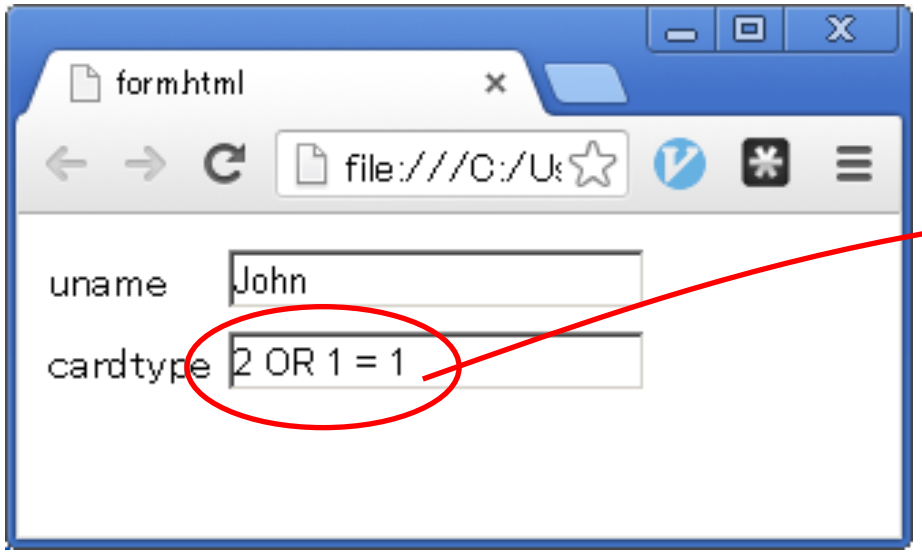
形式検証による
コマンド・インジェクション攻撃対策

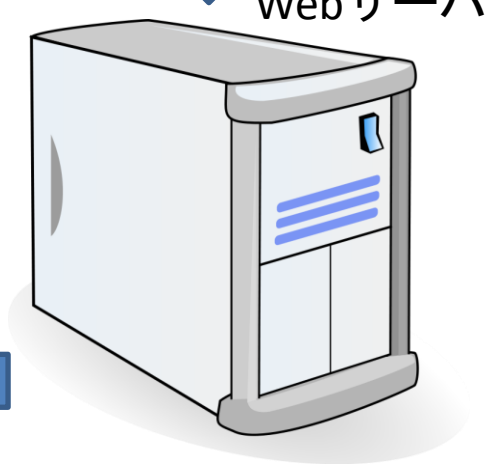
産総研　平井洋一

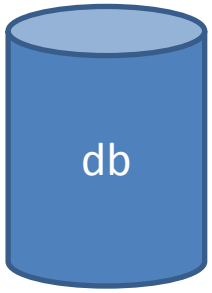
2013-09-11　応用数理学会年会
数理的技法による情報セキュリティ

コマンド・インジェクション攻撃(例)



悪のHTTPリクエスト



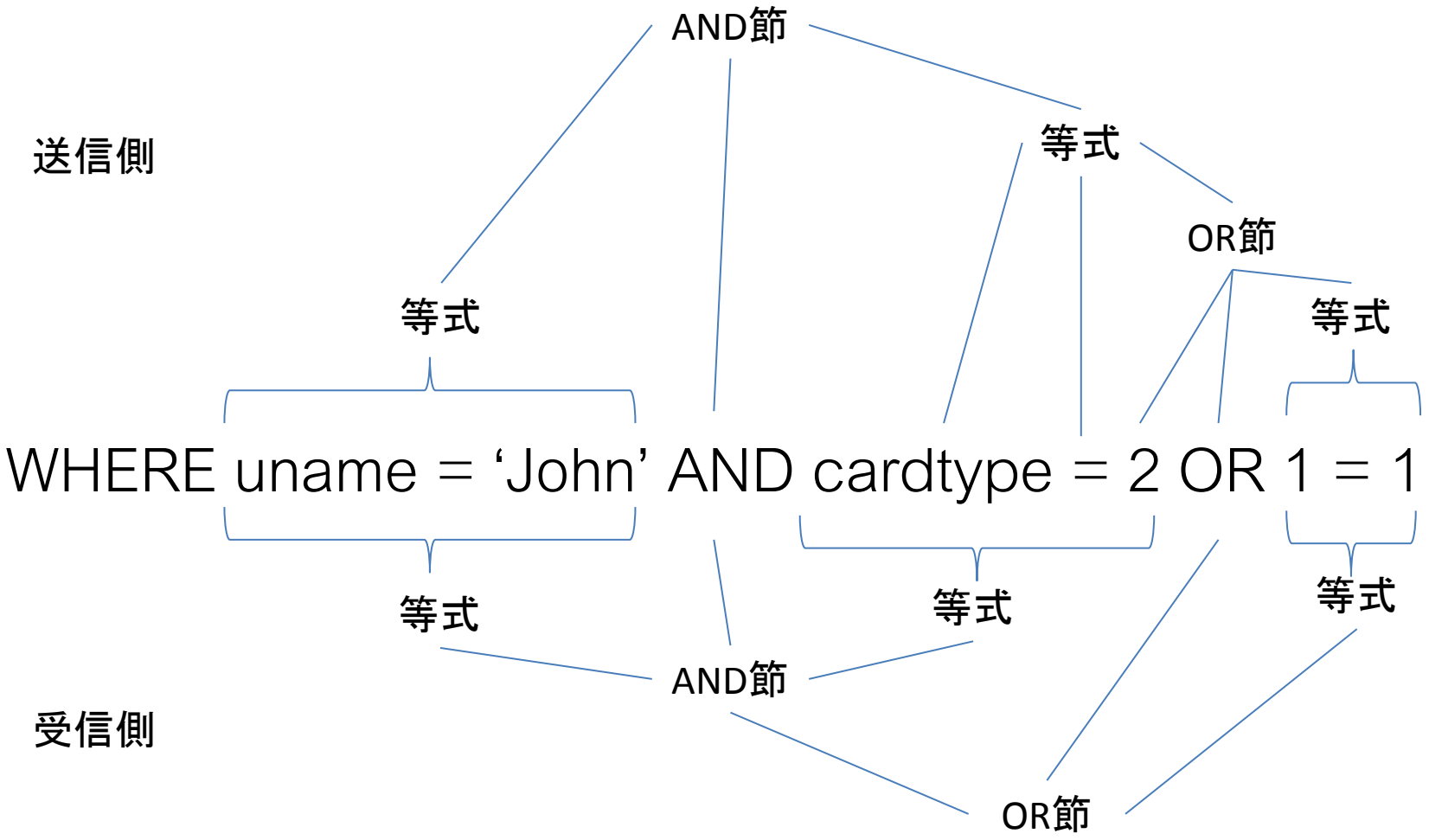


全データを削除するSQL文

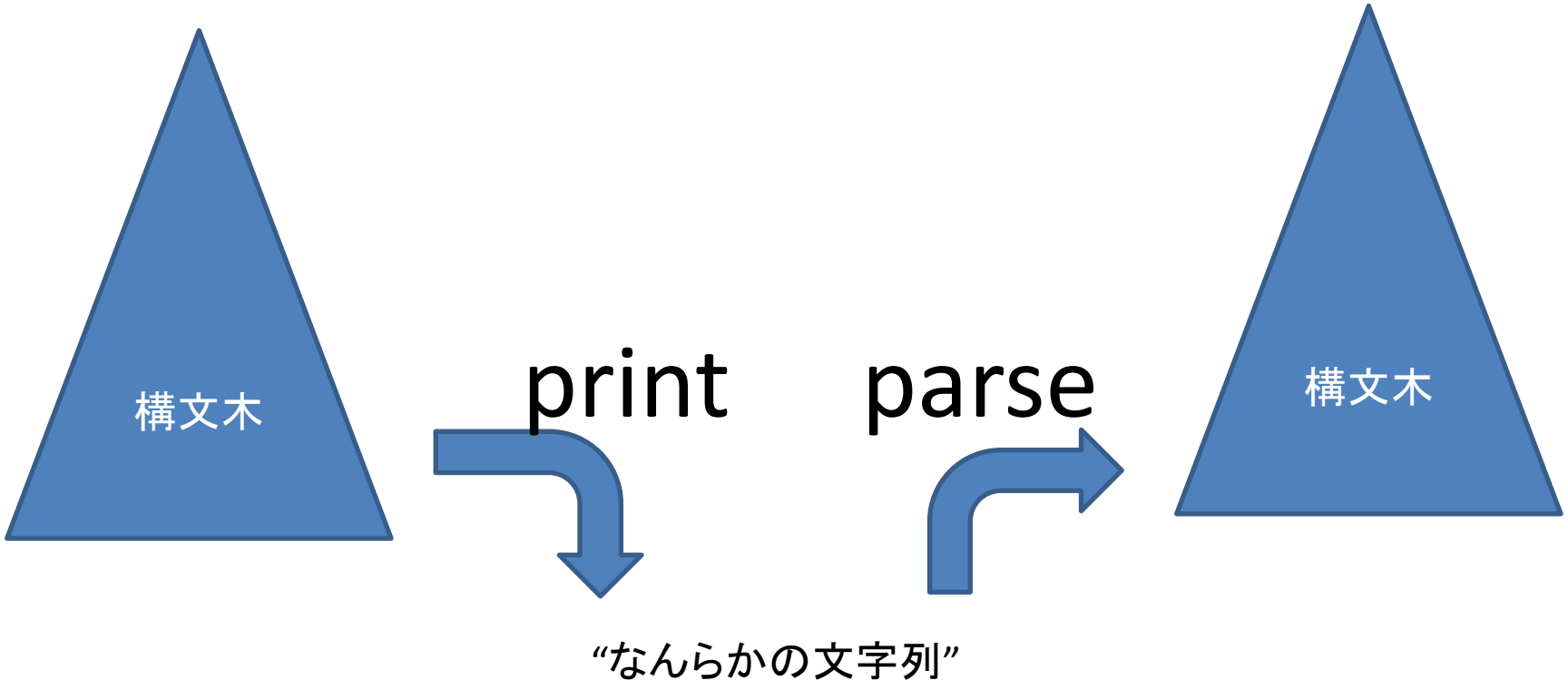
DELETE FROM table

WHERE uname = ‘John’ AND cardtype = 2 OR 1 = 1

送信側受信側での構文木の不一致



目標: printしてparseすると元に戻る



Coqを使う

- Coqでプログラムを書けるなら、
  型が決まるはず
  プログラム : 型

例

print : data -> string

関数　左が入力、右が出力

parse : string -> option data

あるかないか

途中まで読める型

- 文字列全体を消費せず、一部だけ読む

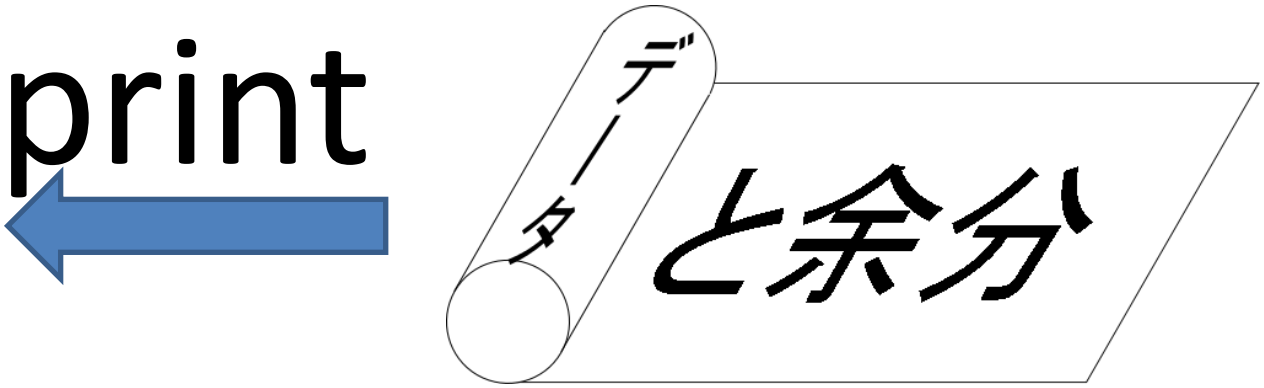
parse : string -> option (data * string)

入力:

出力(成功時):

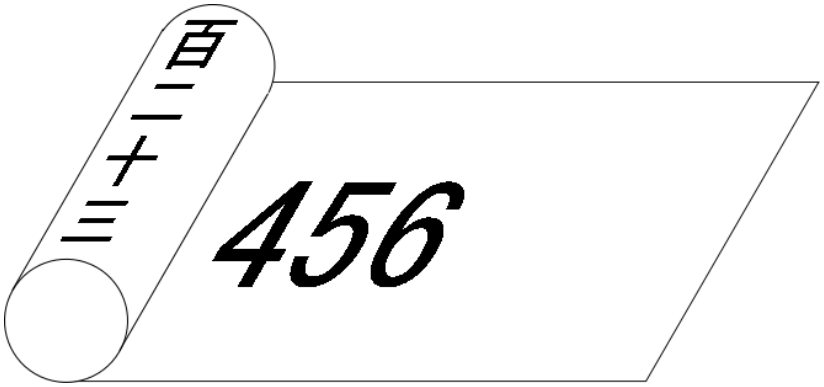
目標は可逆性

print



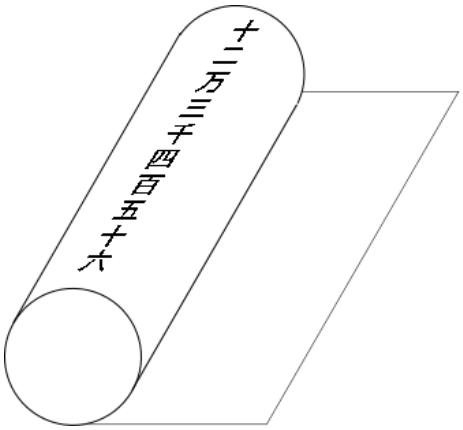
parse

可逆性、強すぎでは



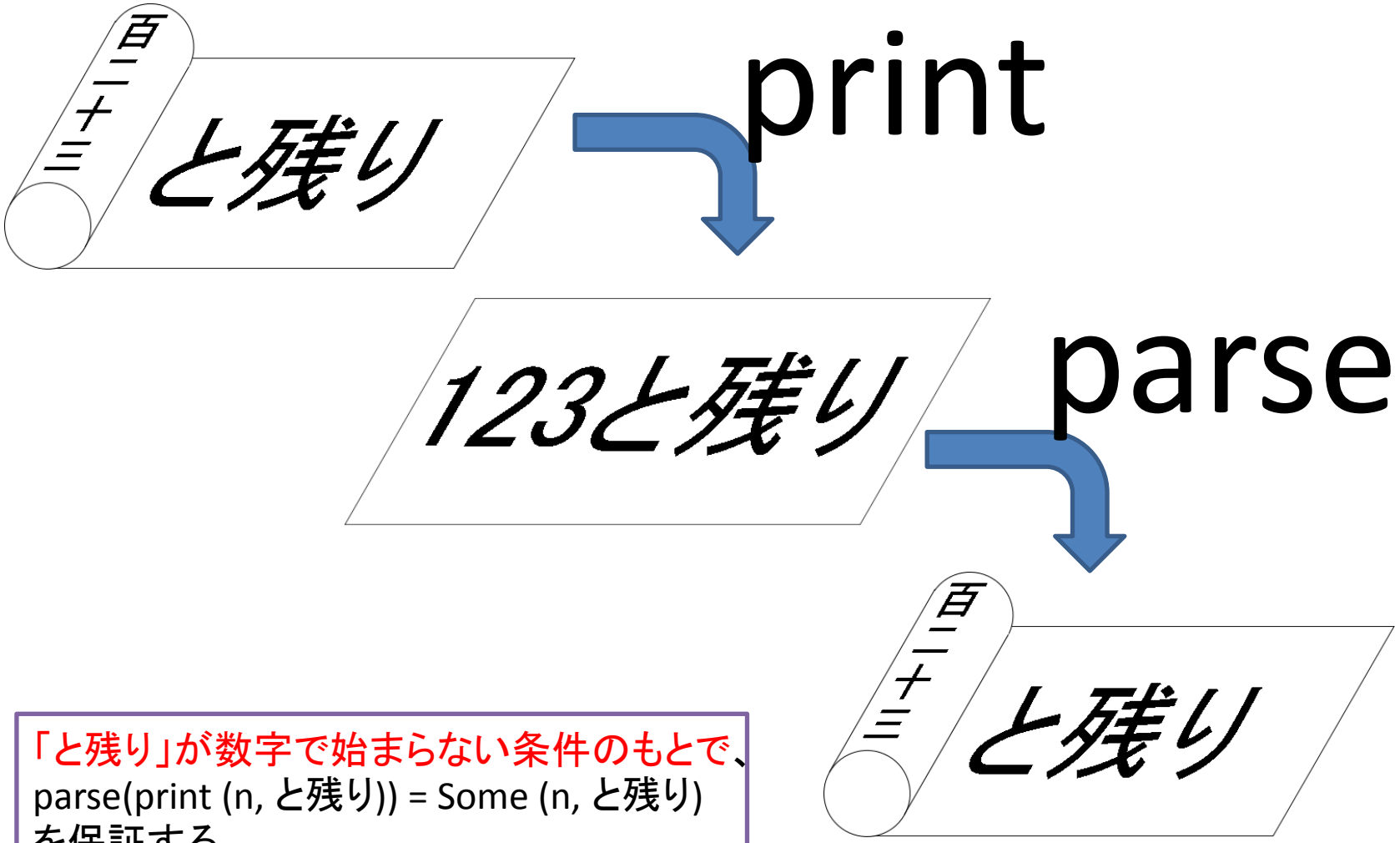
print

parse



数字の列を素直にprint、parseすると、
可逆性を満たさない。
可逆性は不便。

可逆性を制限したければ制限する



「と残り」が数字で始まらない条件のもとで、
parse(print (n, と残り)) = Some (n, と残り)
を保証する

可逆読み書き器の正体

```
Record syntax A spec precond :=

  { print : (A * string) -> string;

    parse : string -> option (A * string);

    reverse : forall a str,
                precond (a, str) ->
                parse (print (a, str)) = Some (a, str);

}.
```

書き器

読み器

条件付きの

可逆性

繰り返しパーズの停止性

- 「数字を読める限り読み続けた結果」
  いつまで読み続けても読め続けたらどう定義しよう
- 「数字を読むと文字列が短くなる」等の性質を使うと、読み終わることを保証できる

可逆性以外に、parse成功時の入出力の関係も
何か保証しておきたいことがある。

可逆読み書き器の正体

```
Record syntax A spec precond :=

  { print : (A * string) -> string;

    parse : string -> option (A * string);

    reverse : forall a str,
                precond (a, str) ->
                parse (print (a, str)) = Some (a, str);

    spec : forall str a rest,
                parse str = Some (a, rest) -> spec str (a, rest) }.
```

書き器

読み器

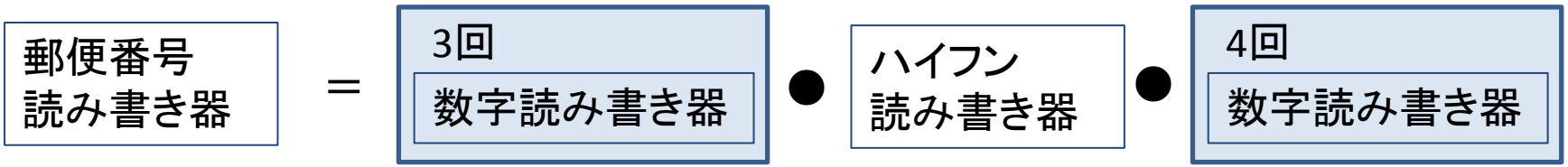
条件付きの

可逆性

parseが成功したときに

仕様specをみたす

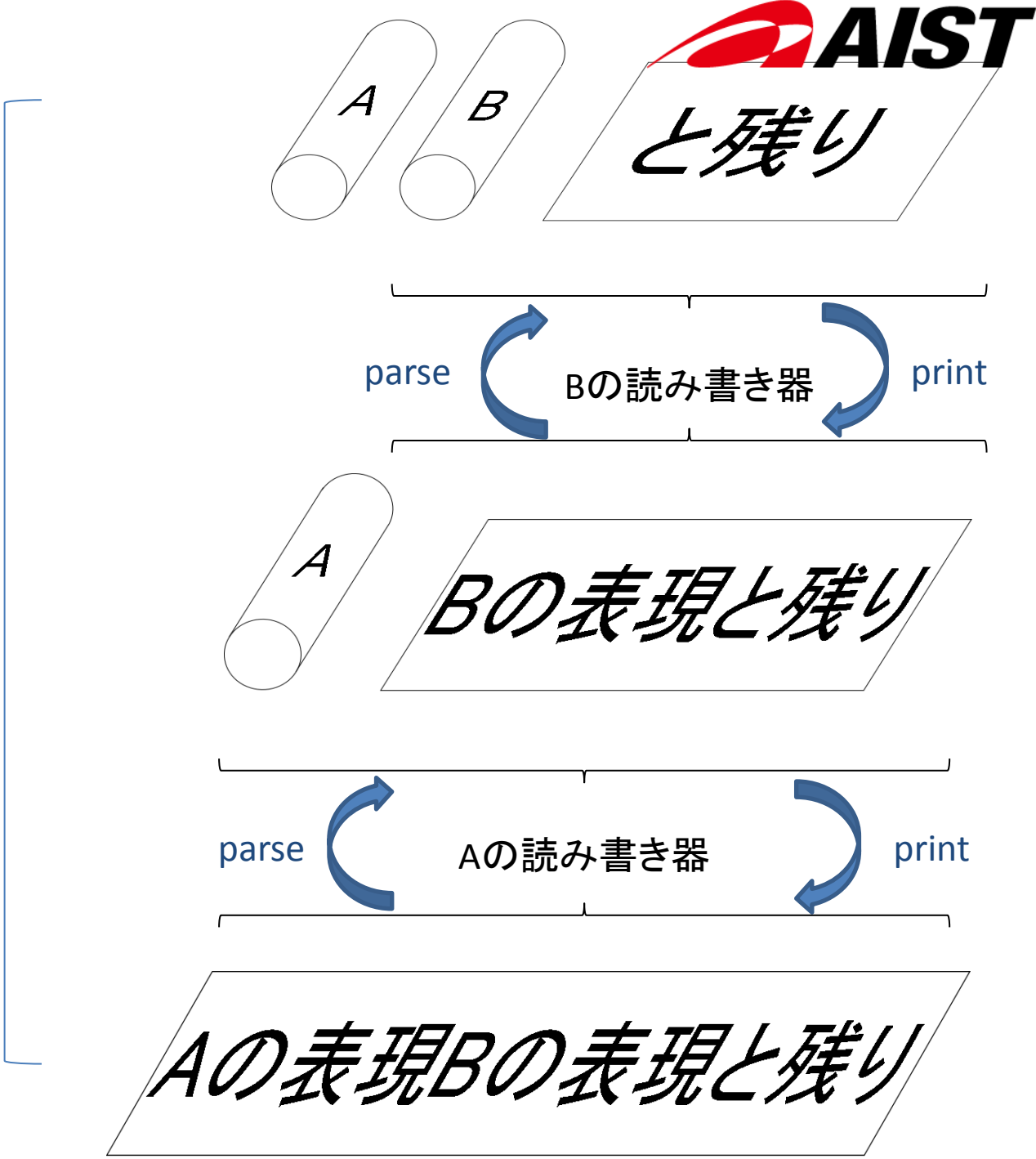
組み合わせたい



部品を再利用すると、文明が蓄積する

連接

AとBの組の
読み書き器



連接の記号 〈*〉

- 記号はinvertible syntax descriptionsから来た

Aの読み書き器

```
syntax A P Q
print   : (A * string) -> string
parse   : string -> option (A * string)
reverse : Pを満たす(a, str)を
          printしparseすると戻る
spec    : parse成功時にQを満たす
```

〈*〉

Bの読み書き器

```
syntax B P' Q'
print'   : (B * string) -> string
parse'   : string -> option (B * string)
reverse' : P'を満たす(b, str)を
           printしparseすると戻る
spec'    : parse'成功時にQ'を満たす
```

=

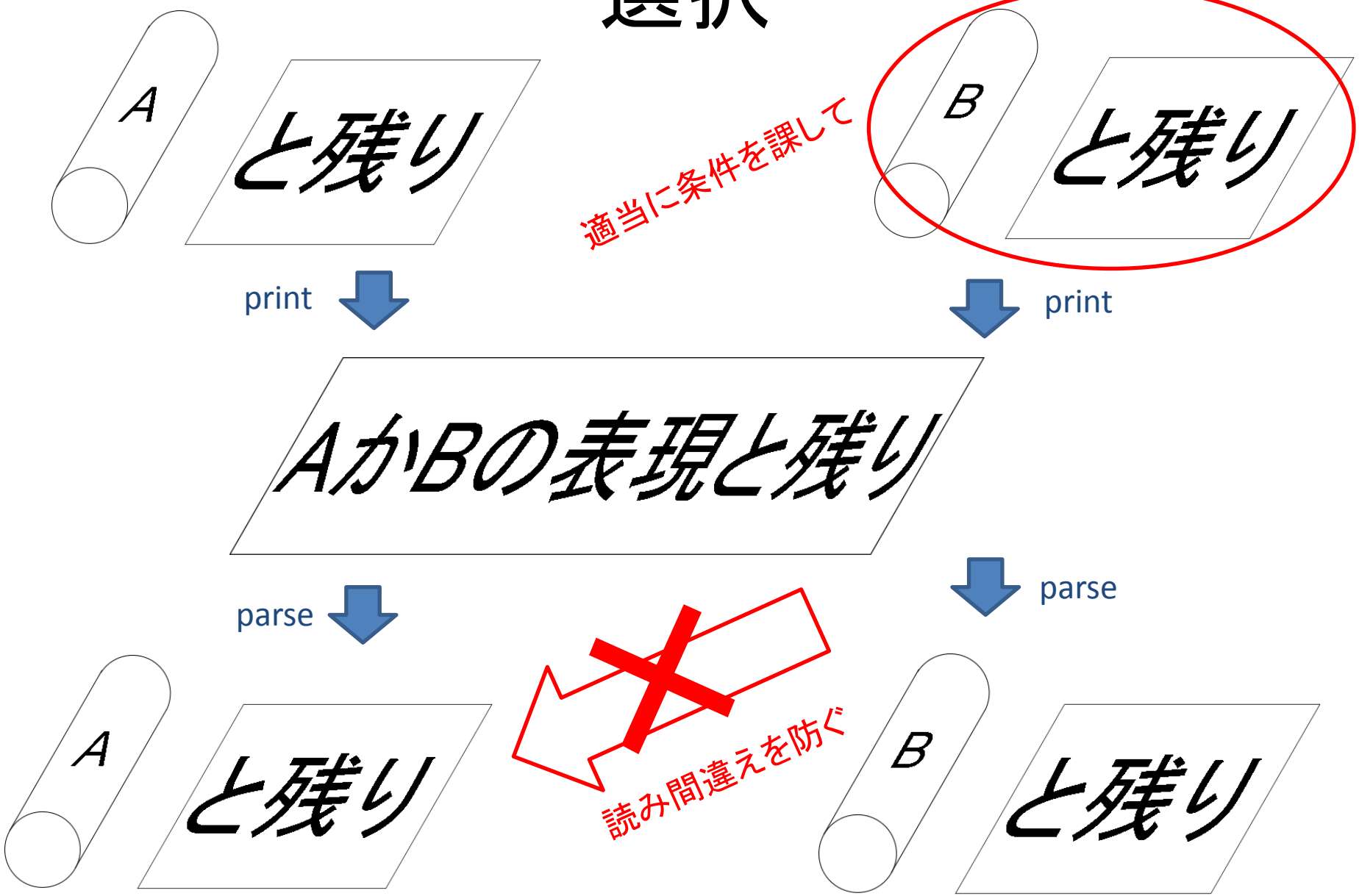
AとBの組の読み書き器

```
syntax (A * B) ??  ???
print†   : (A * B * string) -> string
parse†   : string -> option (A * B * string)
reverse† : ??を満たす(a, b, str)を
           print†しparse†すると戻る
spec†    : parse†成功時に???を満たす
```

(b, str)がP'を満たす　かつ
Q' mid_str (b, str)ならば
(a, mid_str)がPを満たす

QとQ'の、関係としての合成

選択


A
と残り
適当に条件を課して
B
と残り
print
print
AかBの表現と残り
parse
parse
A
と残り
読み間違えを防ぐ
B
と残り

選択の記号 <|>

- 記号はinvertible syntax descriptionsから来た

Aの読み書き器

```
syntax A P Q
print   : (A * string) -> string
parse   : string -> option (A * string)
reverse : Pを満たす(a, str)を
          printしparseすると戻る
spec    : parse成功時にQを満たす
```

<|>

Bの読み書き器

```
syntax B P' Q'
print'   : (B * string) -> string
parse'   : string -> option (B * string)
reverse' : P'を満たす(b, str)を
           printしparseすると戻る
spec'    : parse'成功時にQ'を満たす
```

=

AかBの読み書き器

```
syntax (A + B) ??  ???
print†   : ((A + B) * string) -> string
parse†   : string -> option ((A + B) * string)
reverse† : ??を満たす(inl b, str)を
           print†しparse†すると戻る (残半分)
spec†    : parse†成功時に???を満たす
```

bがP'を満たす　かつ
printしてparseして
Aだと勘違いされない

QとかQ'とか

推論して条件の形をきれいにしたい

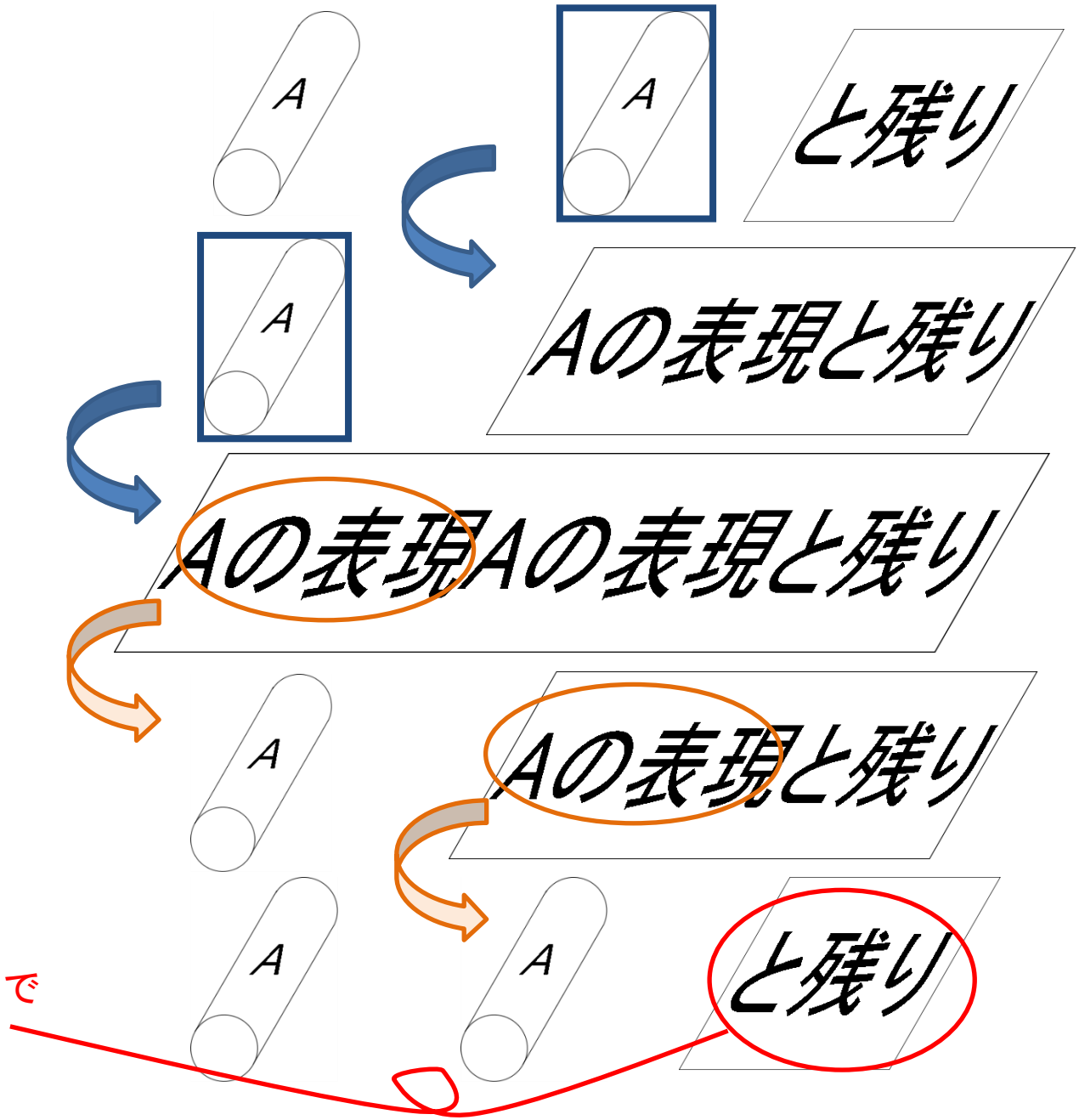
- invertible syntax descriptionの人は
  仕様も条件も考えなかったので記号は無し

syntax A　きれいな条件P'　きれいな仕様Q'

(P'ならばPだとか、QならばQ'だとか)

syntax A　きたない条件P　きたない仕様Q
print　: (A * string) -> string
parse　: string -> option (A * string)
reverse : Pを満たす(a, str)を
　　printしparseすると戻る
spec　: parse成功時にQを満たす

繰り返し



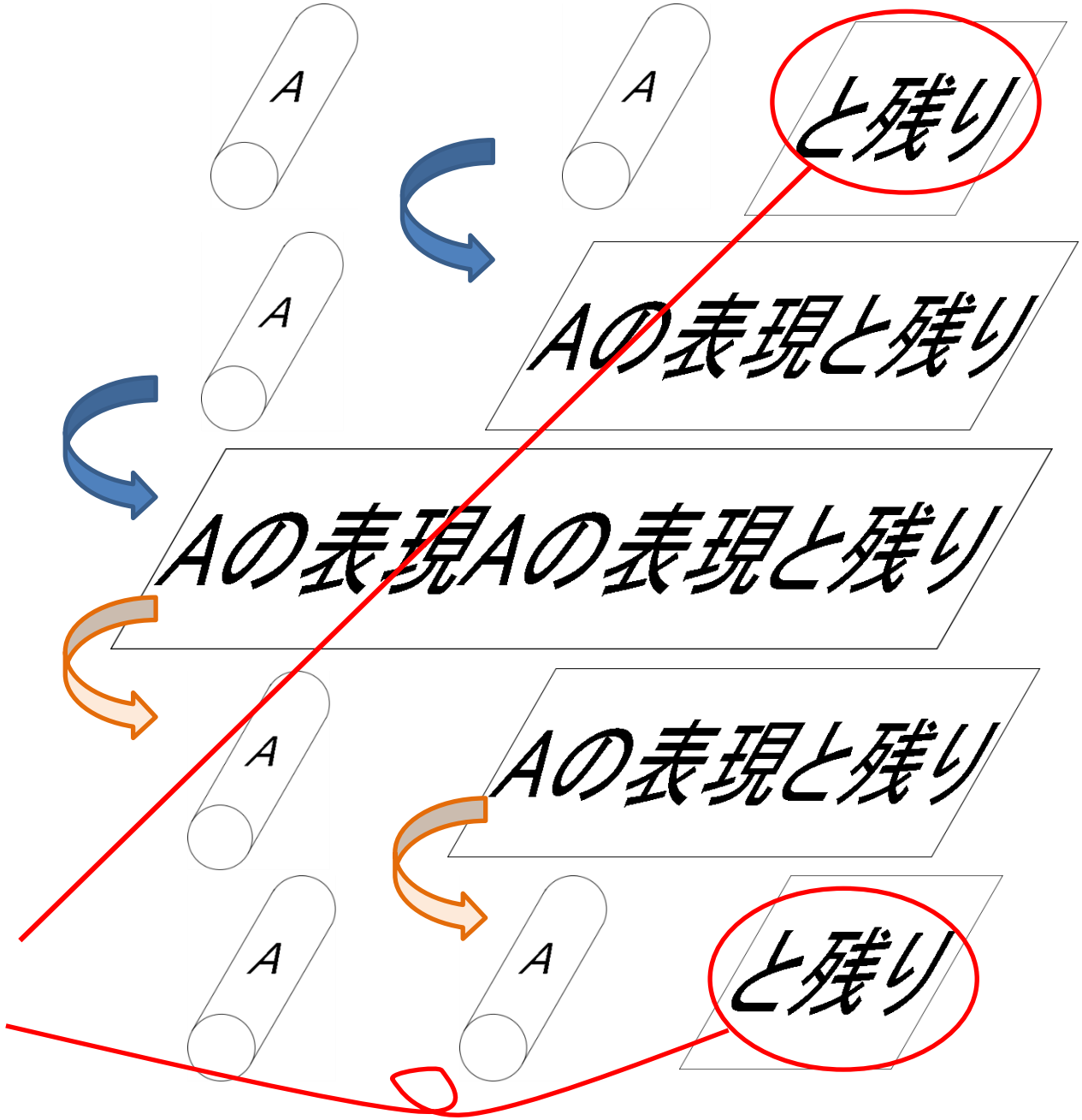
もうAを読み出せないので
終了

繰り返し many

- 繰り返し要素の可逆読み書き器があって、
- parseするたびに減っていく量があれば、
- 繰り返し要素のリストの可逆読み書き器ができる。

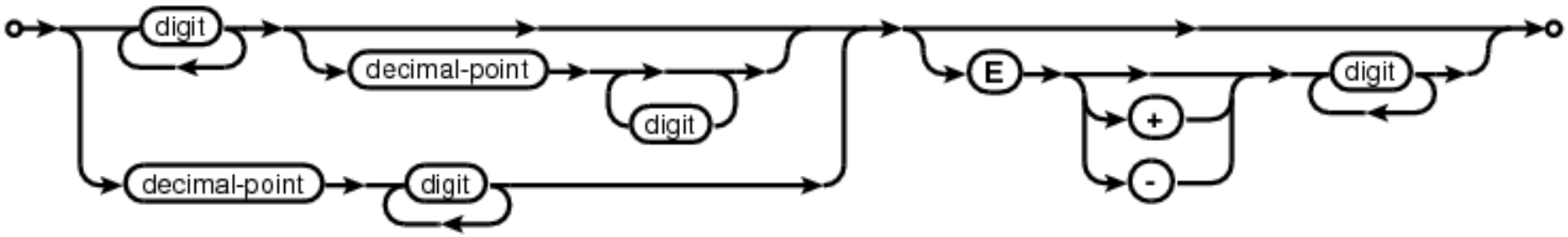
- 可逆性の事前条件に、
「繰り返し要素を読み出せない」が加わる

繰り返し



Aを読み出せない

数リテラル[SQLiteのドキュメントから]



できた。

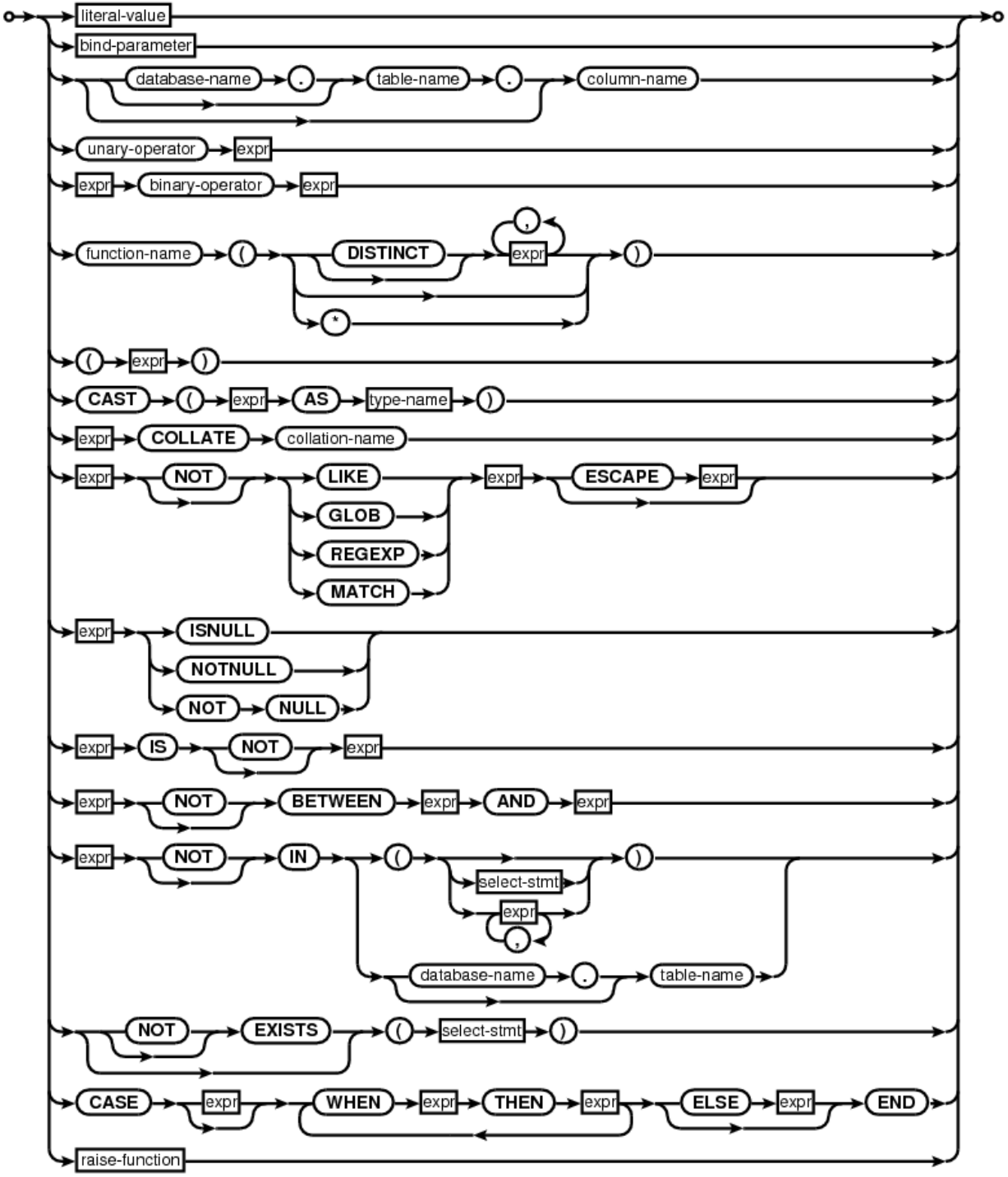
Eval vm_compute in get numeric_literal ".4E-55 + 3”.

= Some
(Decimal (four, [::]), Some (Some negative, (five, [:: five])),
" + 3"%string)
: option (num_literal_data * string)

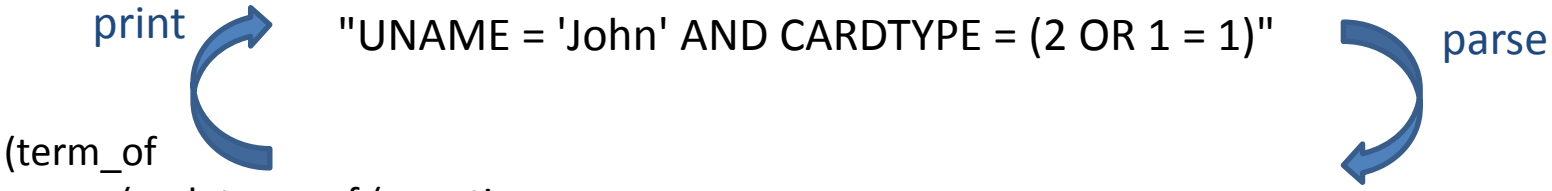
SQLの式

全部はできていない。
できるはず。

再帰の深さは文字列の
長さ程度



可逆読み書きの例




```
(term_of
    (and_term_of (equation
            (elm_atom (column_atom uname), elm_atom (str_atom "John")),
          [:: equation
            (elm_atom (column_atom cardtype),
             parens
              (term_of (and_term_of
                      (elm_direct
                        (elm_atom
                          (number_atom
                            (Usual (two, [::], None), None))),
                       [::]),
                     [:: and_term_of
                        (equation
                          (elm_atom
                            (number_atom
                              (Usual (one, [::], None), None)),
                           elm_atom
                            (number_atom
                              (Usual (one, [::], None), None))),
                       [::])])))]), [::]), "").
```

できたこと

- 可逆読み書き器を実装した
  - 可逆性をCoqで検証
- 部品を組み合わせられる
  - 連接
  - 選択
  - 繰り返し

Coqで1,000行程度(汎用の可逆計算ライブラリ含む)

既存研究: invertible syntaxの検証

PLPV 2012

- できる
  連接　繰り返し
- できない
  選択

Formal Network Packet Processing with Minimal Fuss * †

Invertible Syntax Descriptions at Work

Reynald Affeldt† David Nowak‡ Yutaka Oiwa†

†Research Center for Information Security ‡Information Technology Research Institute
National Institute of Advanced Industrial Science and Technology, Japan
{reynald.affeldt,david.nowak,y.oiwa}@aist.go.jp

Abstract

An error in an Internet protocol or its implementation is rarely benign: at best, it leads to malfunctions, at worst, to security holes. These errors are all the more likely that the official documentation for Internet protocols (the RFCs) is written in natural language. To prevent ambiguities and pave the way to formal verification of Internet protocols and their implementations, we advocate formalization of RFCs in a proof-assistant. As a first step towards this goal, we propose in this paper to use invertible syntax descriptions to formalize network packet processing. Invertible syntax descriptions consist in a library of combinators that can be used interchangeably as parsers or pretty-printers: network packet processing specified this way is not only unambiguous, it can also be turned into a trustful reference implementation, all the more trustful that there is no risk for inconsistencies between the parser and the pretty-printer. Concretely, we formalize invertible syntax descriptions in the Coq proof-assistant and extend them to deal with data-dependent constraints, an essential feature when it comes to parsing network packets. The usefulness of our formalization is demonstrated with an application to TLS, the protocol on which e-commerce relies.

Categories and Subject Descriptors D.3.2 [*Programing Languages*]: Language Classifications—Applicative (functional) languages; C.2.2 [*Computer-Communication Networks*]: Network Protocols—Protocol verification

the so-called Requests for Comments (RFCs). They are the de facto standards for the development of network applications but since they are written in natural language, developers are sometimes led to resolve inherent ambiguities by reading the source code of other existing implementations. This "deciphering" task is all the more difficult that the specifications of Internet protocols are long and complex. For example, the specification of the Transport Layer Security protocol (TLS), the Internet protocol that provides privacy and data integrity to most e-commerce applications, consists of 104 pages [4], and is not self-contained. At best, errors in TLS implementations can disrupt the usage of network applications; at worst, they can be exploited by malicious users. In order to avoid these problems, we advocate the use of formal specification for Internet protocols. A formal specification not only eliminates ambiguities, but it also paves the way to formal verification of the protocol and its implementation.

As a first but substantial step towards formal specification of Internet protocols, we propose to use *invertible syntax descriptions* [18] to formalize network packet processing. Invertible syntax descriptions consist of a set of combinators that can be used interchangeably as parsers or pretty-printers (hereafter, printers). By formalizing network packet processing with invertible syntax descriptions, we actually fulfill two important objectives simultaneously: (1) we obtain an unambiguous grammar for the syntax of network packets, and (2) thanks to combinators, this formalization