A Grammar-based Approach to Program Inversion

Kazutaka Matsuda (Univ. of Tokyo)
joint work with
Shin-Cheng Mu, Zhenjiang Hu, Masato Takeichi

Mar. 13th, 2010. 4th BT-in-ABC WS

Background

- Writing a program that is inverse to some other program, such as ...
  - redoing for undoing
  - serialization for deserialization
  - implementation of constant complement

    [Bancilhon&Spyratos: TODS81] BT.

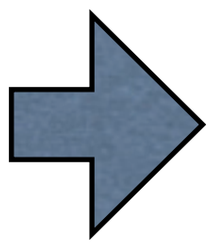
- ... is quite tedious and error-prone.

Program Inversion

Given a program describing function f,
find a program describing f^{-1}.

e.g.

Program of
`snoc(x,b)`
`= x++[b]`



Inv. Prog. of
`snoc`$^{-1}$`(x++[b])`
`= (x,b)`

[Dijkstra: PC78, Gries: 81, Glück&Kawabe: FLOPS04, ...]

Issues of Program Inversion

- For efficient inverses, an inversion method often targets "some" progs.
  - … instead of all.
    - cf. breadth first search
- … but what is the "some"?

```
reverse(x) = rev(x,[])
rev(a:x,r) = rev(x,a:r)
rev( [],r) = r
```

```
dbl(Z) = Z
dbl(S(x)) = S(S(dbl(x))
```

```
snoc( [],b) = b:[]
snoc(a:x,b) = a:snoc(x,b)
```

Question

- What is the "some"? Concretely, ...
  - Which ones are easy to invert, and which ones are difficult?
  - How efficient inverses can be derived for the "some" programs?

```
reverse(x) = rev(x,[])
rev(a:x,r) = rev(x,a:r)
rev( [],r) = r
```

```
dbl(Z) = Z
dbl(S(x)) = S(S(dbl(x))
```

```
snoc( [],b) = b:[]
snoc(a:x,b) = a:snoc(x,b)
```

Question

- What is the "some"? Concretely, ...
  - Which ones are easy to invert, and which ones are difficult?
  - How efficient inverses can be derived for the "some" programs?

a bit Hard

```
reverse(x) = rev(x,[])
rev(a:x,r) = rev(x,a:r)
rev( [],r) = r
```

Very Easy

```
dbl(Z) = Z
dbl(S(x)) = S(S(dbl(x))
```

Easy

```
snoc( [],b) = b:[]
snoc(a:x,b) = a:snoc(x,b)
```

Question

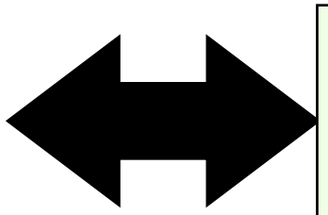
- What is the "some"? Concretely, ...
  - Which ones are easy to invert, and which ones are difficult?
  - How efficient inverses can be derived for the "some" programs?

Can we give a formal answer to the question?

Our Approach

- Grammar-based Inversion
  - … approximates program evaluation by grammar's production.
  - … characterizes program's invertivility by grammar's unambiguity.
  - … is based on bidirectional conversion of evaluation tree/production tree.

```
snoc(a:y,b) = a:snoc(y,b)
snoc( [],b) = b:[]
```



```
snoc → ■:[]
snoc → ■:snoc
```

Our Answer to the Question

- What is the "some"? Concretely, ...
  - Which ones are easy to invert, and which ones are difficult?

⬅ By how complex grammar is used

- How efficient inverses can be derived for the "some" programs?

⬅ By the computational complexity of parsing with the grammar

Classification Result

Every Invertible function

By Context-free Tree Grammar

`reverse`, …

By Regular Tree Grammar

`snoc`, …

(top-down deterministic)

`dbl`, …

Outline

- Idea of Grammar-based Inversion
- Experiments
- Related Work
- Conclusion

Target Programs

- First-Order Functional Programs
  - e.g.: `snoc` appends an element to the last of the input list.

```
snoc( [],b) = b:[]
snoc(a:x,b) = a:snoc(x,b)
```

- example of evaluation

```
snoc(3:[],4)
→ 3:snoc([],4)
→ 3:(4:[])
```

$$\begin{pmatrix} 3:4:[] \\ \| \\ [3,4] \end{pmatrix}$$

Idea of Inversion

```
snoc([] ,b) = b:[]
snoc(a:x,b) = a:snoc(x,b)
```

Evaluation

```
snoc(3:[],4)
→ 3:snoc([],4)
→ 3:(4:[])
```

Grammar

```
snoc
  → ■:[]
snoc
  → ■:snoc
```

3:4:[]

$snoc^{-1}$

Idea of Inversion

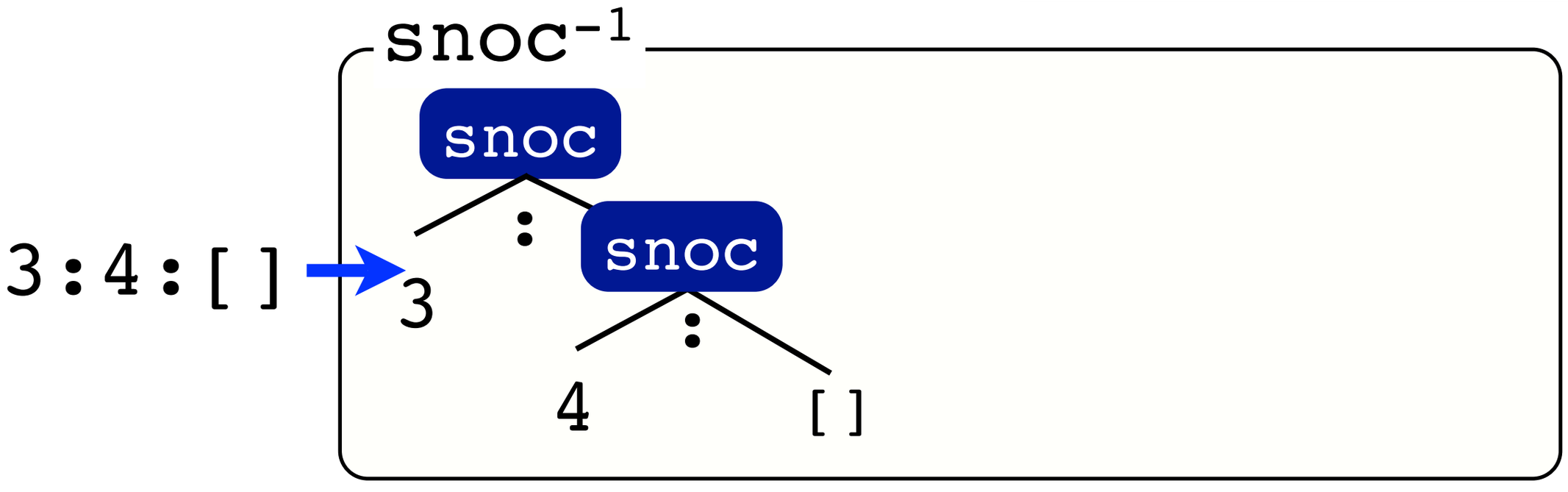
```
snoc([] ,b) = b:[]
snoc(a:x,b) = a:snoc(x,b)
```

Evaluation

```
snoc(3:[],4)
→ 3:snoc([],4)
→ 3:(4:[])
```

Grammar

```
snoc
  → ■:[]
snoc
  → ■:snoc
```



Idea of Inversion

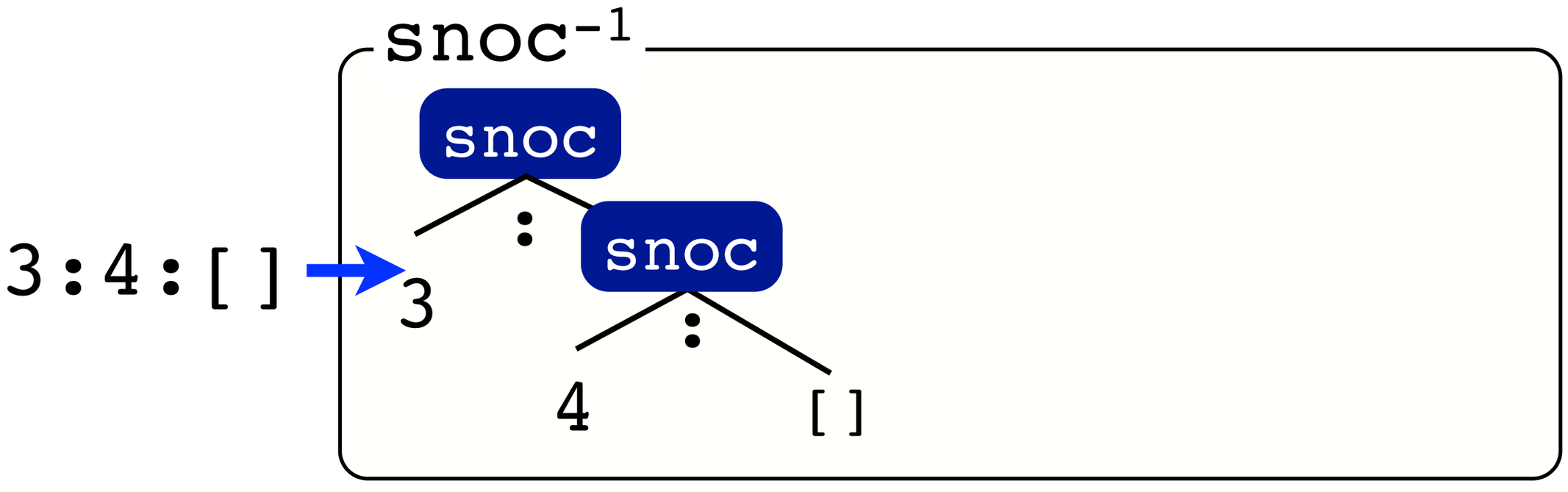
```
snoc([] ,b) = b:[]
snoc(a:x,b) = a:snoc(x,b)
```

Evaluation

```
snoc(3:[],4)
→ 3:snoc([],4)
→ 3:(4:[])
```

Grammar

```
snoc( [],b)
  → b:[]
snoc(a:x,b)
  → a:snoc(x,b)
```



Idea of Inversion

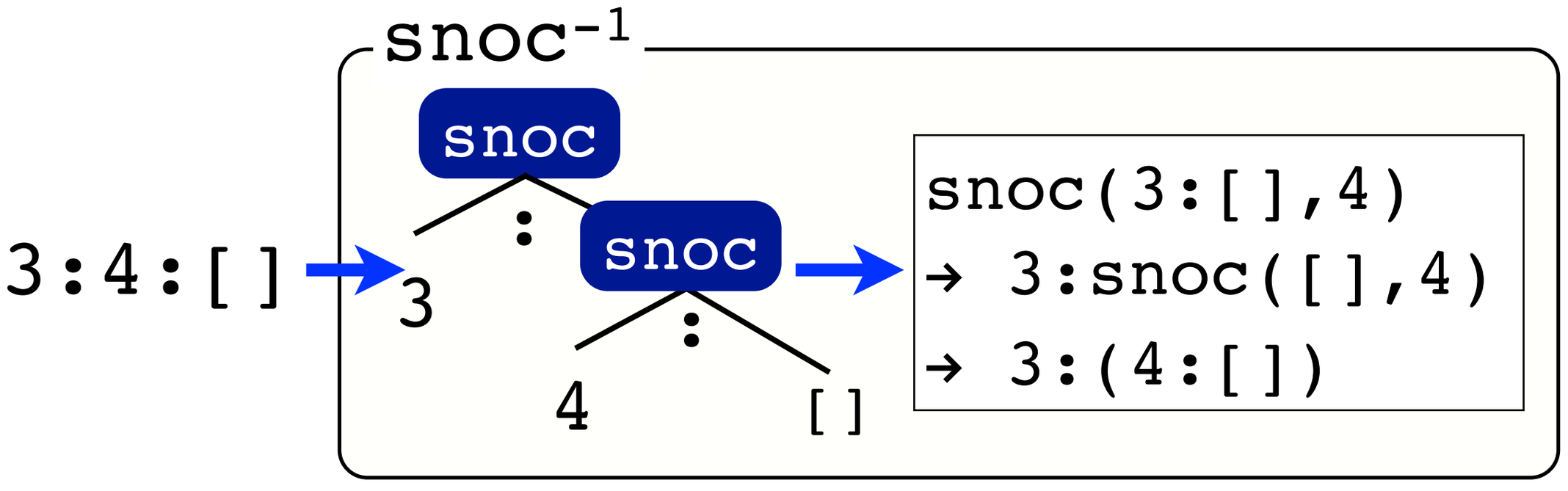
```
snoc([] ,b) = b:[]
snoc(a:x,b) = a:snoc(x,b)
```

Evaluation

```
snoc(3:[],4)
→ 3:snoc([],4)
→ 3:(4:[])
```

Grammar

```
snoc( [],b)
  → b:[]
snoc(a:x,b)
  → a:snoc(x,b)
```



Idea of Inversion

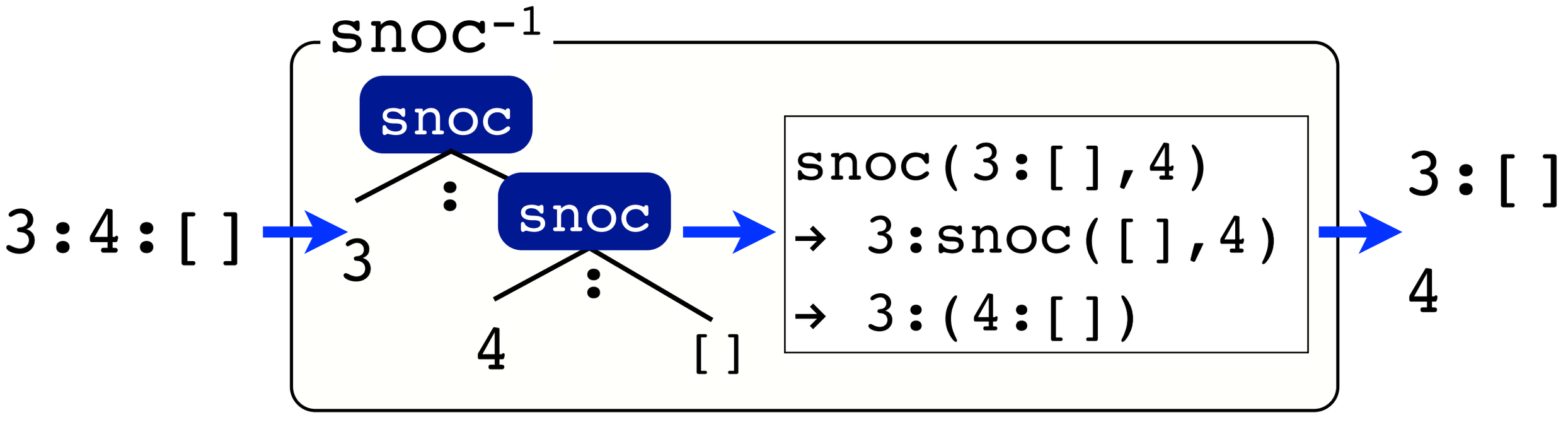
```
snoc([] ,b) = b:[]
snoc(a:x,b) = a:snoc(x,b)
```

Evaluation

```
snoc(3:[],4)
→ 3:snoc([],4)
→ 3:(4:[])
```

Grammar

```
snoc( [],b)
  → b:[]
snoc(a:x,b)
  → a:snoc(x,b)
```



Whole Inversion System

Input Program

```
snoc(x,b) = …
```

Grammar

```
snoc → ■ :snoc …
```

generate!

Invertible? — No → Fail

Yes → generate!

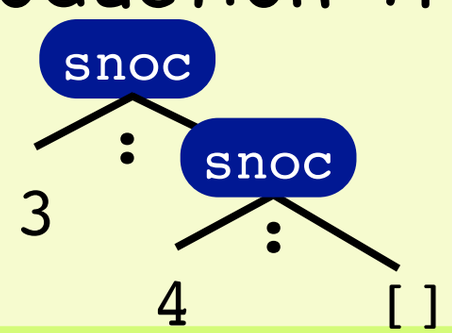
Output Inverse

```
snoc⁻¹(r)=…
```

input of inverse

```
3 : 4 : [ ]
```

Production Tree



reconstruct

Evaluation Path

```
snoc(3:[ ],4)
→ 3:snoc([ ],4)
→ 3:(4:[ ])
```

output of inverse

```
(3:[ ],4)
```

```
snoc → ■ :snoc …
```

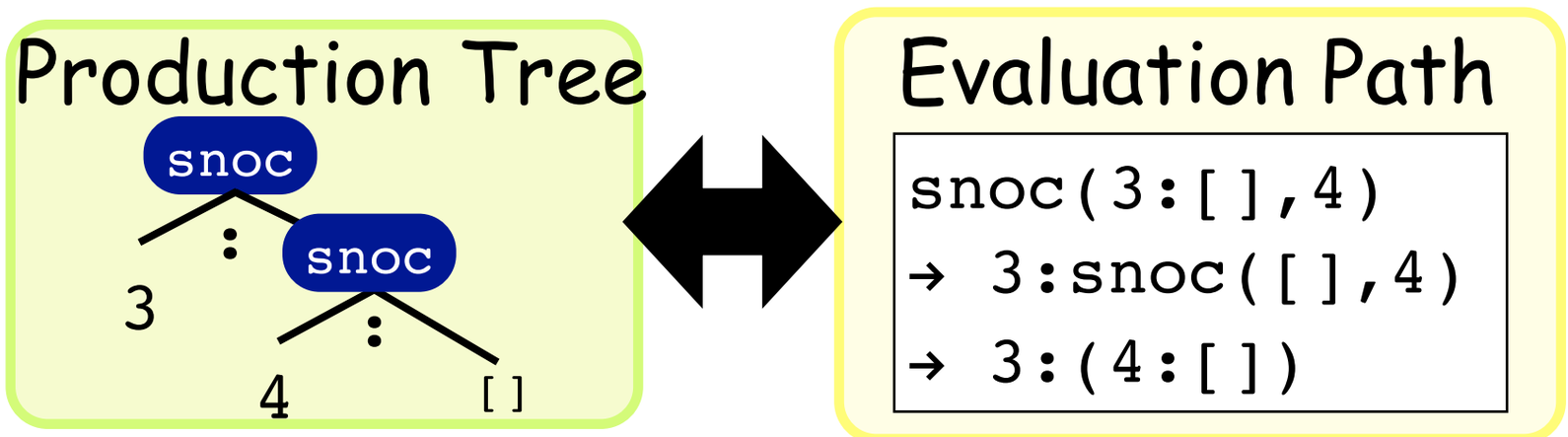
Correctness

Theorem.

If a program is non-erasing and the derived grammar is unambiguous, then the derived inverse is actually inverse.

Proof.

Bijection between production&evaluation



Extensibility

- Function compositions

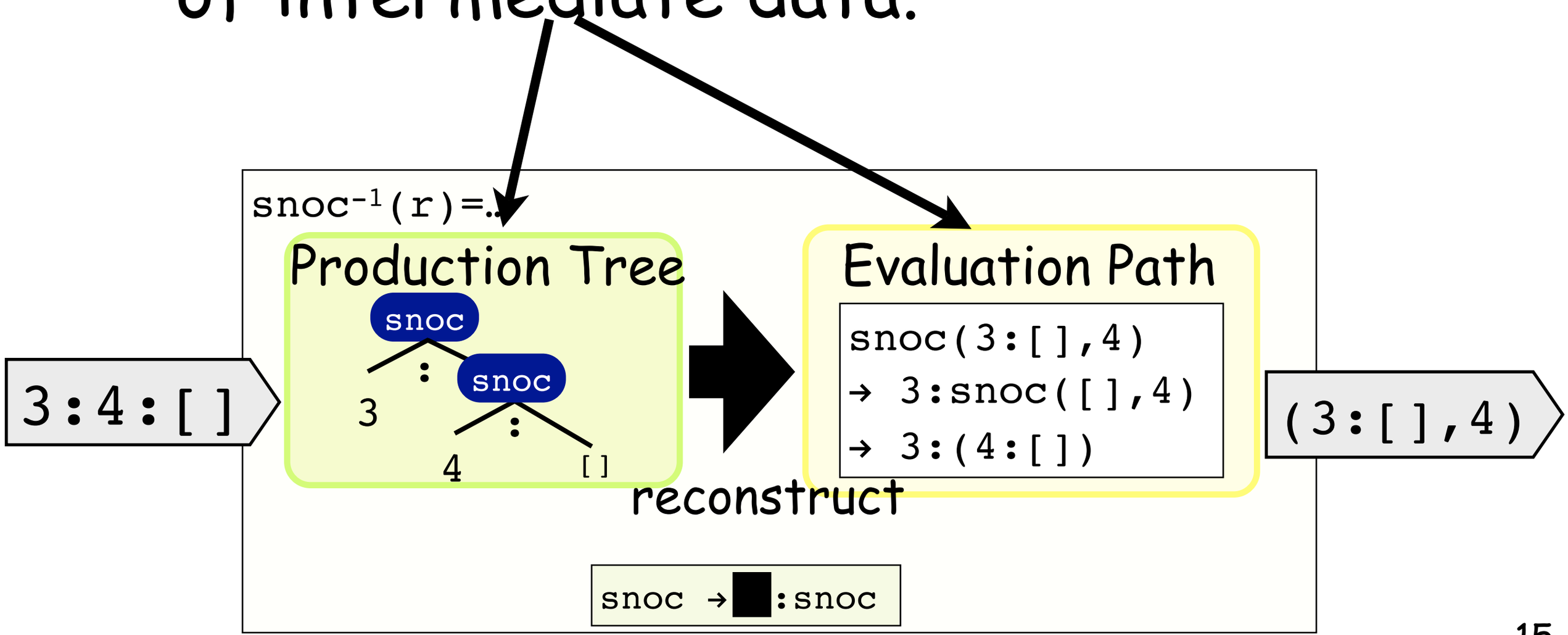
```
snoc2(x,b,c) = snoc(snoc(x,b),c)
```

  - Reparse the result of outer "snoc".

- Grammars
  - Regular tree grammars
    - `snoc`, `dbl`, …
  - IO Context-free tree grammar
    [Engelfriet&Schmidt: J.Compt.Syst.Sci77]
    - `reverse`, …

Is it efficient?

- A derived inverse seems to entail a lot of overhead due to large amount of intermediate data.



Experiments

prototype impl.: http://www.ipl.t.u-tokyo.ac.jp/~kztk/PaI/

- Comparison to handwritten inverses.
  - to know overhead caused by "parsing".

Program	#input	handwritten	derived	speed ratio
snoc	8M	0.67s	0.95s	1.4
dbl	10M	0.11s	0.23s	2.1
zip	8M	0.28s	0.70s	2.5
runlength	9M	0.33s	0.76s	2.3

Acceptable overhead!

Related Work

- [Glück&Kawabe: FLOPS04]
  - LR-parsing techniques to determines inverse programs
- [Yellin: ICSE88]
  - Program inversion of some class of *AG*

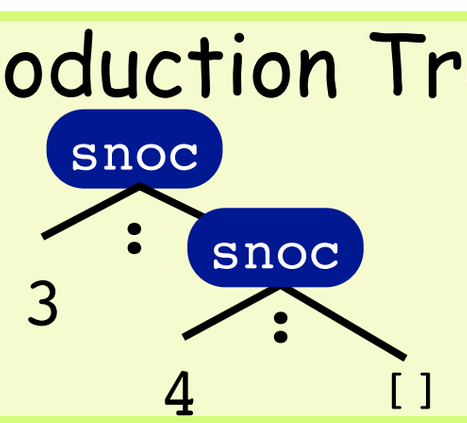
- Our work is a *generalization* and reformulation of these works.

Conclusion

prototype impl.: http://www.ipl.t.u-tokyo.ac.jp/~kztk/PaI/

- Grammar-based Inversion
  - … is based on bijective mapping between production and evaluation.

Production Tree



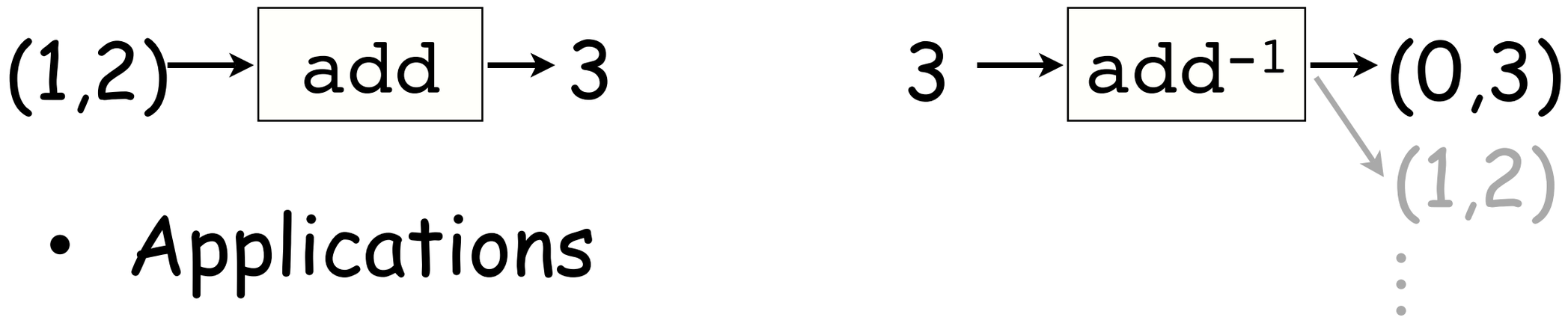
Evaluation Path

```
snoc(3:[],4)
→ 3:snoc([],4)
→ 3:(4:[])
```

  - … classifies programs by unambiguous grammars requires in inversion.
    - unambiguity (+α) implies invertivility!

Future Work

- More studies on right-inverse.
  - returns one/all of possible input.



  - Applications
    - bidirectional transformations
      [Foster et al.: POPL05, Hu et al.: PEPM04, ...]
    - testing with predicate ∀x.P(x) -> Q(x)
      [Runciman et al.: Haskell08, Fischer: D.Thesis, ...]
    - and many more.