Multi-Return Macro Tree Transducers

Kazuhiro Inaba
Haruo Hosoya

University of Tokyo

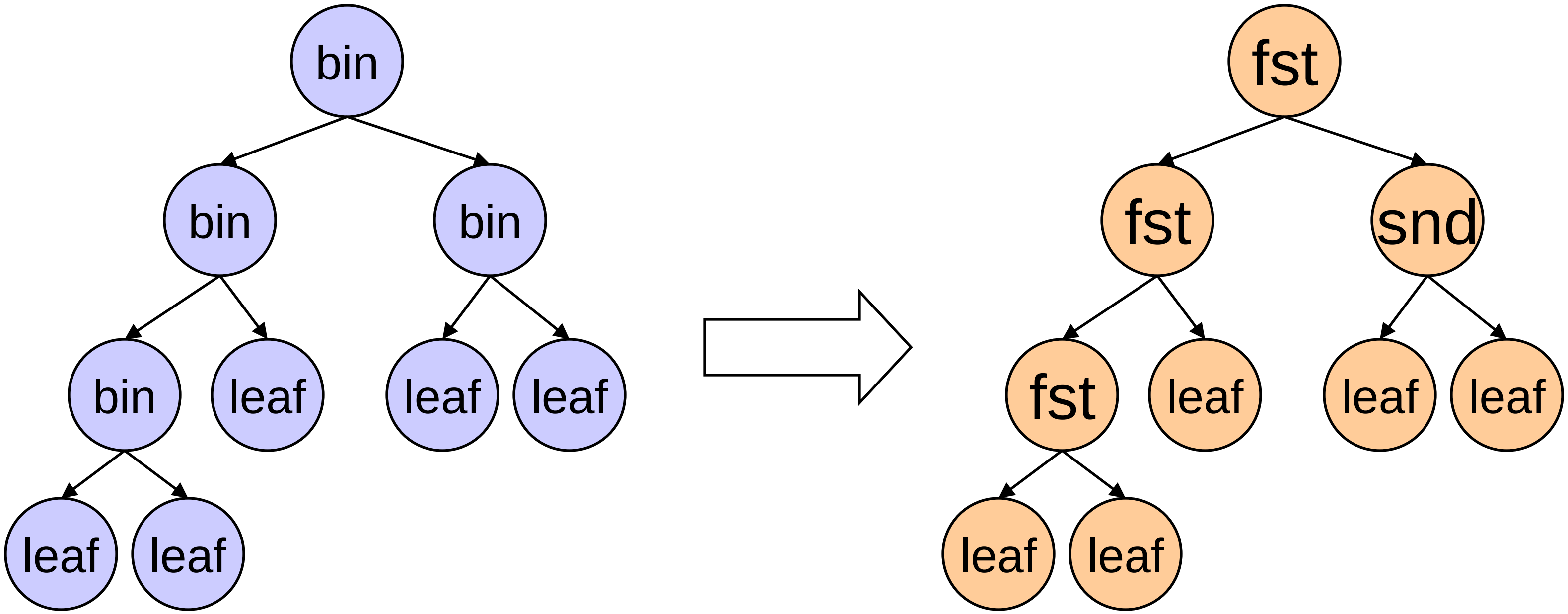
PLAN-X 2008, San Francisco

Models of Tree Translation

- (Top-down) Tree Transducer (TOP) [Rounds/Thatcher, 70's]
  - Finite set of relations from a tree to a tree
  - Defined by structural (mutual) recursion on the input tree

```
<q, bin(x1,x2)> → fst( <q,x1>, <p,x2> )
<q, leaf()>     → leaf()

<p, bin(x1,x2)> → snd( <q,x1>, <p,x2> )
<p, leaf()>     → leaf()
```




```
<q, bin(x1,x2)> → fst( <q,x1>, <p,x2> )
<q, leaf()>     → leaf()

<p, bin(x1,x2)> → snd( <q,x1>, <p,x2> )
<p, leaf()>     → leaf()
```

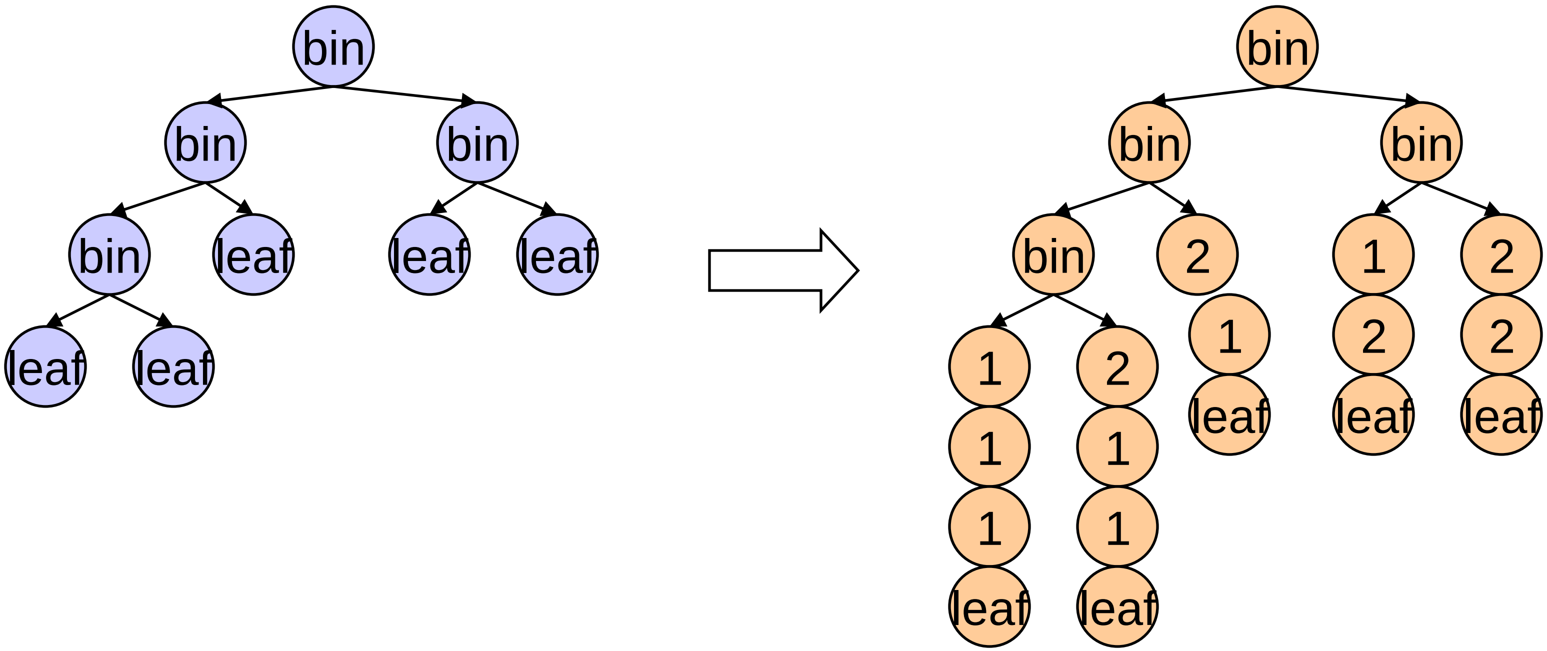
Models of Tree Translation

- Macro Tree Transducer (MTT)

  [Engelfriet/Vogler 85]

  - Tree Transducer + Accumulating parameters
  - Strictly more expressive than TOP

```
<q, bin(x1,x2)>(y) → bin( <q,x1>(1(y)),
                            <q,x2>(2(y)) )
<q, leaf()>    (y) → y
```




```
<q, bin(x1,x2)>(y) → bin( <q,x1>(1(y)),
                          <q,x2>(2(y)) )
<q, leaf()>    (y) → y
```

- Multi-Return Macro Tree Transducer [Our Work]
  - Macro Tree Transducer + Multiple return values

```
<q, bin(x1,x2)>(y) → let (z1,z2) = <q,x1>(1(y)) in
                      let (z3,z4) = <p,x2>(2(y)) in
                        (bin(z1,z3), fst(z2,z4))
<q, leaf()>    (y) → (leaf(), y)

<p, bin(x1,x2)>(y) → let (z1,z2) = <q,x1>(1(y)) in
                      let (z3,z4) = <p,x2>(2(y)) in
                        (bin(z1,z3), snd(z2,z4))
<p, leaf()>    (y) → (leaf(), y)
```

Outline

- Why Multi-Return?
- Definition of Multi-Return MTT
- Expressiveness of Multi-Return MTT
  - Deterministic case
  - Nondeterministic case

Why Multi-Return?

Why Multi-Return?

- MTT is not symmetric
  - can pass multiple tree-fragments from a parent to the children via accumulation parameters

```
<q0, a(x)>         → <q1,x>( some(tree,here),
                             other(tree,here) )

<q1, b(x)>(y1,y2) → use( y1, and(y2), here )
```

Why Multi-Return?

- MTT is not symmetric
  - can not pass multiple tree-fragment from a child to the parent

```
<q0, a(x)> → can( use(<q1,x>), here )

<q1, b(x)> → one(tree)
```

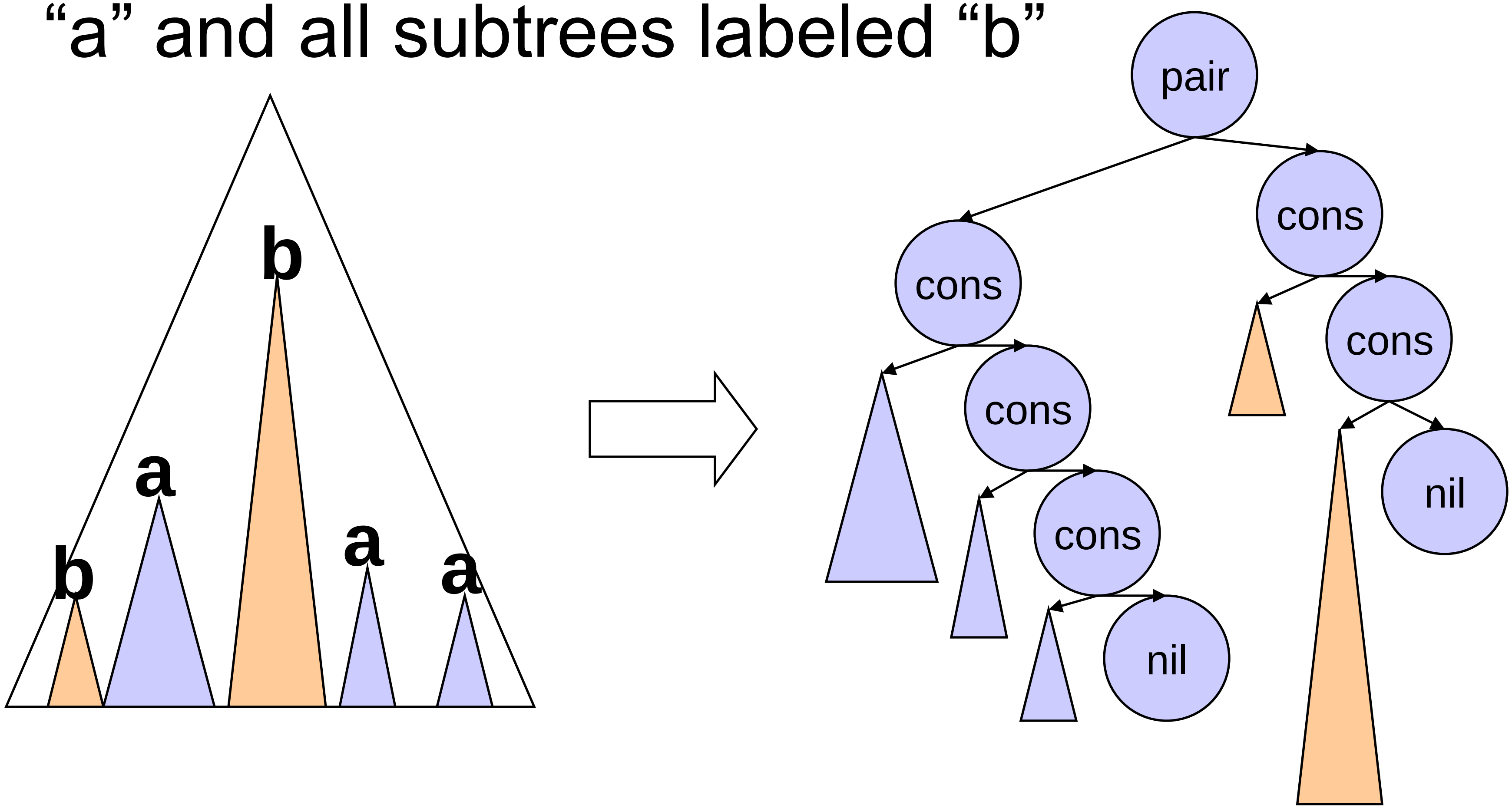
  - Multi-Return MTT can:

```
<q0, a(x)> → let (z1,z2) = <q1,x> in
                can( use(z1), and(z2), here )

<q1, b(x)> → (one(tree), two(tree))
```

Inefficiency caused by the lack of child-to-parent multiple tree passing

- Gather all subtrees with root node labeled "a" and all subtrees labeled "b"



- Normal MTT realizing this translation must traverse the input tree twice
  - For gathering “a” and gathering “b”
  - No way to pass two intermediate lists from child to parent!

```
<q0, root(x)> → pair( <get_a,x>(nil()),
                      <get_b,x>(nil()) )

<get_a, a(x)>(y) → cons( a(x), <get_a,x>(y) )
<get_a, b(x)>(y) → <get_a, x>(y)
…
<get_b, a(x)>(y) → <get_b, x>(y)
<get_b, b(x)>(y) → cons( b(x), <get_b,x>(y) )
```

- Multi-Return MTT realizing this translation must traverse the input tree twice

```
<q0, root(x)>       → let (z1,z2) = <get,x>(nil(),nil()) in
                          pair(z1, z2)
<get, a(x)>(ya,yb) → let (z1,z2) = <get,x>(ya,yb) in
                          (cons(a(x),ya), yb)
<get, b(x)>(ya,yb) → let (z1,z2) = <get,x>(ya,yb) in
                          (ya, cons(b(x),yb))
```

Definition of
(Multi-Return) MTT

Macro Tree Transducer (MTT)

- A MTT is a tuple consisting of
  - Q : Set of states
  - q0 : Initial state
  - Σ : Set of input alphabet
  - Δ : Set of output alphabet
  - R : Set of rules of the following form:

```
<q, σ(x1,…,xk)>(y1, …, ym) → rhs

rhs ::= δ(rhs, …, rhs)
      | <q, xi>( rhs, …, rhs )
      | yi
```

Macro Tree Transducer (MTT)

- A MTT is defined to be
  - Deterministic if for every pair of $q \in Q$, $\sigma \in \Sigma$, there exists at most one rule of the form $\langle q, \sigma(\ldots) \rangle(\ldots) \rightarrow \ldots$
  - Nondeterministic otherwise
- Call-by-Value (Inside-Out) Evaluation
  - Arguments are evaluated first, before function calls

```
<q1, a(x)>() → <q2,x>( <q3,x>() )
<q2, a(x)>(y)→ b(y, y)
<q3, a(x)>() → c()
<q3, a(x)>() → d()
```

```
<q1, a(a(c()))> ⇒
 b(c(),c()) or b(d(),d())
```

Multi-Return Macro Tree Transducer (mr-MTT)

- A mr-MTT is a tuple consisting of
  - Q : Set of states
  - q0 : Initial state
  - Σ : Set of input alphabet
  - Δ : Set of output alphabet
  - R : Set of rules of the following form:

```
<q, σ(x1,…,xk)>(y1, …, ym) → rhs

rhs ::= (let (z1,..zn) = <q,xi>(t,…,t) in)* (t,…,t)
  t ::= δ(t,…,t) | yi | zi
```

Multi-Return Macro Tree Transducer (mr-MTT)

- A mr-MTT is defined to be
  - Deterministic if for every pair of $q \in Q$, $\sigma \in \Sigma$, there exists at most one rule of the form $\langle q, \sigma(\ldots) \rangle (\ldots) \rightarrow \ldots$
  - Nondeterministic otherwise
- Call-by-Value (Inside-Out) Evaluation
  - Arguments are evaluated first, before function calls

Expressiveness

Question

- Are multi-return MTTs more expressive than single-return MTTs?

  (Is there any translation that can be written in mr-MTT but not in MTT?)

Answer

- Deterministic mr-MTTs are equal in expressiveness to normal MTTs
    - In other words, every deterministic mr-MTT can be simulated by a normal MTT
- Nondeterministic mr-MTTs are strictly more expressive than normal MTTs

Proof Sketch (Deterministic Case)

- A state returning n-tuples of trees can be split into n states returning a single tree

```
<q,…>(…)→let (z1,z2) = <q,x> in (a(z1,z2), b(z2,z1))
```

⇓

```
<q_1,…>(…) → let z1 = <q_1,x> in
              let z2 = <q_2,x> in a(z1,z2)
<q_2,…>(…) → let z1 = <q_1,x> in
              let z2 = <q_2,x> in b(z2,z1)
```

⇓

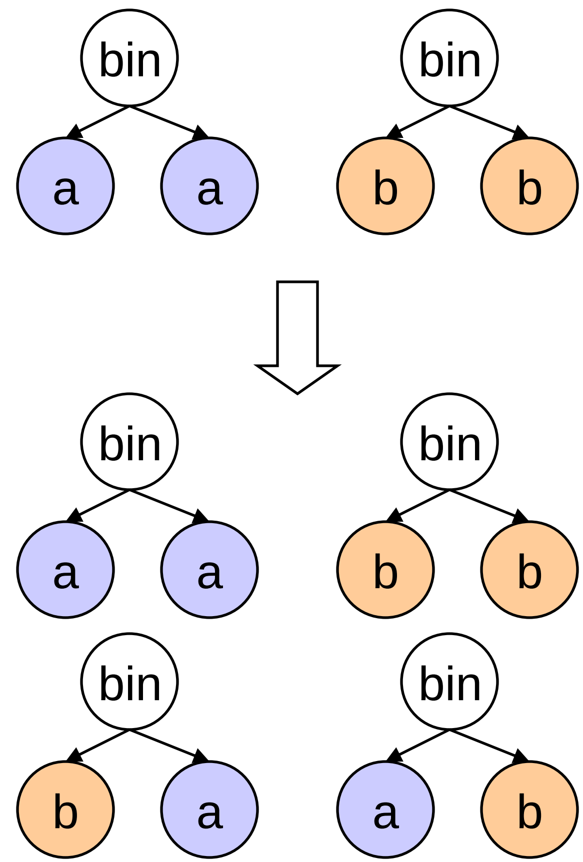
```
<q_1,…>(…) → a(<q_1,x>, <q_2,x>)
<q_2,…>(…) → b(<q_2,x>, <q_1,x>)
```

Nondeterministic case…

- State-splitting may change the behavior

```
<q0, node(x)>
  → let (z1,z2) = <q,x> in
      bin(z1,z2)
<q, leaf()> → (a(), a())
<q, leaf()> → (b(), b())
```

```
<q0, node(x)>
  → bin(<q_1,x>, <q_2,x>)
<q_1, leaf()> → a()
<q_2, leaf()> → a()
<q_1, leaf()> → b()
<q_2, leaf()> → b()
```



Nondeterministic case…

- In fact, there is no general way to simulate a nondeterministic mr-MTT in a normal MTT
- Example of such translation ⇒ “twist”

Nondeterministically translates one input string

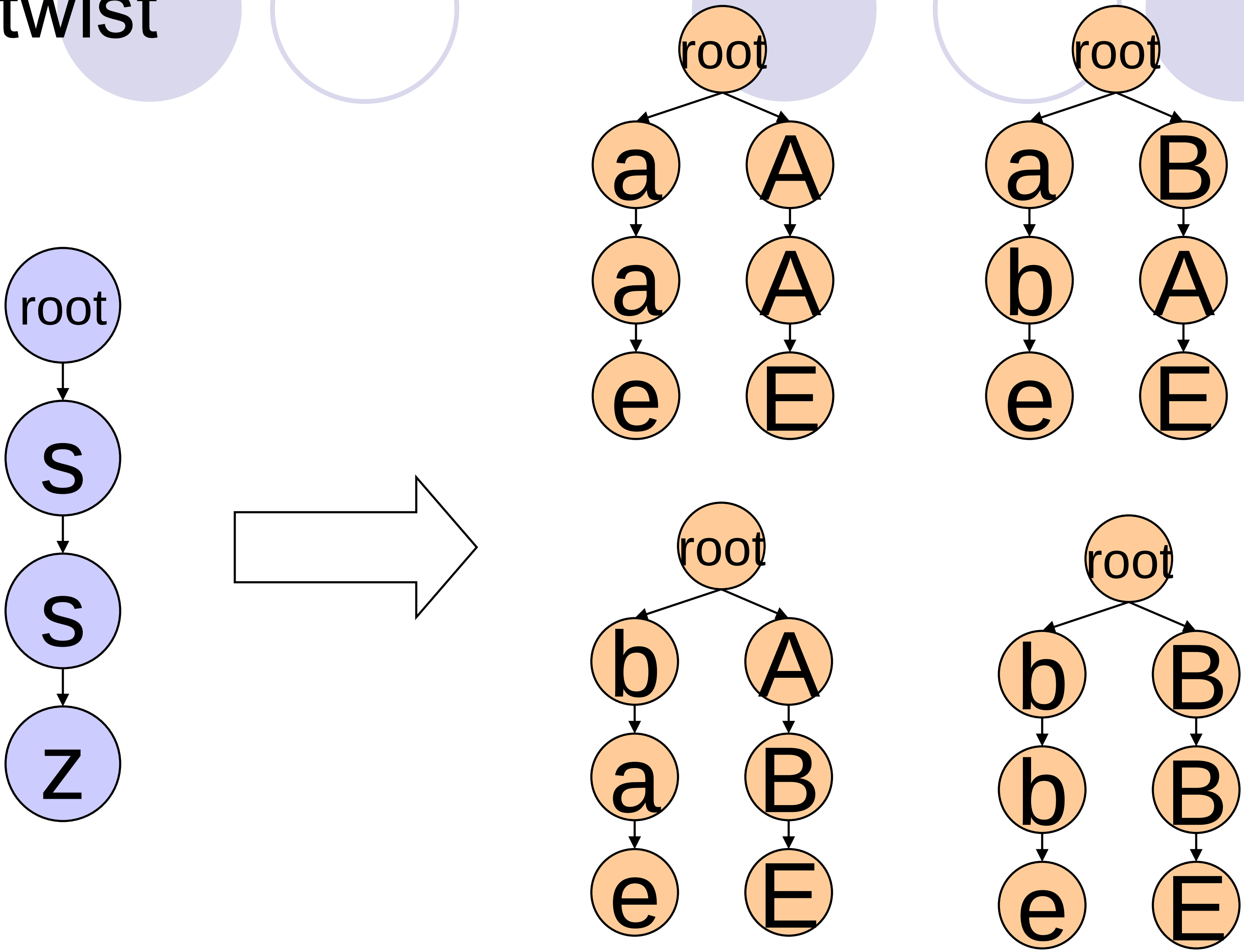
sss…ss

of length n to two string of the same length:

- one consists of symbols a and b, and
- the other consists of symbols A and B

such that the outputs are being reversal of each other.

“twist”



“twist” in Multi-Return MTT

```
<q, root(x)>→ let (z1,z2) = <p,x>( E() ) in
                    root(z1, z2)

<p, s(x)>(y)→ let (z1,z2) = <p,x>( A(y) ) in
                    (a(z1), z2)
<p, s(x)>(y)→ let (z1,z2) = <p,x>( B(y) ) in
                    (b(z1), z2)
<p, z>(y)    → (e(), y)
```

How to prove the inexpressibility in MTT?

- Known proof techniques
  - Height Property
  - Size Property
  - Output Language
  - …

- … all fails here.
- → Long and involved proof specialized for the “twist” translation

Proof Sketch (Inexpressibility of “twist”)

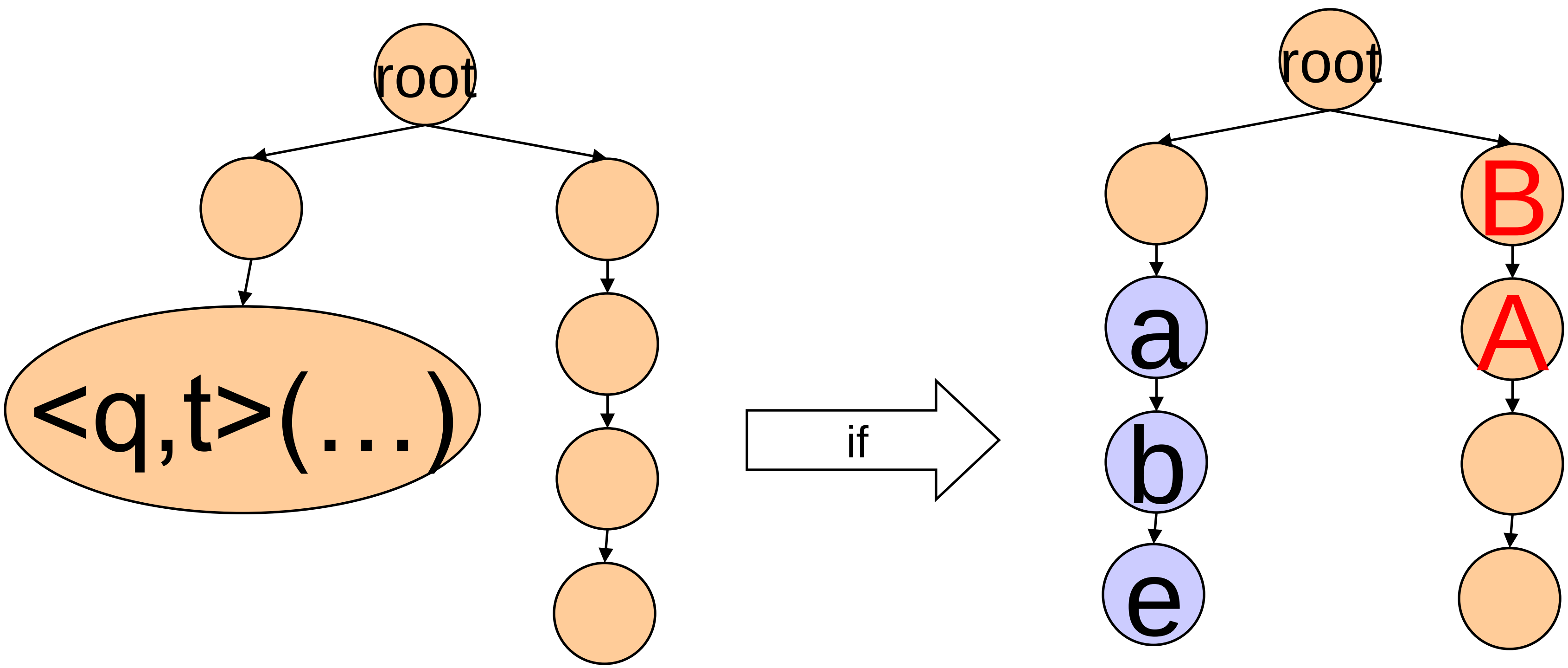
- “Reductio ad absurdum” argument
  - First, suppose a MTT realizing twist
  - Then, we show that the size of the set of output from the MTT has polynomial upper bound w.r.t. the size of the input tree
  - which is not the case for “twist”, having exponential number of outputs

Rough Proof Sketch :: Step 0/5

- Suppose a MTT M is realizing “twist”

Rough Proof Sketch :: Step 1/5

- Lemma 4
  - If a term of M is evaluated to a proper subpart of an output, it MUST be evaluated to the term



Rough Proof Sketch :: Step 2/5

- Lemma 5
  - Any term of M generating only the output of "twist" is equivalent to a term if the following form:

```
wnf ::= <q,t>(wnf, …, wnf)  (always generates "root")
      | ct
ct  ::= δ(ct, …, ct)
```

Example:

```
<q1,t1>( <q2,t2>(a(e), A(E)),
         <q3,t3>(),
         <q4,t4>(<q5,t5>(b(a(e), E)) )
```

Rough Proof Sketch :: Step 3/5

- Lemma 7
  - Any term of M in the form of preceding slide is equivalent to a set of terms in the following form ("normal form" in the paper):

```
nf ::= <q,t>(st, …, st)
st ::= a(st) | b(st) | e() | A(st) | B(st) | E()
```

Rough Proof Sketch :: Step 4/5

- Lemma 8
  - Two normal form terms with the same head produces “similar” set of outputs – the number of different output trees are constant
  - Shown by a similar argument to the first lemma

Rough Proof Sketch :: Step 5/5

- Lemma 10 / Cor 1
  - The MTT M can produce at most O(n^2) number of output trees, where n is the length of the input string
  - This is a contradiction, since
    - M is supposed to realize “twist”
    - The number of output trees from “twist” is 2^n

Conclusion

Conclusion

- Multi-return MTT
  - MTT + Multiple Return Values
- Expressiveness
  - Deterministic: same as MTT
  - Nondeterministic: more powerful than MTT

Future/Ongoing Work

- Decomposition of mr-MTT
  - Is a mr-MTT can be simulated by a composition of multiple MTTs?
- Hierarchy of mr-MTT
  - The width of returned tuples affects the expressivenss?
- Application of the proof technique to other translations know “as a folklore” not to be expressible in MTT

Thank you for listening!