·····

<same>

Add • · · to ··· is ····· }

“ “ “ minus “ “ is “ “. { • • , , , • · · ··· • · · ·····

Add • · · to ··· is ····· ····· minus • · · is ··· }

<same>

This is this This is this

“Divide “ “ by “ “ is “ “. { • • , , , • · · ··· • · · ·····

Add • · · to ··· is ····· ····· minus • · · is ···

Divide ··· by · · is ✔ • }

<same> This is this This is this

“The number is “ “ “.

]

It tries to have another program P_b do as I do. It outputs strings with quotations that are bound to what I do, but finds some are not bound to strings with quotations. It tries to figure out strings and to bind what I do to the strings with quotations so that I can output the strings.

It gives three examples of the question and how to find the number to the other program P$_a$. Then it gives a new example to the program P$_a$. The program Pa finds the number.

It tries to explain what it does to another program P$_b$. It binds string with quotations when it finds strings that are not bound to strings with quotations.

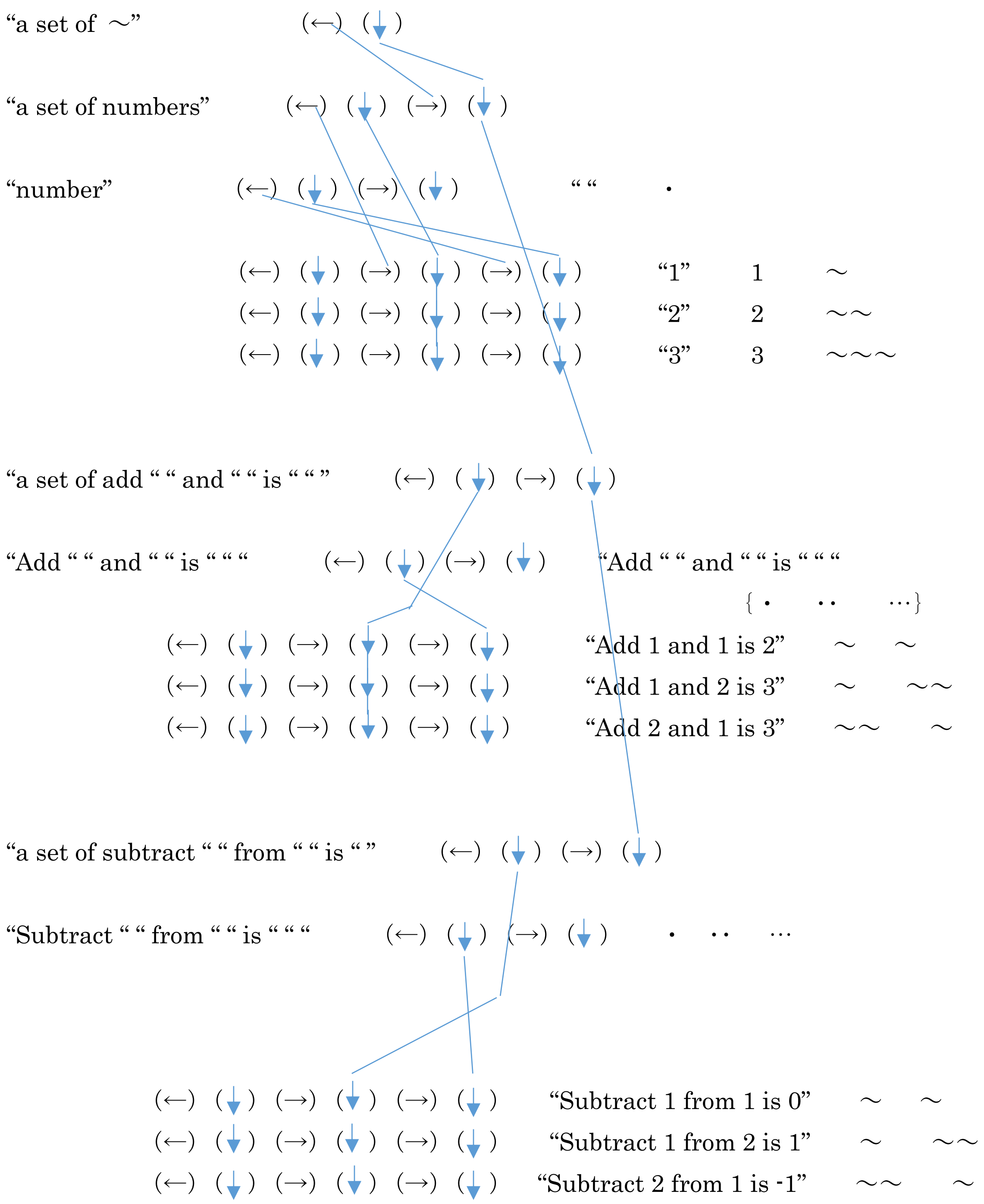
Later, the program devises regularities that are bound to strings "unkown" and "known" / "given". When the program devises the regularities, it uses the following lists.

(←) (↓) (→) (↓)
"There is one number. { ・ }
"The number times " " " { ・ 1 }
, , ,

(←) (↓) (→) (↓)
"There is one number. { ・ }
"The number times 2 " { ・ 1 }
, , ,

(←) (↓) (→) (↓)
"There is one number. { ・ }
"The number times 3 " { ・ 1 }
, , ,

A representative string and its members (examples)



“a set of ～” (←) (↓)

“a set of numbers” (←) (↓) (→) (↓)

“number” (←) (↓) (→) (↓) “ “ •

(←) (↓) (→) (↓) (→) (↓) “1” 1 ～

(←) (↓) (→) (↓) (→) (↓) “2” 2 ～～

(←) (↓) (→) (↓) (→) (↓) “3” 3 ～～～

“a set of add “ “ and “ “ is “ “ ” (←) (↓) (→) (↓)

“Add “ “ and “ “ is “ “ “ (←) (↓) (→) (↓) “Add “ “ and “ “ is “ “ “

{ • •• …}

(←) (↓) (→) (↓) (→) (↓) “Add 1 and 1 is 2” ～ ～

(←) (↓) (→) (↓) (→) (↓) “Add 1 and 2 is 3” ～ ～～

(←) (↓) (→) (↓) (→) (↓) “Add 2 and 1 is 3” ～～ ～

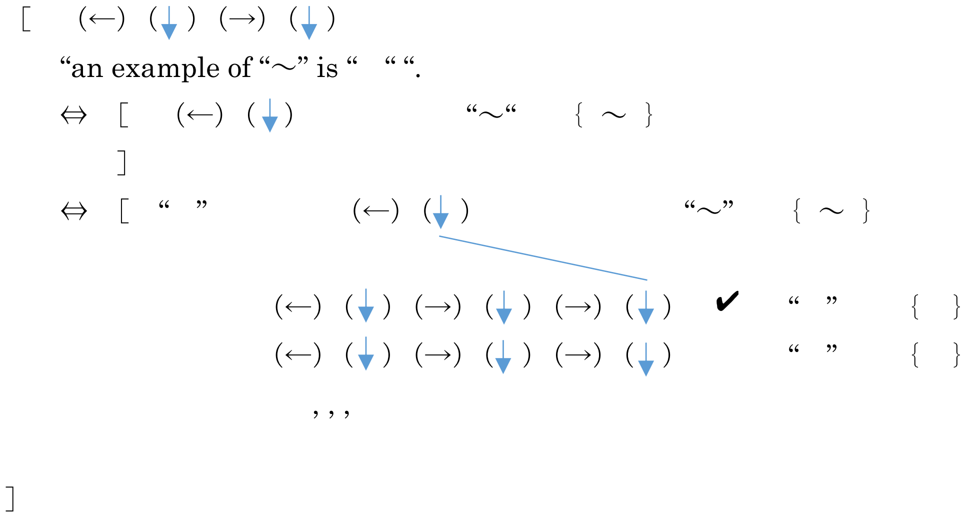
“a set of subtract “ “ from “ “ is “ ” (←) (↓) (→) (↓)

“Subtract “ “ from “ “ is “ “ “ (←) (↓) (→) (↓) • •• …

(←) (↓) (→) (↓) (→) (↓) “Subtract 1 from 1 is 0” ～ ～

(←) (↓) (→) (↓) (→) (↓) “Subtract 1 from 2 is 1” ～ ～～

(←) (↓) (→) (↓) (→) (↓) “Subtract 2 from 1 is -1” ～～ ～



The program gets “an example of object is ○”, “an example of object is □”, “an example of object is ▽”, and forms a regularity that has “an example of object is ∼” as a string with quotations.

Suppose the program gets “an example of object is”, it retrieves the regularity and tries to make “∼” be a member of “∼”. But it does not have a way to do so. Suppose “T” gives the program a member of “∼”, the program retrieves the member.

The program then makes a regularity that, given “an example of object is”, retrieves a member of “∼” that is in the link of members of the object.

Strings to see a member and a representative

When the program does not retrieve a way to find the number, it tries to form a way to find it. It also collects cases where the program does not find the number and cases where it finds the number. The program tries to form a hierarchical regularity that describes when it finds the number, and when it does not find the number. The program already formed a regularity that describes it reaches a goal state when it has a way and it does not when it does not have the way, and describes making the way to reach the goal state. The program replaces “find the number” by “reach a goal state”, and finds the current cases match the regularity already formed. It places “find the number” in “make the way to reach the goal state” and gets “make the way to find the number”. “Make the way to find the number” consists of having three examples of “find the number”.

Later the program has a goal to have another program P_a do “find the number”. But it does not have the other do “find the number”. It retrieves a regularity of changing states from not finding the number to finding the number. The regularity consists of a state not finding the number, having three examples of finding the number, and a state finding it.

The program replaces “I” by “one” and replaces “one” by P_a, and has P_a get three examples of finding the number. To achieve a goal of having P_a get three examples, the program replaces “one” by “I” and “I” by “Pa” in a regularity of “one has me get examples”, and gets a new regularity of “I have Pa get three examples”.

Further later the program has a goal to have P_a do to have P_b find the number, and finds Pa fails to do so. The program gives Pa how to have Pb make a way to find the number. It gives *Pa* “*Pa* has *Pb* get three examples”.

When the program finds *Pa* does not do it, the program retrieves a regularity of my making examples of finding the number, and replaces “I” by “Pa” to give *Pa* the regularity of “*Pa* makes examples”.

[(←) (↓) (→) (↓)

”An example of count how many “∼” is ”

⇔ [“count how many “∼“ “ (←) (↓) (→) (↓)

“1” {$\overset{✔}{\sim}\sim$, , , }

“2” {$\sim\overset{✔}{\sim}$, , , }

, , ,

“ “ { , , , $\sim\overset{✔}{\sim}$}

]

⇔ [“An example of ∼”

“∼” (←) (↓) “∼“ { ∼ }

(←) (↓) (→) (↓) (→) (↓) ✔ “ ” { }

(←) (↓) (→) (↓) (→) (↓) “ ” { }

, , ,

“is “ “ “.

]

“count how many “∼“ “ (←) (↓) (→) (↓)

“1” {$\overset{✔}{\sim}\sim$, , , }

“2” {$\sim\overset{✔}{\sim}$, , , }

, , ,

“ “ { , , , $\sim\overset{✔}{\sim}$}

(←) (↓) (→) (↓) (→) (↓) ✔ “1” {$\overset{✔}{\sim}$ }

(←) (↓) (→) (↓) (→) (↓) “1” {$\overset{✔}{\sim}\sim$ }

“2” {$\sim\overset{✔}{\sim}$ }

(←) (↓) (→) (↓) (→) (↓) “1” {$\overset{✔}{\sim}\sim\sim$ }

“2” {$\sim\overset{✔}{\sim}\sim$ }

“3” {$\sim\sim\overset{✔}{\sim}$ }

, , ,

]

[(←) (↓) (→) (↓)

“An example of multiply a number and a number is “

⇔ [(←) (↓) (→) (↓)

“Multiply “ “ and “ “ is “ { • ·· } (next)

“ “ { • ·· …}

]

⇔ [“An example of ～”

“～” (←) (↓) (→) (↓) “ “ { }

(←) (↓) (→) (↓) (→) (↓) ✔ “ ” { }

(←) (↓) (→) (↓) (→) (↓) “ ” { }

, , ,

“is “ “ “.

]

(←) (↓) (→) (↓) “Multiply a number and a number is” (next)

“a number”.

(←) (↓) (→) (↓) (→) (↓) ✔ “Multiply 1 and 1 is 1” { }

(←) (↓) (→) (↓) (→) (↓) “Multiply 1 and 2 is 2” { }

, , ,

(←) (↓) (→) (↓) (→) (↓) “Multiply “ “ and “ “ is “ “ ” { }

“Multiply 1 and 1 is” { 1 1 } (next)

“1” { 1 1 1 }

]

[

“an example of a number is”

⇔ [“An example of ～”

“～” (←) (↓) (→) (↓) “ “ { }

(←) (↓) (→) (↓) (→) (↓) ✔ “ ” { }

(←) (↓) (→) (↓) (→) (↓) “ ” { }

, , ,

“is “ “ “.

]

⇔ [(←) (↓) (→) (↓) “a number”

]

(←) (↓) (→) (↓) “a number”.

(←) (↓) (→) (↓) (→) (↓) ✔ “1” { }

(←) (↓) (→) (↓) (→) (↓) “2” { }

, , ,

(←) (↓) (→) (↓) (→) (↓) “ “ { }

“1”

]

Suppose that the program gets input strings. Then the program sees if a sub string of the input matches a string of a member of a regularity R_{sub} in its memory. If the sub string matches the string, the program replaces the sub string by a representative R_{sub} of the string by using a pointer (→) to its representative. The program then sees if the input strings with replacements, match strings of a regularity R in its memory, and if the input match the strings, it retrieves the strings from the memory.

As the input strings continue, the program sees if the input keep matching the retrieved. When the input strings end and the retrieved strings continue further than the input, the program sees if a sub string of the retrieved is a representative of some members. It tries to replace the sub string by a specific member (↓) of the representative. The program determines the specific member 1) by relations <this is this> and <same>, 2) by a sequence of relations <this is this> and <same>, 3) by using strings of the retrieved as a regularity already formed. For example, suppose the program retrieves strings “ “ “$_1$ minus “ “$_2$ is “ “$_3$ “, and has “ “$_1$ and “ “$_2$ determined, then the program uses the strings to determine “ “$_3$, namely it conducts a minus operation.

Here, the program finds relations <this is this> and <same> in input strings and keeps the relations to form a regularity out of the input strings.

A way to find the number

This section describes how the program forms a regularity specifying either what strings make one be able to find the number, or what strings make one not be able to find the number. Suppose the program gets various questions such as “There is one number. Add 5 to the number makes 12. Find the number”, the program is able to find the number when it has formed a way to solve it, and it cannot otherwise. An issue here is how the program finds the fact (or regularity) that the program is able to find the number when it has formed a way to solve it, and it cannot otherwise.

Suppose the program gets a question of finding the number after the program keeps in its memory sequences of cases where the program can find the number and cases where it cannot find the number. Then it tries to find the number, but suppose it fails to do so. It tries to find the number (reach its goal) in two directions; 1) it further retrieves regularities to combine them to find a way to the goal, 2) it places focus on “find the number” and “does not find the number”, namely a specific case of the possible inconsistent: cases of reaching the goal and cases of not reaching the goal. When the program takes the second option, it drops strings specifying indivisual questions, namely replacing individual strings by their representatives. Then the program finds there exist sequences of strings that ends with strings “the number is “ “ “, its goal, and there does not exist such sequences of strings when it does not find the number. It forms a regularity that there exist sequences of strings that ends with strings “the number is “ “ “ when it finds the number, and there does not exist such sequences of strings when it does not find the number.

Later the program forms a regularity of changing states from does not exist sequences to exist sequences. In other words make a way to find the number, but strings with quotations are not bound to make a way to find the number.

The next issue is how the program bind strings with quotations to steps of the regularity just formed. Suppose the program have another program P_a find the number, the other program P_a cannot find the number. A goal is to have the other program find the number. The program retrieves a regularity that matches the current states with replacing “I” by “Pa”; the current are “do not find the number”, and “the goal, find the number”.

It makes the current be consistent with the retrieved, namely conducts the regularity by replacing “I” by the other program “Pa” and “T” by “I”. The program binds a string “make a way to find the number” to the regularity, and have the other program Pa make a way to find the number. Then have the other program find the number.

5. Conclusion

This paper describes a case where a program outputs its original (or innate) capabilities in strings acquired after it runs, in English. The original capabilities are written in a programming language. A capability is, given inputs, finds sub strings that match strings already acquired, and replaces them by their representatives; for example, given "Add 4 and 3 is 7", the program replaces "4", "3", and "7" by "a number" after the program has formed a set including "4", "3", and "7", and binds "a number" to their representative.

What the program does are 1) it extracts strings common in pairs of inputs and outputs (strings of regularities), 2) it binds English strings to the common strings, and 3) it outputs English strings. The strings common in the pairs describe what the program does to given inputs. Therfore the program describes in English what it does to given inputs, in other words, capabilities written in a programming language. But the program does not know yet that the capabilities described are original (or innate) ones.

Although this paper describes only a case of saying original capabilities of a program in natural language, it appears to show an important direction; namely, a machine on which the program runs may become a device that acquires ways of counting, numbers, and computing as well as their meanings. Most people think any currently available computer with programs have no meanings of counting, numbers, or computing in the computer or the programs. But the paper shows a possible way to make a device become able to have meanings of counting, numbers, and computing.

Much needs to be explored to construct a program that describes, in a natural language, its capabilities in more cases and in more detail than the case described here; 1) Study a structure (or an architecture) of a machine such as channels through which strings are given and output, and a way of segmenting and sequencing strings. 2) Sets formed by the progam need to be sorted when many sets are formed. An architecture and techniques to sort them out need to be investigated. 3) Perspectives to classify capabilities need to be introduced; in particular, the capabilities should be classified into two, original and acquired, in other words, the program becomes to see if its capability is original (or innate) or acquired. 4) Devise a structure and methods of ordering sub strings (keeping a syntax) to make strings each of which is bound to a step of doing things.

Bibliography

Cook, V. J. and Newson, M. (2007). Chomsky's Universal Grammar: An Introduction. Blackwell Publ.

Cypher, A. (1993). Introduction: Bringing Programming to End Users. In A. Cypher. (Ed.). Watch what I do: Programming by Demonstration. MIT Press.

Güzerdere, G. (1996). Consciousness and Introspective Link Principle. In S. R. Hameroff, A. W. Kaszniak, A. C. Scott. (Eds.). Toward a Science of Consciousness. The first Tucson Discussions and Debates. MIT Press.

Hadamard, J. (1945). The Psychology of Invention in the Mathematical Field. Princeton U. Press.

Haikonen, P. O. (2003). The Cognitive Approach to Conscious Machines. Imprint Academic.

Harnad, S. (1990). Symbol grounding problem. Physica D 42, 335-346.

Holland, J. Holyoak, K. Nisbett, R. and Thagard, P. (1986). Induction: Processes of Inference, Learning, and Discovery. MIT Press.

Holland, O. (Ed.) (2003). Machine Consciousness. Imprint Academic.

Kitzelmann, E. and Schmid, U. (2006). Inductive Synthesis of Functional Programs: An Explanation Based Generalization Approach. J. Machine Learning Research, 7, 429-454.

Lavrač, N. and Džeroski, S. (1994). Inductive Logic Programming: Techniques and Applications. Ellis Horwood.

Lucas, J. R. (1964). Minds, Machines and Gödel. In A. R. Anderson (Ed.). Minds and Machines. Prentice-Hall, Inc.

McDermott, D. (2001). Mind and Mechanism. MIT Press.

Minsky, M. (1968). Semantic Information Processing. MIT Press.

Mitchell, T. M. (1997). Machine Learning. McGrow-Hill Companies.

Muggleton, S. (1991). Inductive logic programming. New Generation Computing, 8, 295-318.

Pfeifer, R. and Scheier, C. (2001). Understanding Intelligence. MIT Press.

Polya, G. (1953). Induction and Analogy in Mathematics. Princeton U. Press.

Schmid, U. (2003). Inductive Synthesis of Functional Programs. LNAI 2654, Springer.

Searle, J. R. (1992). The Rediscovery of the Mind. MIT Press.

Shieber, S. (Ed.) (2004). The Turing Test. MIT Press.

Smith, D. (1984). The synthesis of LISP programs from examples: A survey. In A. Biermann, G. Guiho, and Y. Kodratoff (Eds.). Automatic Program Construction

Techniques. Macmillan Publ.

Tall, D. (2013). How Humans Learn to Think Mathematically. Cambridge U. Press.

Tallis, R. (1994&2004). Why the Mind is Not a Computer. Imprint Academic.