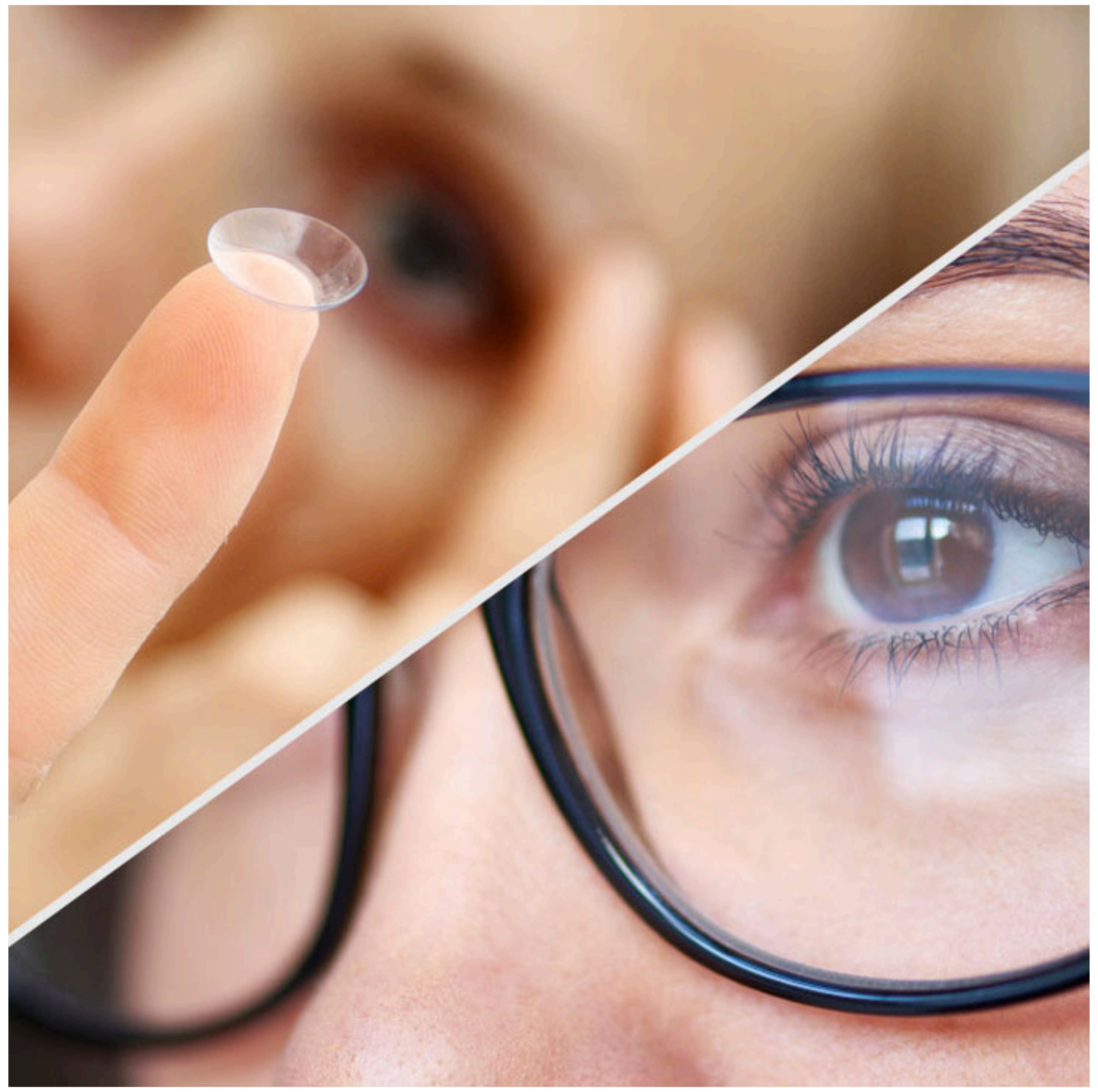
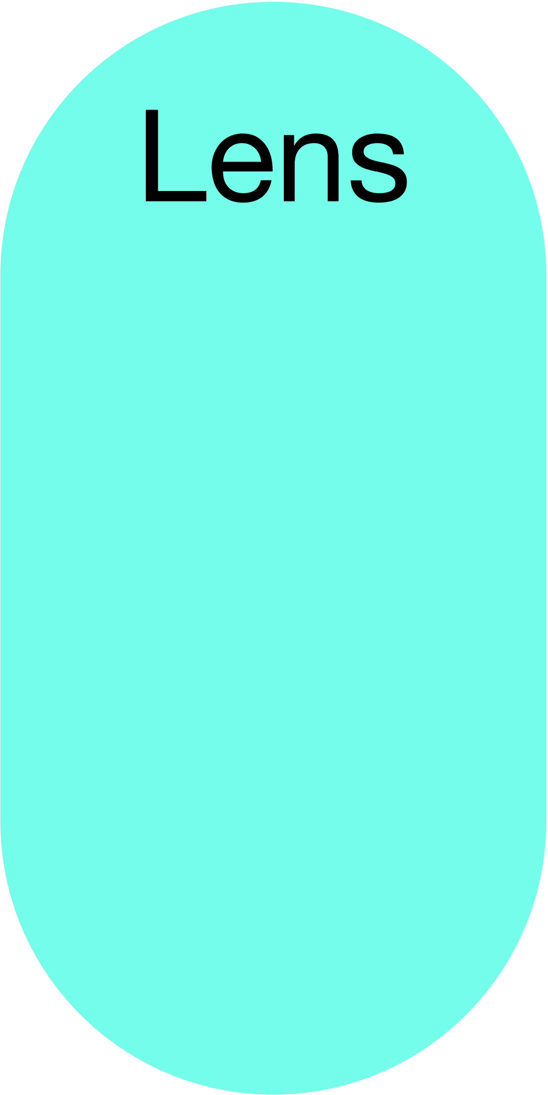
Confluence in Lens Synthesis

Anders Miltner, Kathleen Fisher, Benjamin Pierce, David Walker, Steve Zdancewic

Lenses?



Lenses are Synchronizers

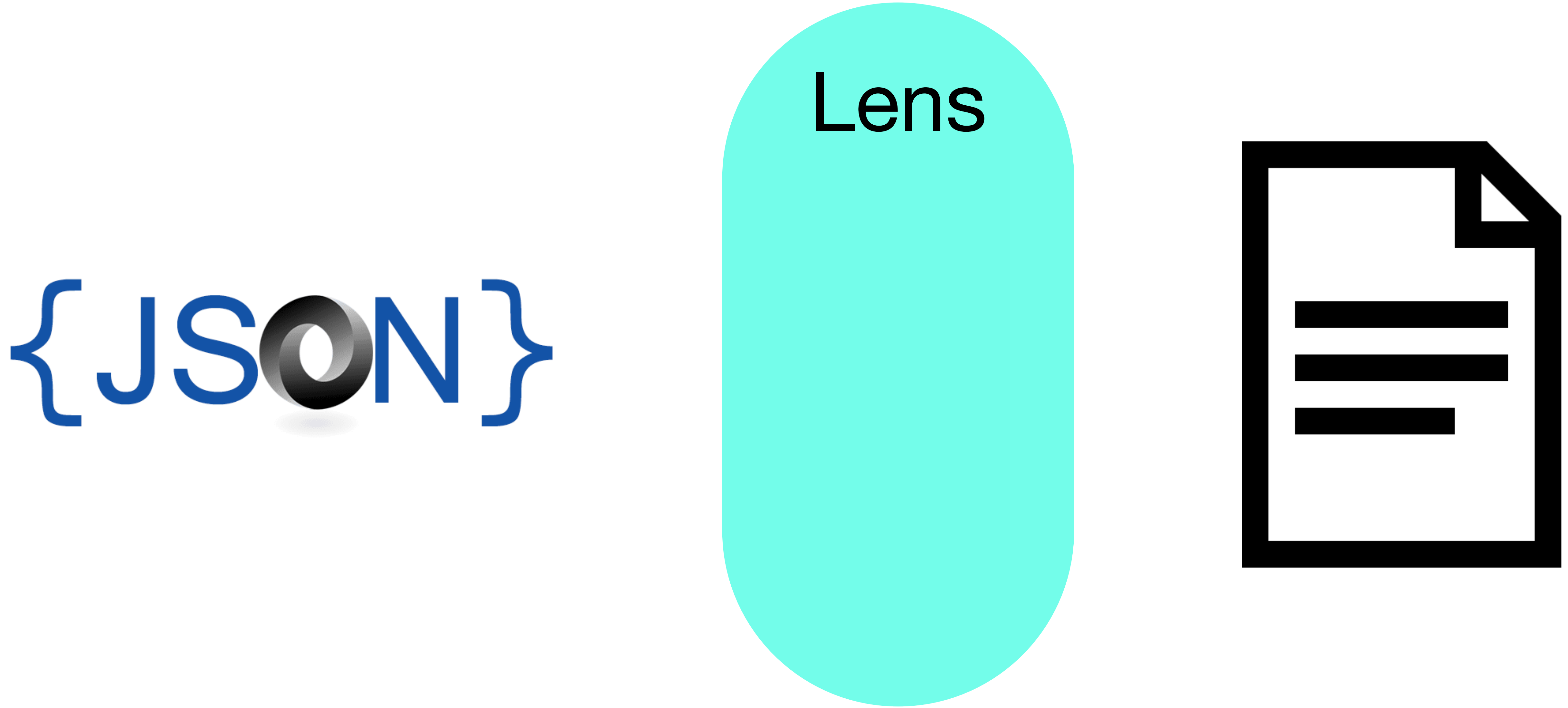


Lenses are Synchronizers

Lens

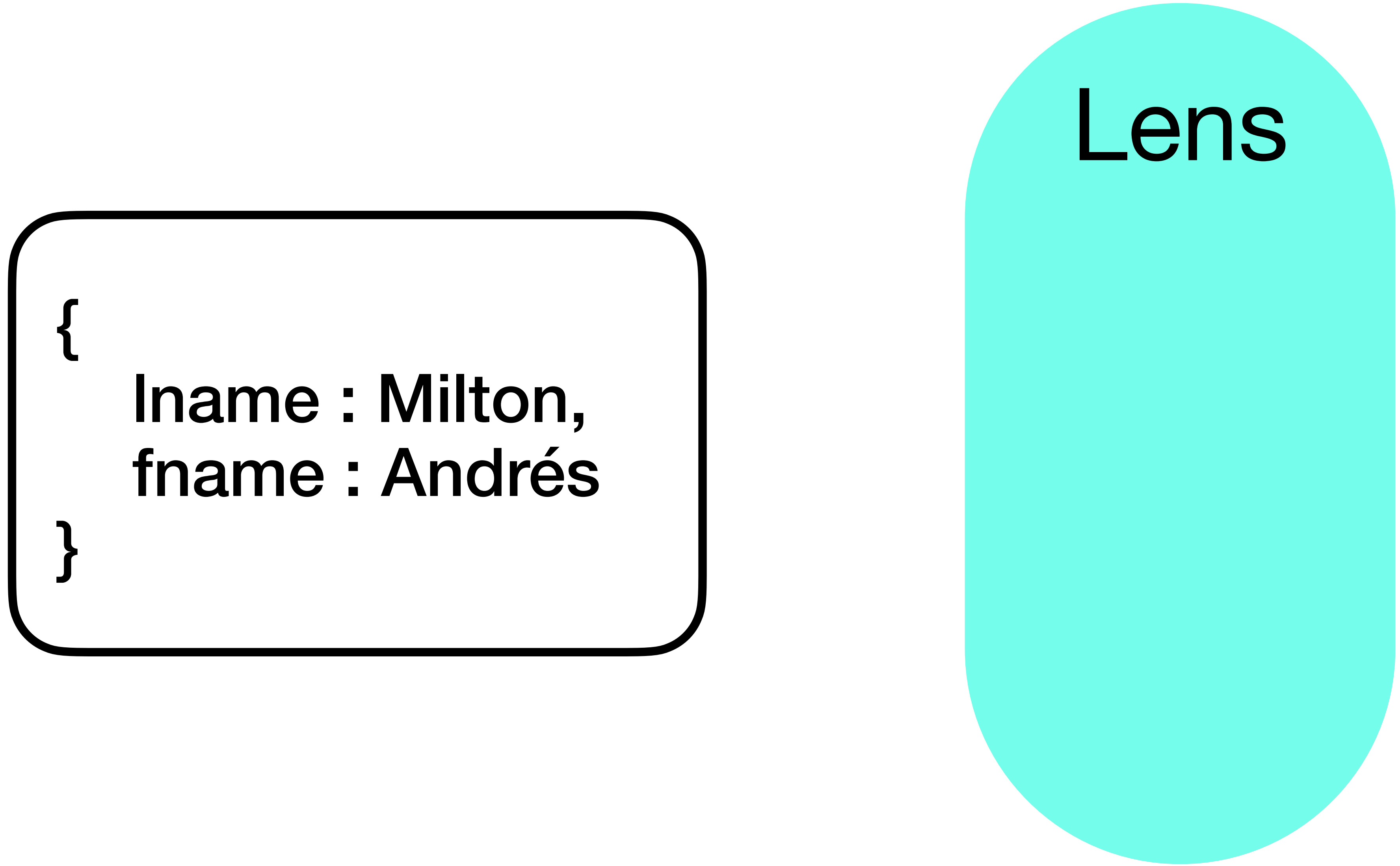
get
put
create

Lenses are Synchronizers



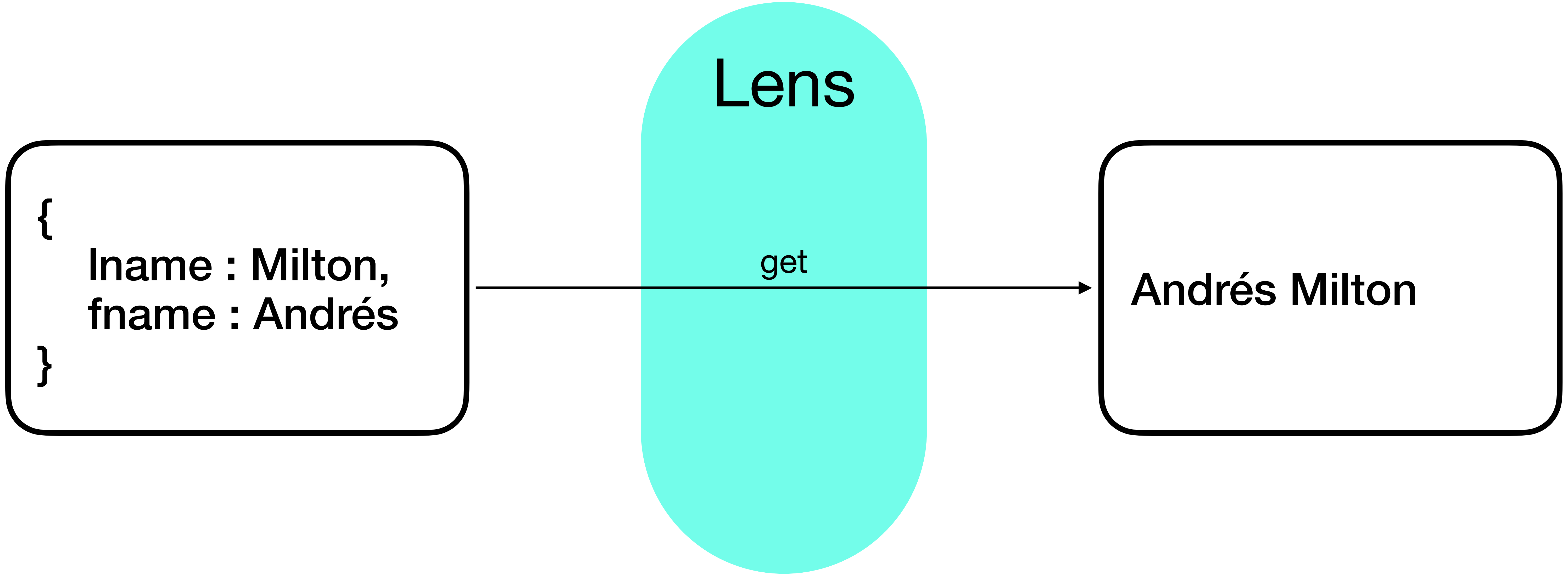
Lens

Lenses are Synchronizers



{
    lname : Milton,
    fname : Andrés
}

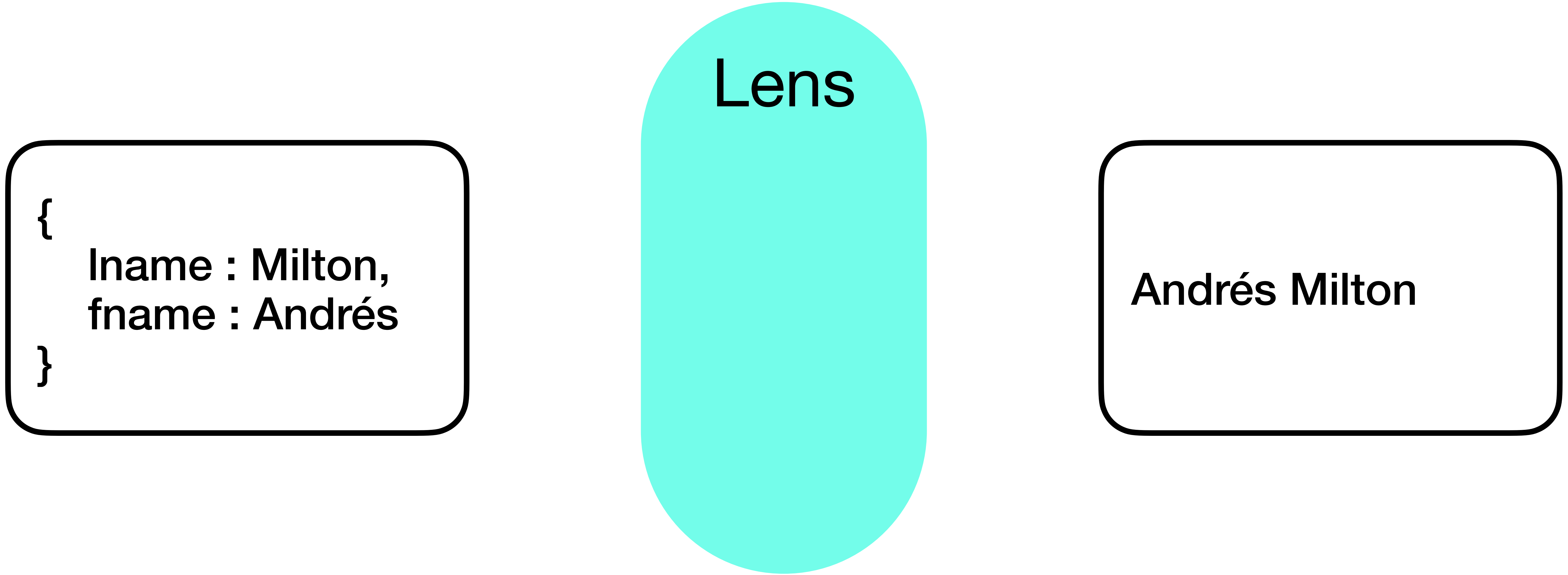
Lenses are Synchronizers



{
lname : Milton,
fname : Andrés
}

Andrés Milton

Lenses are Synchronizers

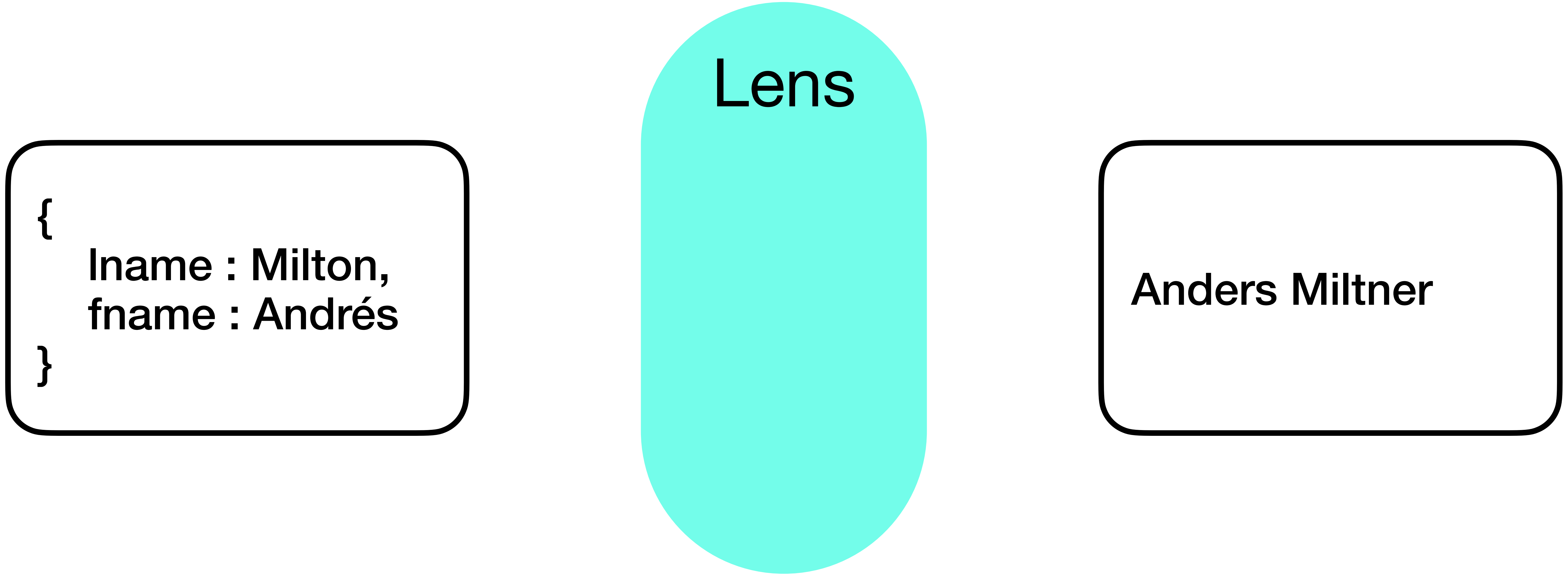


{
lname : Milton,
fname : Andrés
}

Lens

Andrés Milton

Lenses are Synchronizers

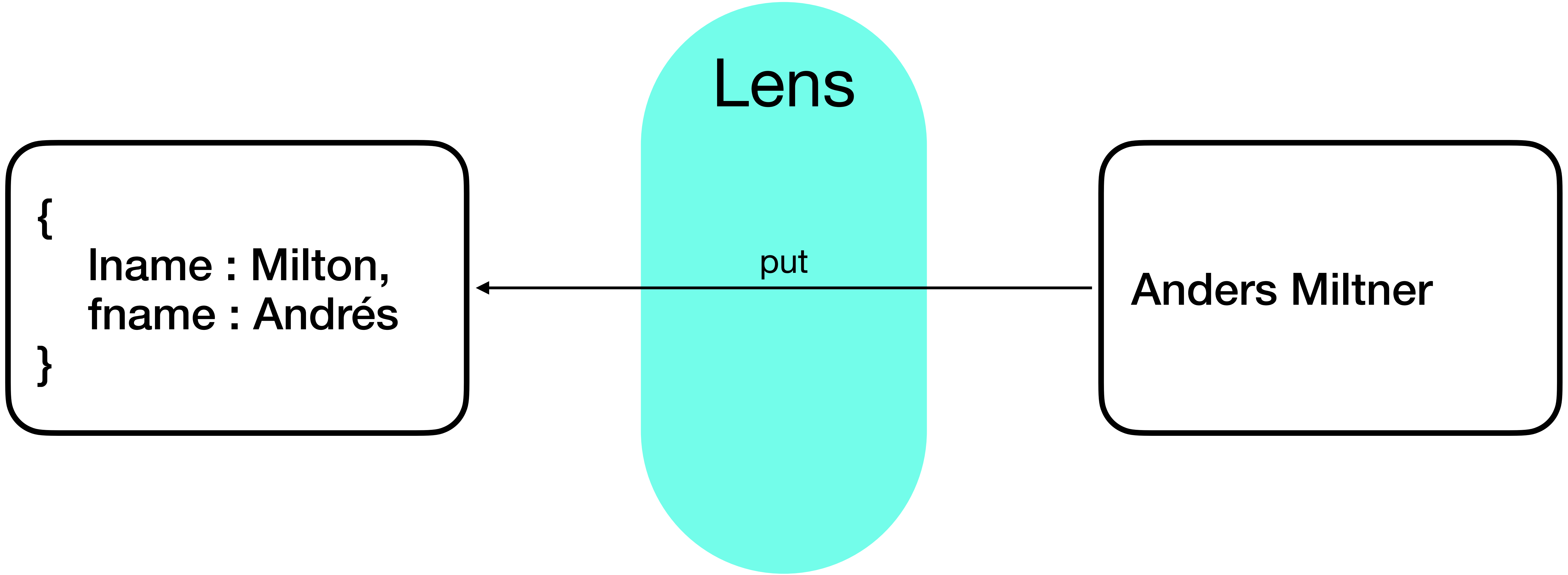


Lens

```
{
    lname : Milton,
    fname : Andrés
}
```

Anders Miltner

Lenses are Synchronizers



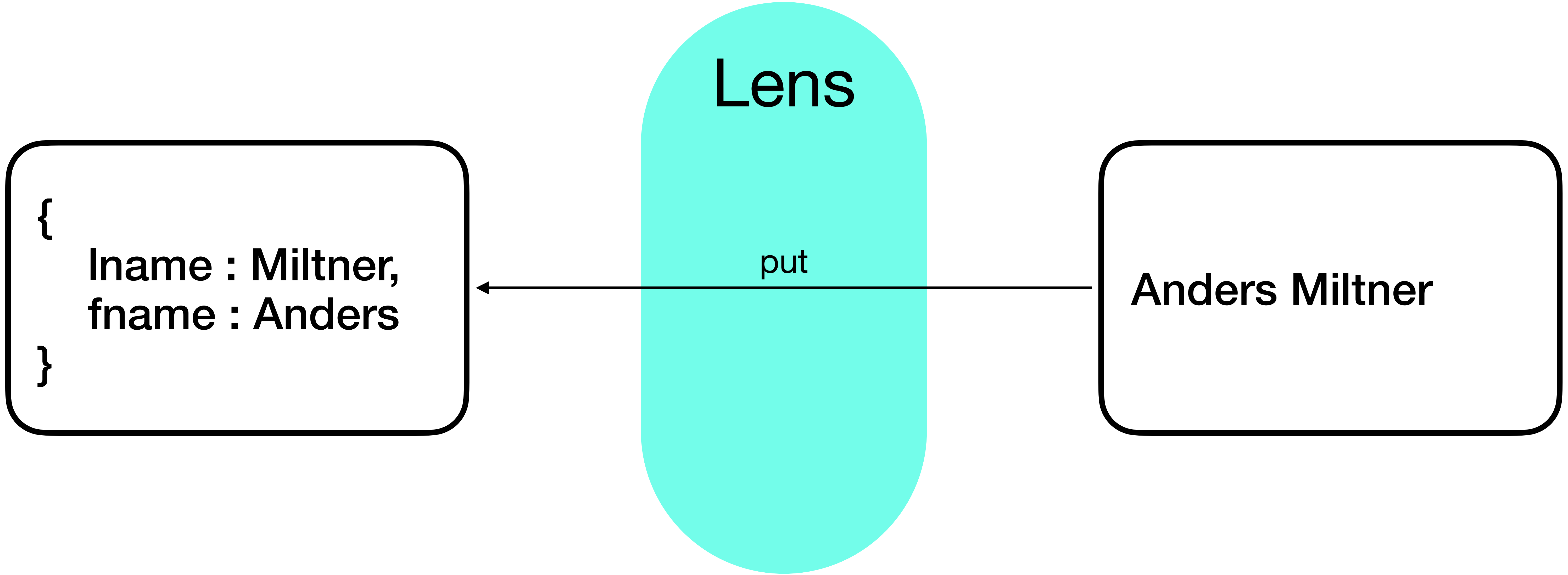
Lens

{
lname : Milton,
fname : Andrés
}

put

Anders Miltner

Lenses are Synchronizers



Lens

{
lname : Miltner,
fname : Anders
}

put

Anders Miltner

Lenses are Pervasive

Lens Variants

Bijections

Classical Lenses

Quotient Lenses

Symmetric Lenses

Edit Lenses

Simple Symmetric Lenses

Lens Domains

Strings

Relational Algebras

Parsing + Pretty Printing

Data Structures

Sets

Combinations of Above

Lenses are Pervasive

Lens Variants

Bijections

Classical Lenses

Quotient Lenses

Symmetric Lenses

Edit Lenses

Simple Symmetric Lenses

Lens Domains

Strings

Relational Algebras

Parsing + Pretty Printing

Data Structures

Sets

Combinations of Above

DSLs for writing lenses in various domains

Lenses are Pervasive

Lens Variants

Bijections

Lens Domains

Strings

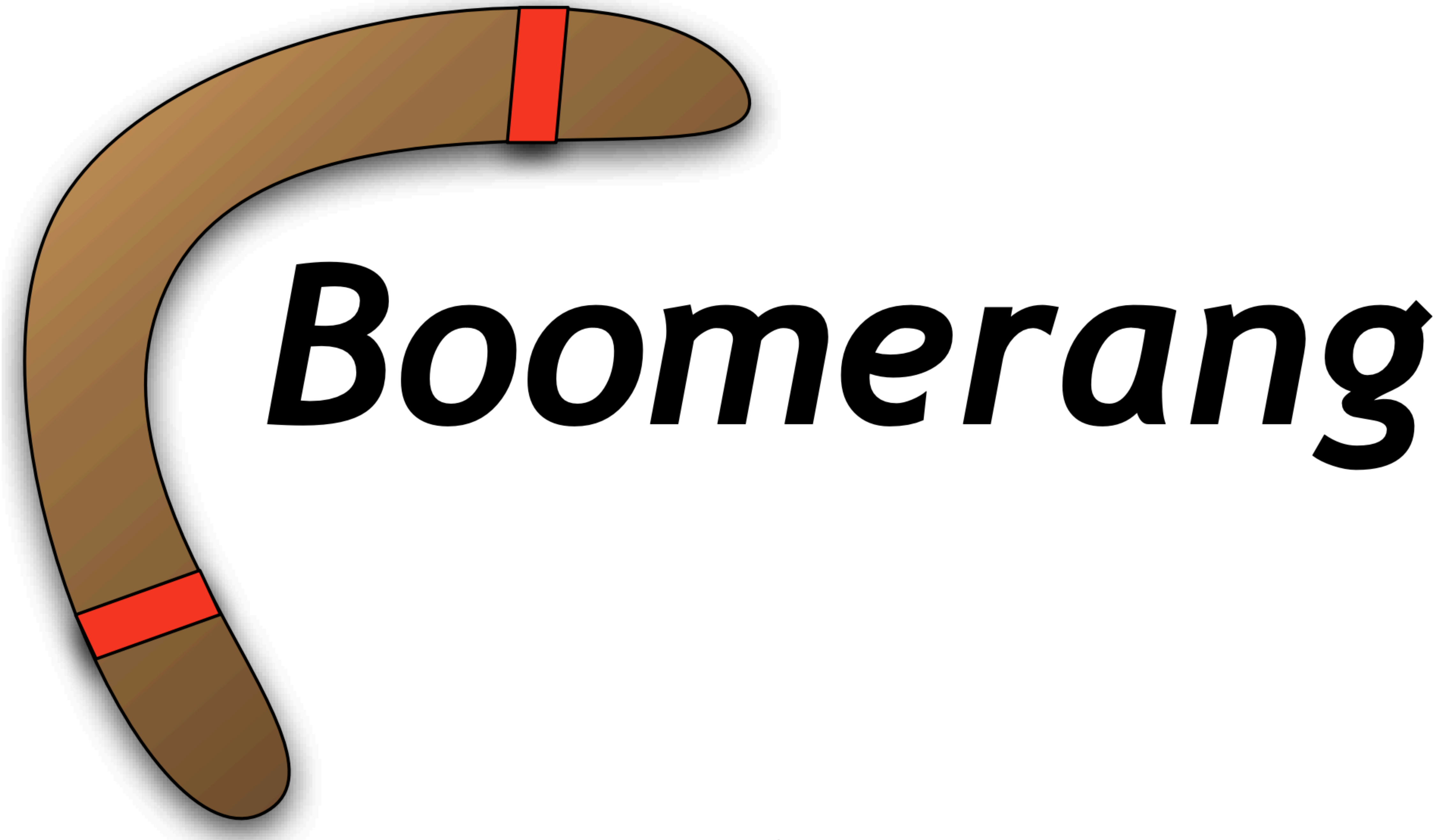
Lenses are Pervasive

Lens Variants

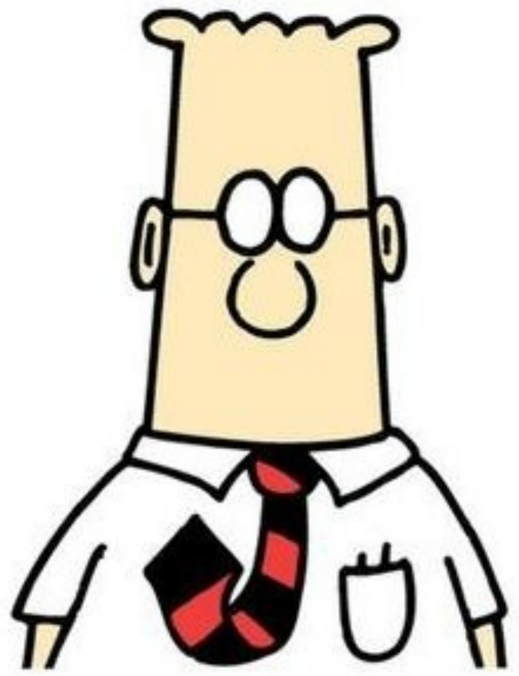
Bijections

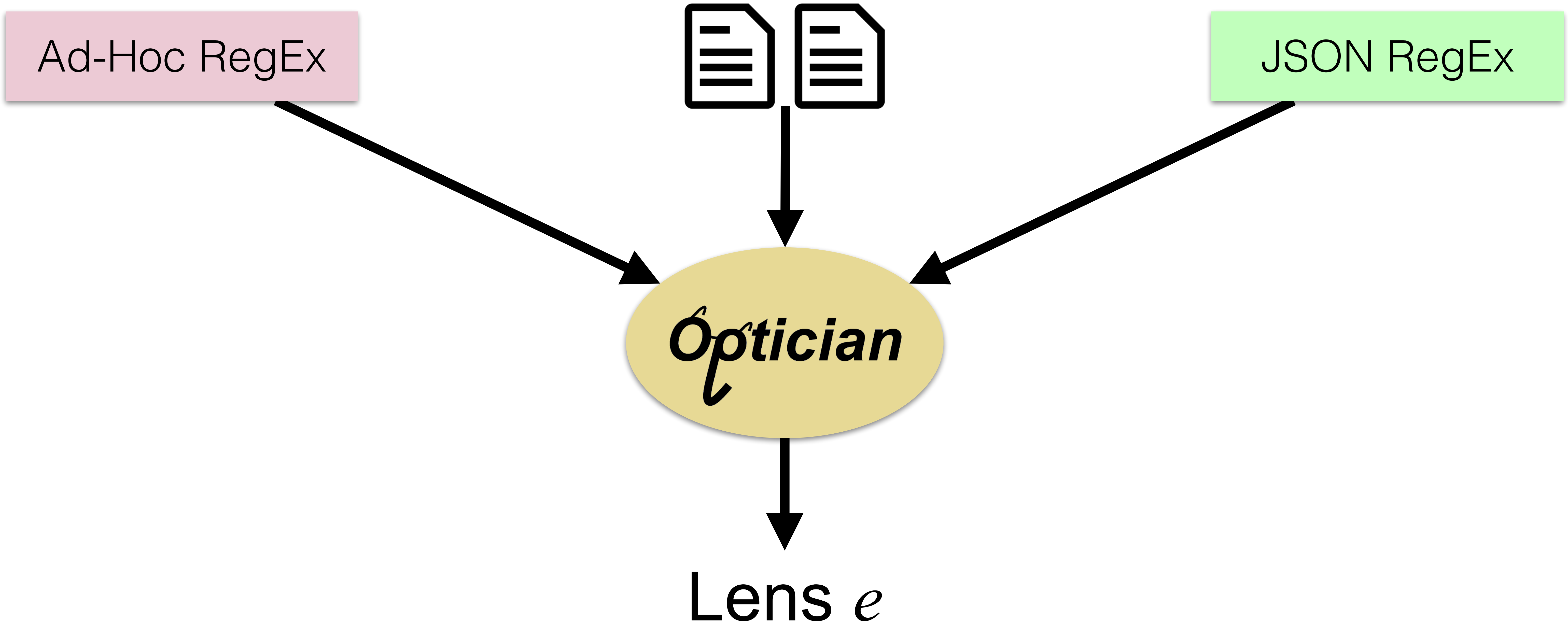
Lens Domains

Strings



[Bohannon, et. al. 2008]





Ad-Hoc RegEx

JSON RegEx

Optician

Lens e

Stephen Cole Kleene   Kleene, Stephen Cole

```
let name        = [A-Z][a-z]* in
let name_map    = Id(name) in
let first_map   = name_map ~ Id(“ “) in
let firsts_map = first_map* in
let sep_map     = const(“”,”,”) in
let head_map    = sep_map . firsts_map in
head_map ~ name_map
```

Stephen Cole Kleene   Kleene, Stephen Cole

```
let name        = [A-Z][a-z]* in
let name_map    = Id(name) in
let first_map   = name_map ~ Id(“ “) in
let firsts_map  = first_map* in
let sep_map     = const(“”,”,”) in
let head_map    = sep_map . firsts_map in
head_map ~ name_map
```

Stephen Cole Kleene   Kleene, **Stephen** Cole

```
let name        = [A-Z][a-z]* in
let name_map    = Id(name) in
let first_map   = name_map ~ Id(“ “) in
let firsts_map = first_map* in
let sep_map     = const(“”,”,”) in
let head_map    = sep_map . firsts_map in
head_map ~ name_map
```

Stephen_ Cole Kleene   Kleene,**_Stephen** Cole

```
let name        = [A-Z][a-z]* in
let name_map    = Id(name) in
let first_map   = name_map ~ Id(“ “) in
let firsts_map = first_map* in
let sep_map     = const(“”,”,”) in
let head_map    = sep_map . firsts_map in
head_map ~ name_map
```

Stephen_Cole_ Kleene

Kleene,**_Stephen_Cole**

```
let name        = [A-Z][a-z]* in
let name_map    = Id(name) in
let first_map   = name_map ~ Id(“ “) in
let firsts_map = first_map* in
let sep_map     = const(“”,”,”) in
let head_map    = sep_map . firsts_map in
head_map ~ name_map
```

Stephen Cole Kleene   Kleene, Stephen Cole

```
let name       = [A-Z][a-z]* in
let name_map   = Id(name) in
let first_map  = name_map ~ Id(“ “) in
let firsts_map = first_map* in
let sep_map    = const(“”,”,”) in
let head_map   = sep_map . firsts_map in
head_map ~ name_map
```

Stephen_Cole_ Kleene   Kleene**,_Stephen_Cole**

```
let name        = [A-Z][a-z]* in
let name_map    = Id(name) in
let first_map   = name_map ~ Id(“ “) in
let firsts_map = first_map* in
let sep_map     = const(“”,”,”) in
let head_map    = sep_map . firsts_map in
head_map ~ name_map
```

Stephen_Cole_Kleene **Kleene,_Stephen_Cole**

```
let name        = [A-Z][a-z]* in
let name_map    = Id(name) in
let first_map   = name_map ~ Id(“ “) in
let firsts_map = first_map* in
let sep_map     = const(“”,”,”) in
let head_map    = sep_map . firsts_map in
head_map ~ name_map
```

Lens Grammar

$e :=$	$\texttt{const}(s_1,s_2)$	constant
$\mid$	$\texttt{Id}(R)$	identity
$\mid$	$e_1 \;\texttt{.}\; e_2$	concatenation
$\mid$	$e_1 \;\texttt{\~{}}\; e_2$	swap
$\mid$	$e_1 \mid e_2$	union
$\mid$	$e\texttt{*}$	iteration
$\mid$	$e_1 \;\texttt{;}\; e_2$	composition

Lens Typing Judgment

Why?

Lens Typing Judgment

Why?

1. To specify the languages the lens maps between

Lens Typing Judgment

Why?

1. To specify the languages the lens maps between

2. To guarantee that the *get* and *put* functions are well-defined, and are inverses

Lens Typing Judgment

Why?

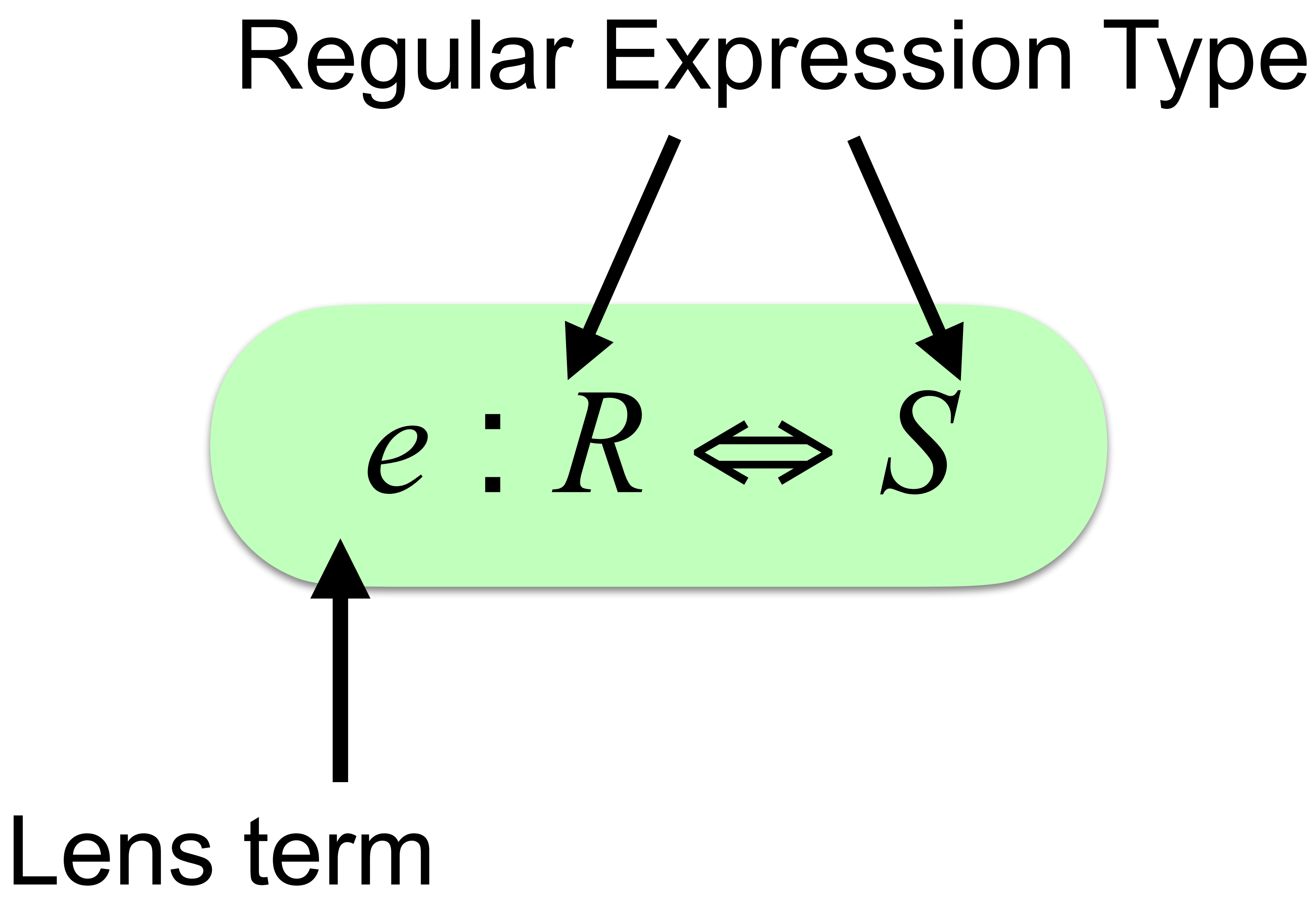
1. To specify the languages the lens maps between
2. To guarantee that the *get* and *put* functions are well-defined, and are inverses

Also… it’s useful for synthesis!

Lens Typing Judgment

$$e : R \Leftrightarrow S$$

Lens Typing Judgment



Regular Expression Type

$$e : R \Leftrightarrow S$$

Lens term

Lens Typing Judgment

$e : R \Leftrightarrow S$ means

Lens Typing Judgment

$e : R \Leftrightarrow S$ means

Lens e provides 2 functions:
$e.put$ from $L(R)$ to $L(S)$
$e.get$ from $L(S)$ to $L(R)$

Lens Typing Judgment

$e : R \Leftrightarrow S$ means

Lens e provides 2 functions:
$e.put$ from $L(R)$ to $L(S)$
$e.get$ from $L(S)$ to $L(R)$

Consequences:
$e.put\ (e.get\ s) = s$
$e.get\ (e.put\ r) = r$

$e.put$ and $e.get$ are inverses

Lens Typing Rules

3 sorts of rules

Lens Typing Rules

3 sorts of rules

1. Syntax-Directed Rules (concatenation, iteration, …)

Lens Typing Rules

3 sorts of rules

1. Syntax-Directed Rules (concatenation, iteration, …)
2. Composition

Lens Typing Rules

3 sorts of rules

1. Syntax-Directed Rules (concatenation, iteration, …)
2. Composition
3. Type Equivalence

Lens Typing Rules

3 sorts of rules

1. Syntax-Directed Rules (concatenation, iteration, …)
2. Composition
3. Type Equivalence

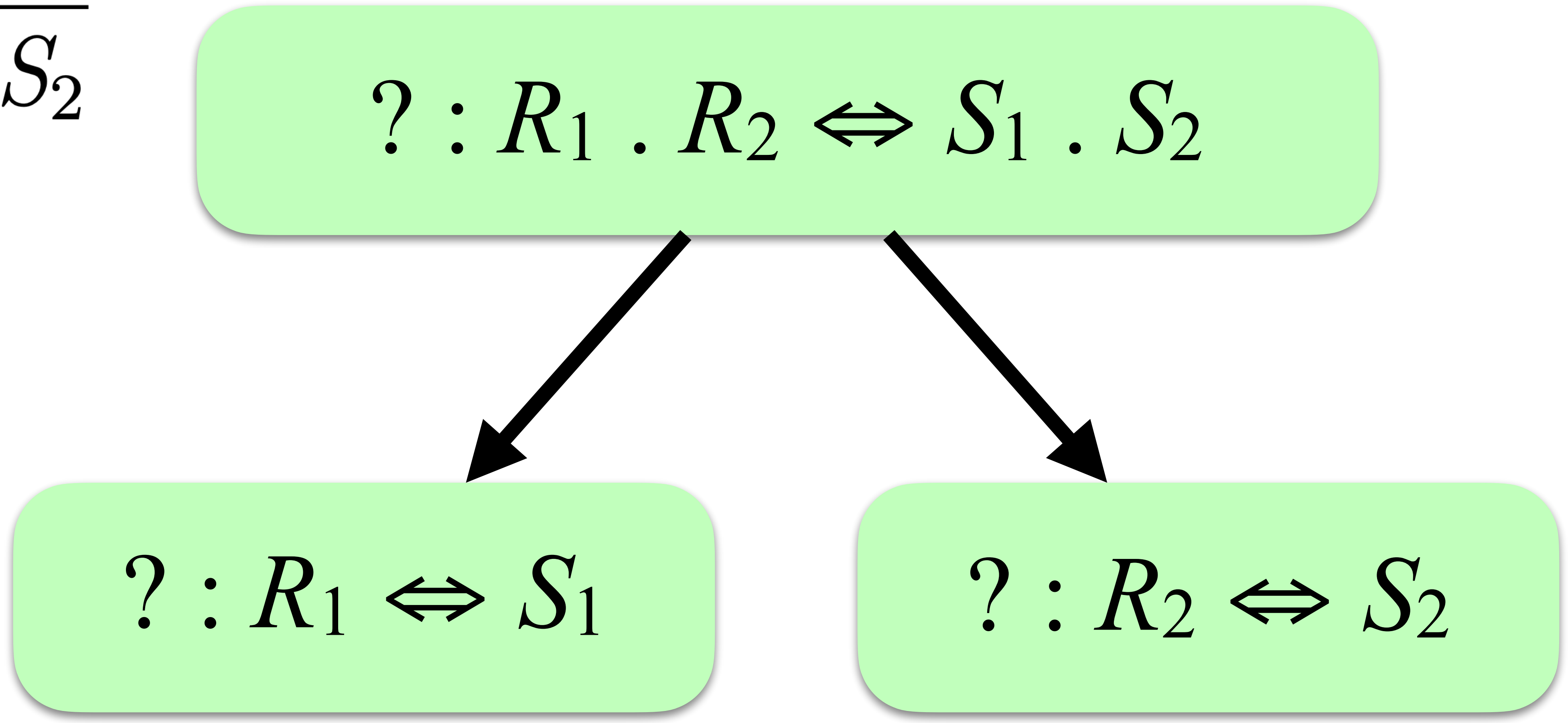
Example Syntax-Directed Rule

$$? : R_1 \, . \, R_2 \Leftrightarrow S_1 \, . \, S_2$$

Example Syntax-Directed Rule

Concat Lens

$$\frac{\begin{array}{c} \ell_1 : R_1 \Leftrightarrow S_1 \\ \ell_2 : R_2 \Leftrightarrow S_2 \\ R_1 \cdot^{!} R_2 \qquad S_1 \cdot^{!} S_2 \end{array}}{\mathsf{concat}(\ell_1, \ell_2) : R_1 R_2 \Leftrightarrow S_1 S_2}$$



$? : R_1 \;.\; R_2 \Leftrightarrow S_1 \;.\; S_2$

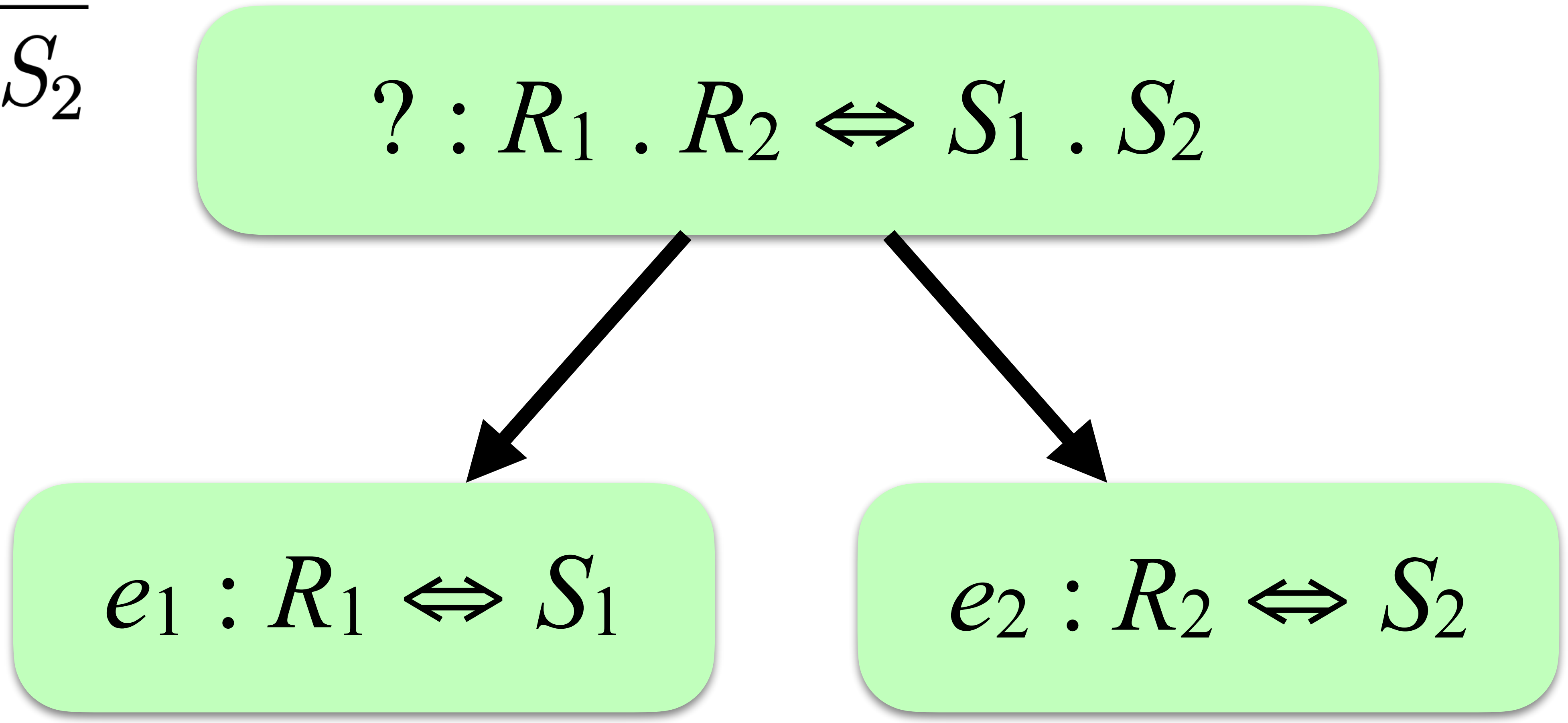
$? : R_1 \Leftrightarrow S_1$

$? : R_2 \Leftrightarrow S_2$

Example Syntax-Directed Rule

CONCAT LENS

$$\frac{\begin{array}{c}\ell_1 : R_1 \Leftrightarrow S_1 \\ \ell_2 : R_2 \Leftrightarrow S_2 \\ R_1 \cdot^{!} R_2 \qquad S_1 \cdot^{!} S_2\end{array}}{\mathsf{concat}(\ell_1, \ell_2) : R_1 R_2 \Leftrightarrow S_1 S_2}$$



$? : R_1 \,.\, R_2 \Leftrightarrow S_1 \,.\, S_2$

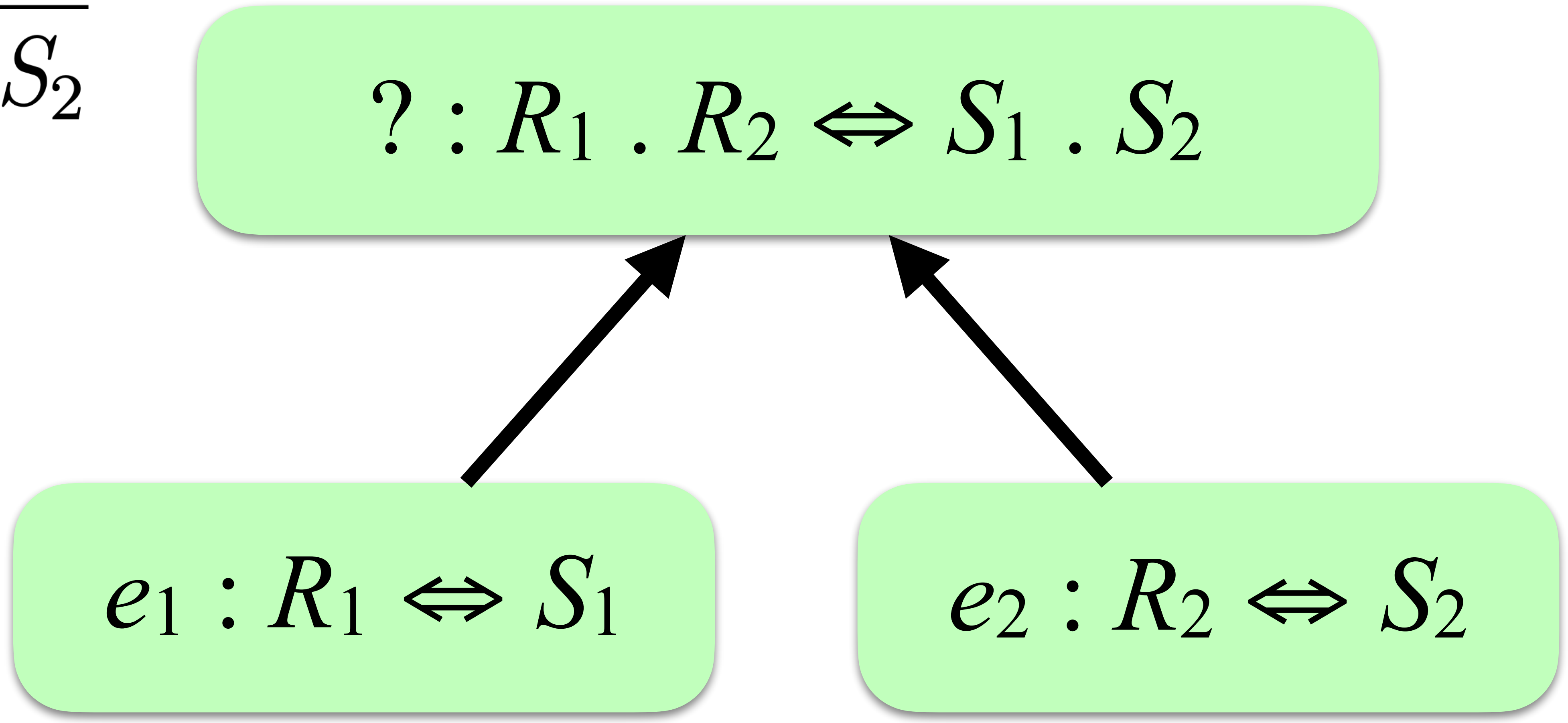
$e_1 : R_1 \Leftrightarrow S_1$

$e_2 : R_2 \Leftrightarrow S_2$

Example Syntax-Directed Rule

Concat Lens

$$\frac{\ell_1 : R_1 \Leftrightarrow S_1 \qquad \ell_2 : R_2 \Leftrightarrow S_2 \qquad R_1 \cdot^{!} R_2 \qquad S_1 \cdot^{!} S_2}{\textsf{concat}(\ell_1, \ell_2) : R_1 R_2 \Leftrightarrow S_1 S_2}$$



$? : R_1 \,.\, R_2 \Leftrightarrow S_1 \,.\, S_2$

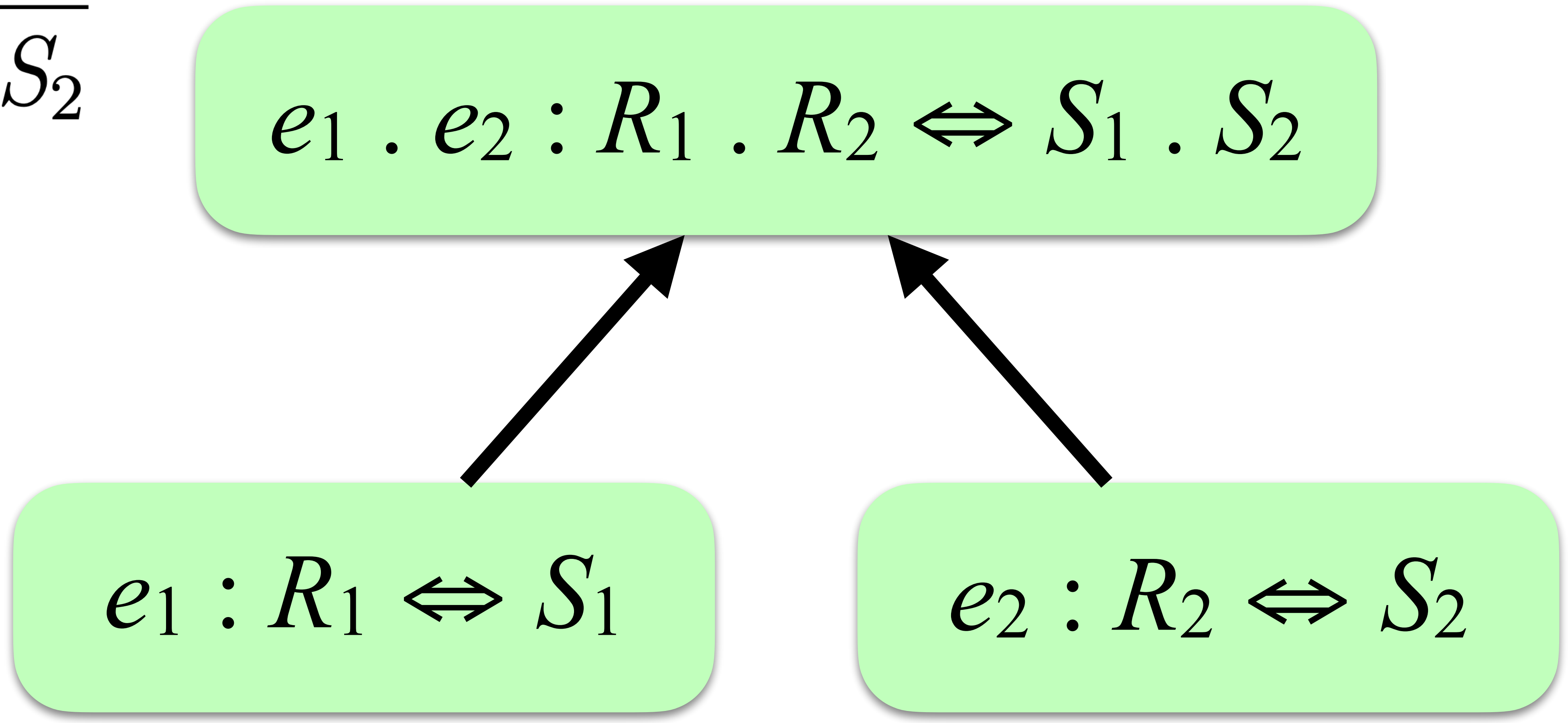
$e_1 : R_1 \Leftrightarrow S_1$

$e_2 : R_2 \Leftrightarrow S_2$

Example Syntax-Directed Rule

Concat Lens

$$\frac{\ell_1 : R_1 \Leftrightarrow S_1 \qquad \ell_2 : R_2 \Leftrightarrow S_2 \qquad R_1 \cdot^! R_2 \qquad S_1 \cdot^! S_2}{\mathsf{concat}(\ell_1, \ell_2) : R_1 R_2 \Leftrightarrow S_1 S_2}$$



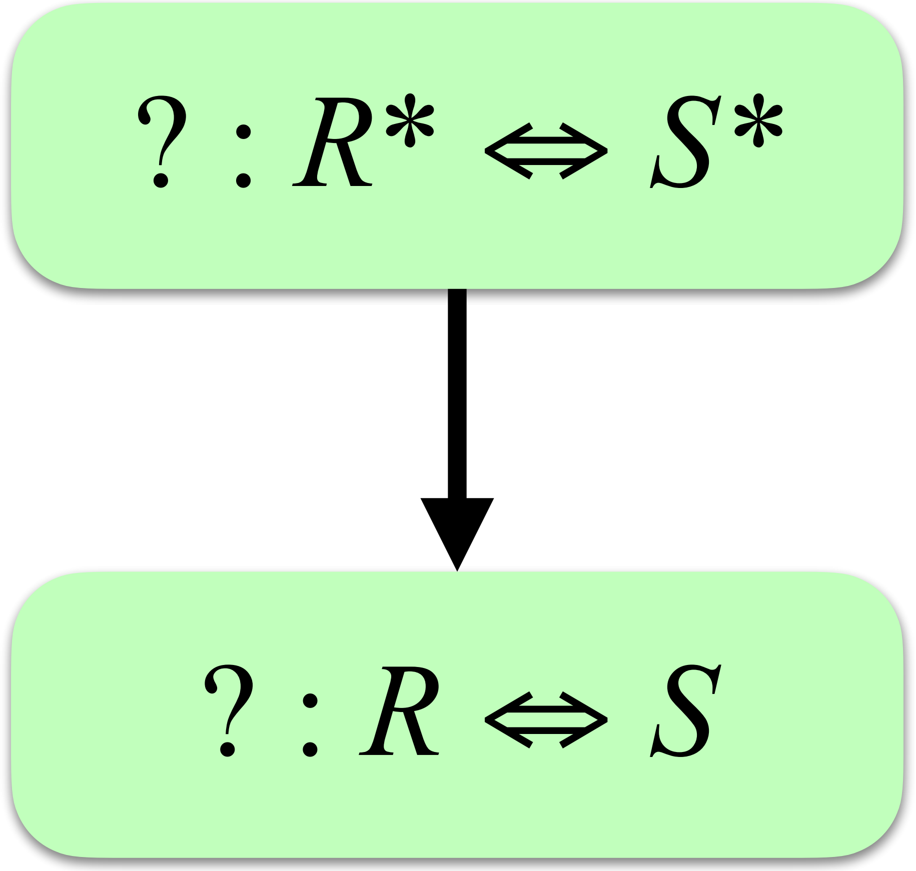
$e_1 \,.\, e_2 : R_1 \,.\, R_2 \Leftrightarrow S_1 \,.\, S_2$

$e_1 : R_1 \Leftrightarrow S_1$

$e_2 : R_2 \Leftrightarrow S_2$

Example Syntax-Directed Rule

$$\textsc{Iterate Lens}\quad \frac{\ell : R \Leftrightarrow S \qquad R^{*!} \qquad S^{*!}}{\ell^{*} : R^{*} \Leftrightarrow S^{*}}$$



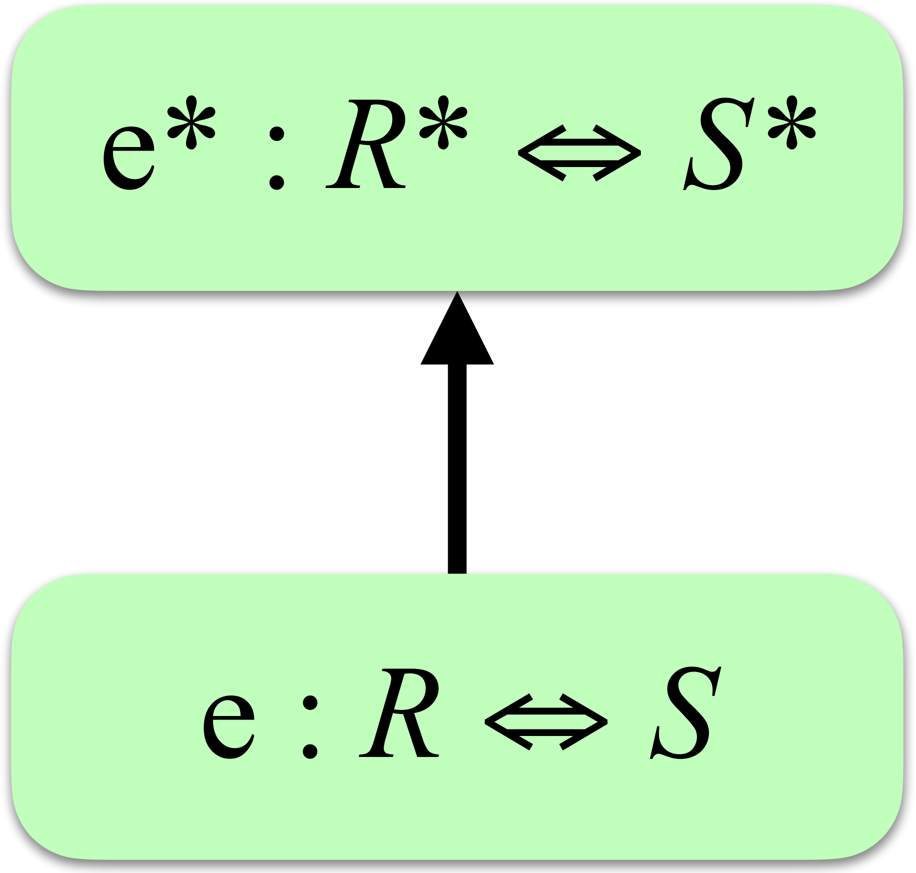
$? : R^* \Leftrightarrow S^*$

$? : R \Leftrightarrow S$

Example Syntax-Directed Rule

ITERATE LENS

$$\frac{\ell : R \Leftrightarrow S \qquad R^{*!} \qquad S^{*!}}{\ell^{*} : R^{*} \Leftrightarrow S^{*}}$$



$e^{*} : R^{*} \Leftrightarrow S^{*}$

$e : R \Leftrightarrow S$

Syntax-Directed Rule Base Case

$$? : R \Leftrightarrow R$$

Syntax-Directed Rule Base Case

$$\dfrac{\textsc{Identity Lens} \\ R \text{ is strongly unambiguous}}{\mathsf{id}(R) : R \Leftrightarrow R}$$

$\texttt{id(}R\texttt{)} : R \Leftrightarrow R$

Syntax-Directed Rule Base Case

$$? : s \Leftrightarrow t$$

Syntax-Directed Rule Base Case

CONSTANT LENS

$$\frac{s_1 \in \Sigma^* \qquad s_2 \in \Sigma^*}{\mathsf{const}(s_1\, s_2) : s_1 \Leftrightarrow s_2}$$

$\texttt{const}(s,t) : s \Leftrightarrow t$

Lens Typing Rules

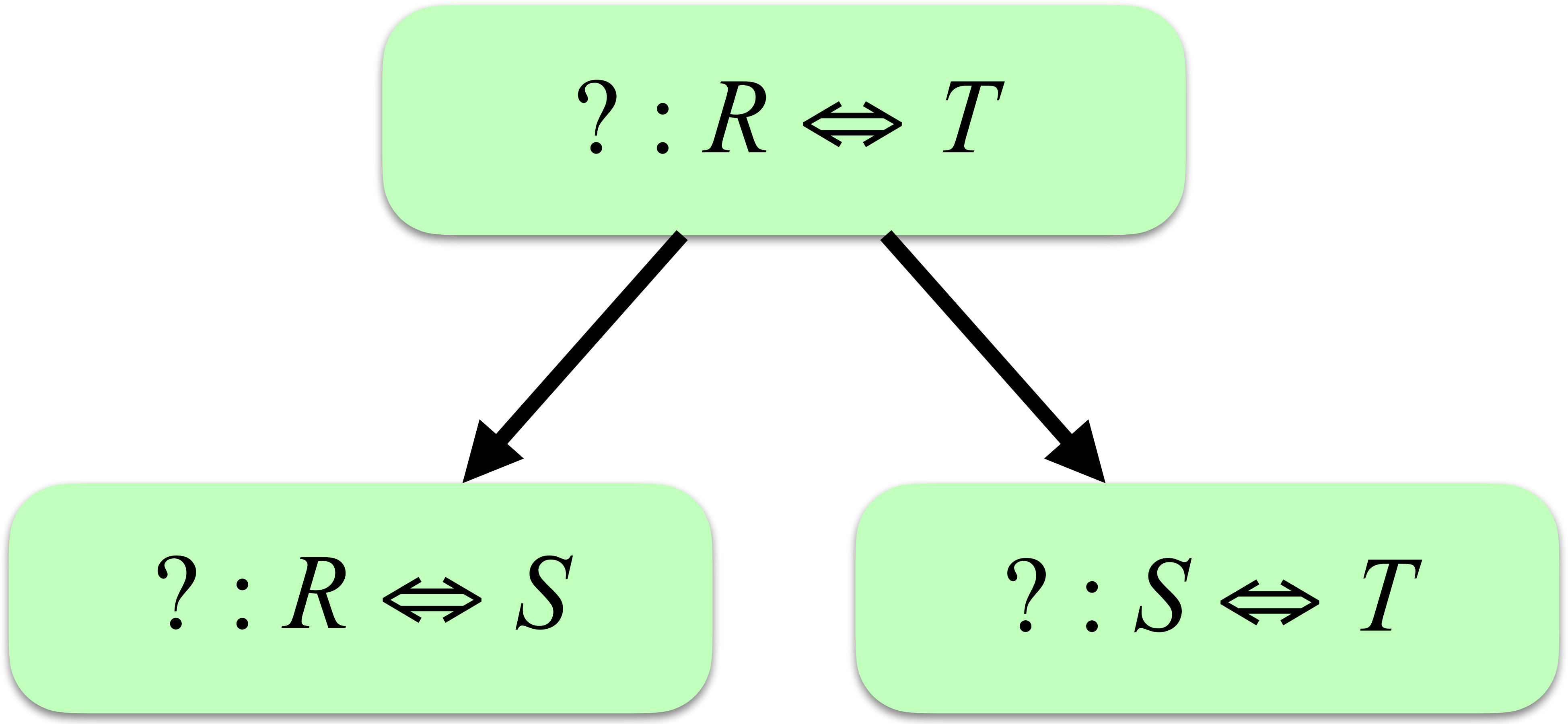
3 sorts of rules

1. Syntax-Directed Rules (concatenation, iteration, …)
2. Composition
3. Type Equivalence

Composition

Compose Lens

$$\frac{\ell_1 : R_1 \Leftrightarrow R_2 \qquad \ell_2 : R_2 \Leftrightarrow R_3}{\ell_1 \mathbin{;} \ell_2 : R_1 \Leftrightarrow R_3}$$



$? : R \Leftrightarrow T$

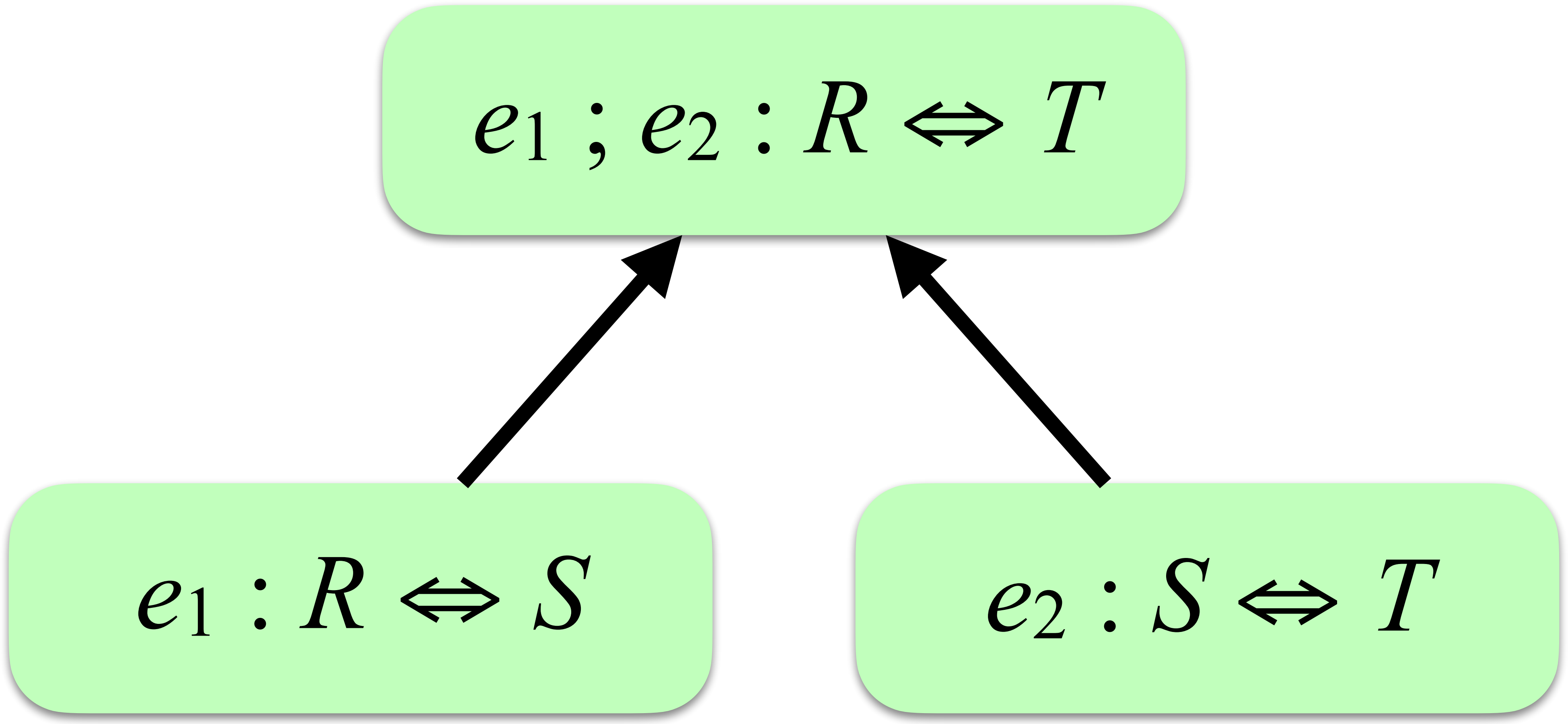
$? : R \Leftrightarrow S$

$? : S \Leftrightarrow T$

Composition

Compose Lens

$$\frac{\ell_1 : R_1 \Leftrightarrow R_2 \qquad \ell_2 : R_2 \Leftrightarrow R_3}{\ell_1 \,;\, \ell_2 : R_1 \Leftrightarrow R_3}$$



$e_1 \,;\, e_2 : R \Leftrightarrow T$

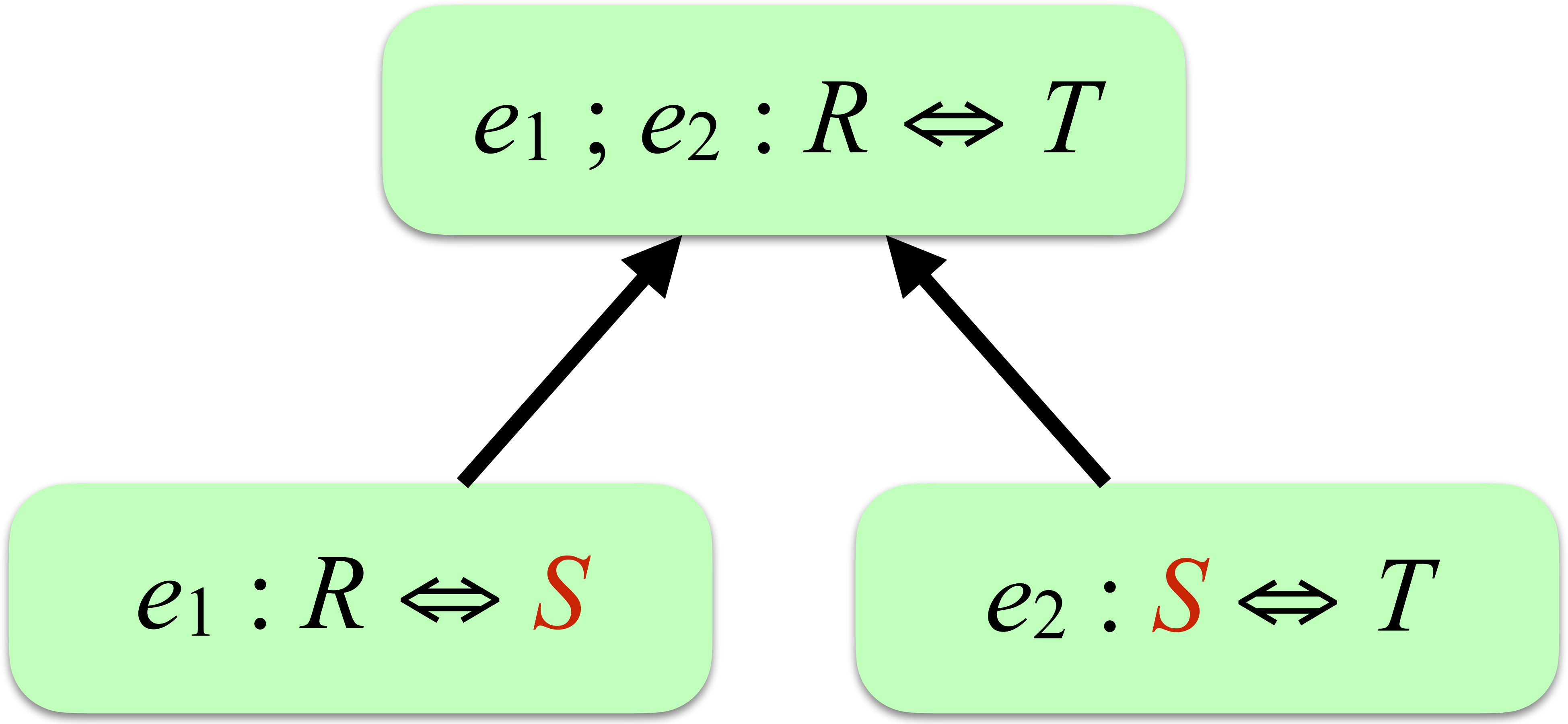
$e_1 : R \Leftrightarrow S$

$e_2 : S \Leftrightarrow T$

Composition

Compose Lens

$$\frac{\ell_1 : R_1 \Leftrightarrow R_2 \qquad \ell_2 : R_2 \Leftrightarrow R_3}{\ell_1 \,;\, \ell_2 : R_1 \Leftrightarrow R_3}$$



$e_1 \,;\, e_2 : R \Leftrightarrow T$

$e_1 : R \Leftrightarrow S$

$e_2 : S \Leftrightarrow T$

Composition Inadmissibility

$$\begin{tikzcd}
A\ B\ C\ D \arrow[r, leftrightarrow, "\texttt{perm1}"] & C\ A\ B\ D \arrow[r, leftrightarrow, "\texttt{perm2}"] & C\ A\ D\ B
\end{tikzcd}$$

```
let l     = Id([A-Z])         in
let perm1 = ((l . l) ~ l) . l in
let perm2 = l . l . (l ~ l)   in
perm1 ; perm2
```

Solution:
Use Alternative Language

DNF Lenses

Solution:
Use Alternative Language

DNF Lenses

No composition operator

Lens Typing Rules

3 sorts of rules

1. Syntax-Directed Rules (concatenation, iteration, …)
2. Composition
3. Type Equivalence

Type Equivalence

$$\frac{\begin{array}{c} l : R \Leftrightarrow S \\ R \equiv^{s} R' \\ S \equiv^{s} S' \end{array}}{l : R' \Leftrightarrow S'}$$

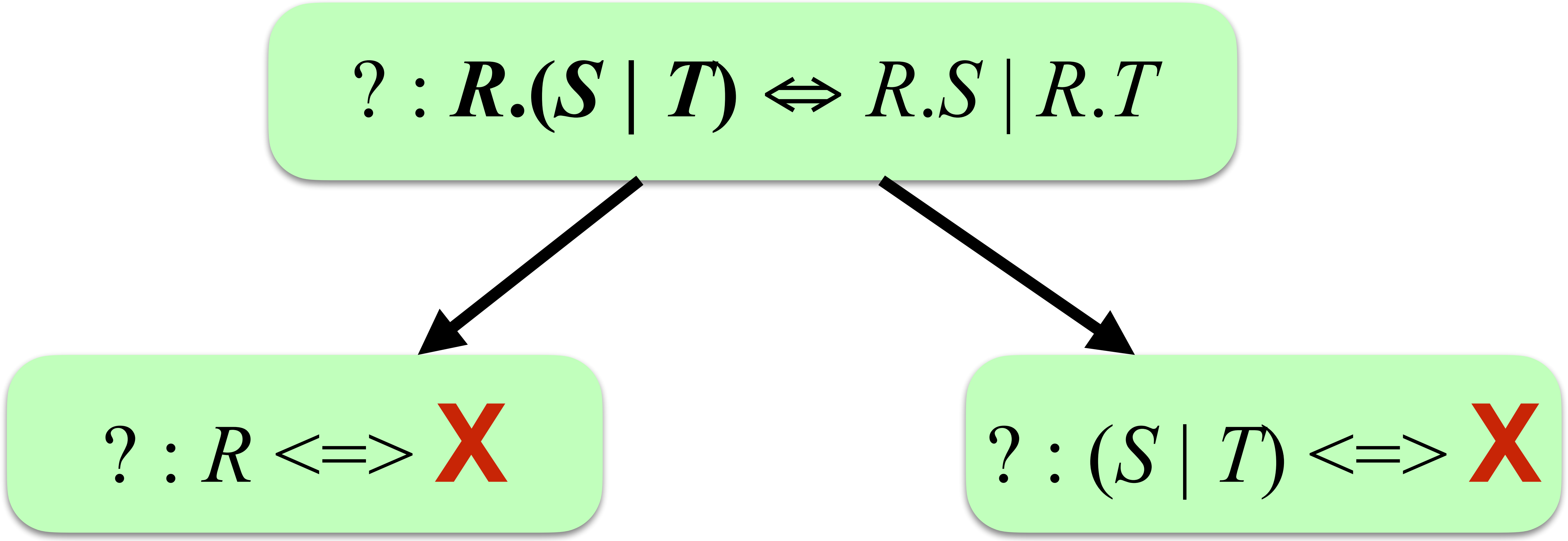
Example Type Equivalence Rule

$$? : R.(S \mid T) \Leftrightarrow R.S \mid R.T$$

Example Type Equivalence Rule

$$? : \boldsymbol{R.(S \mid T)} \Leftrightarrow R.S \mid R.T$$

Example Type Equivalence Rule



$? : \boldsymbol{R.(S \mid T)} \Leftrightarrow R.S \mid R.T$

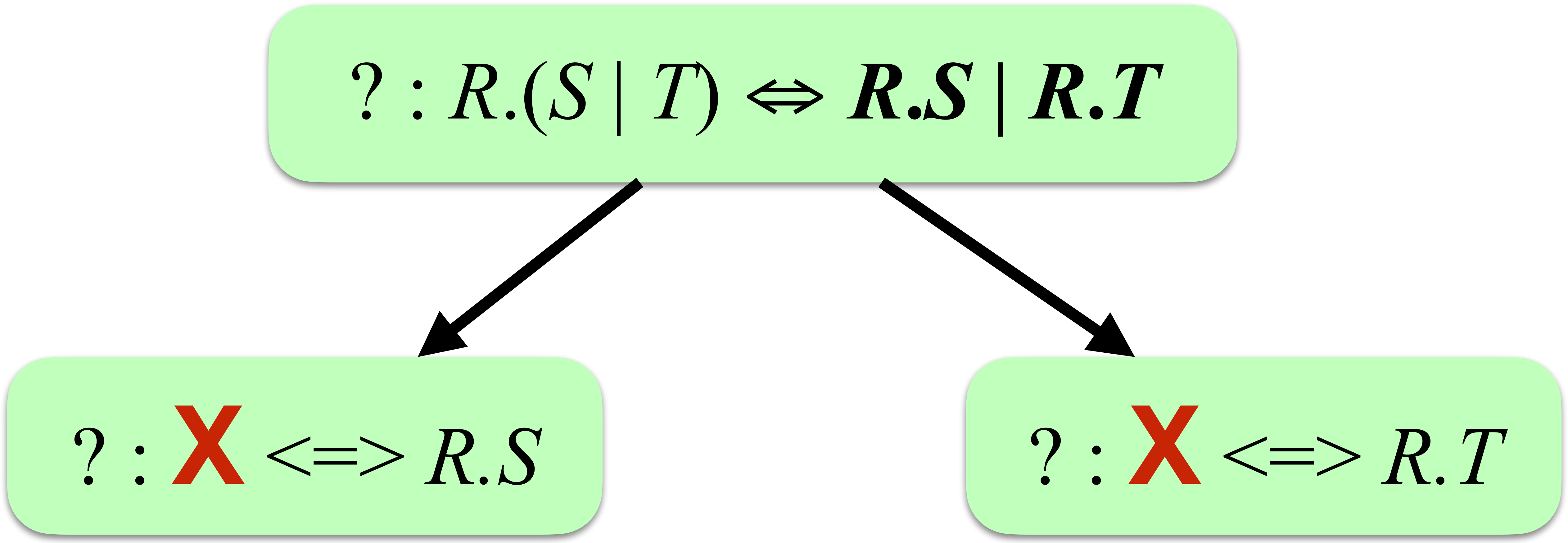
$? : R <\!\!=\!\!> $ X

$? : (S \mid T) <\!\!=\!\!> $ X

Example Type Equivalence Rule

$$? : R.(S \mid T) \Leftrightarrow \boldsymbol{R.S} \mid \boldsymbol{R.T}$$

Example Type Equivalence Rule



$? : R.(S \mid T) \Leftrightarrow \boldsymbol{R.S \mid R.T}$

$? : \mathbf{X} <=> R.S$

$? : \mathbf{X} <=> R.T$

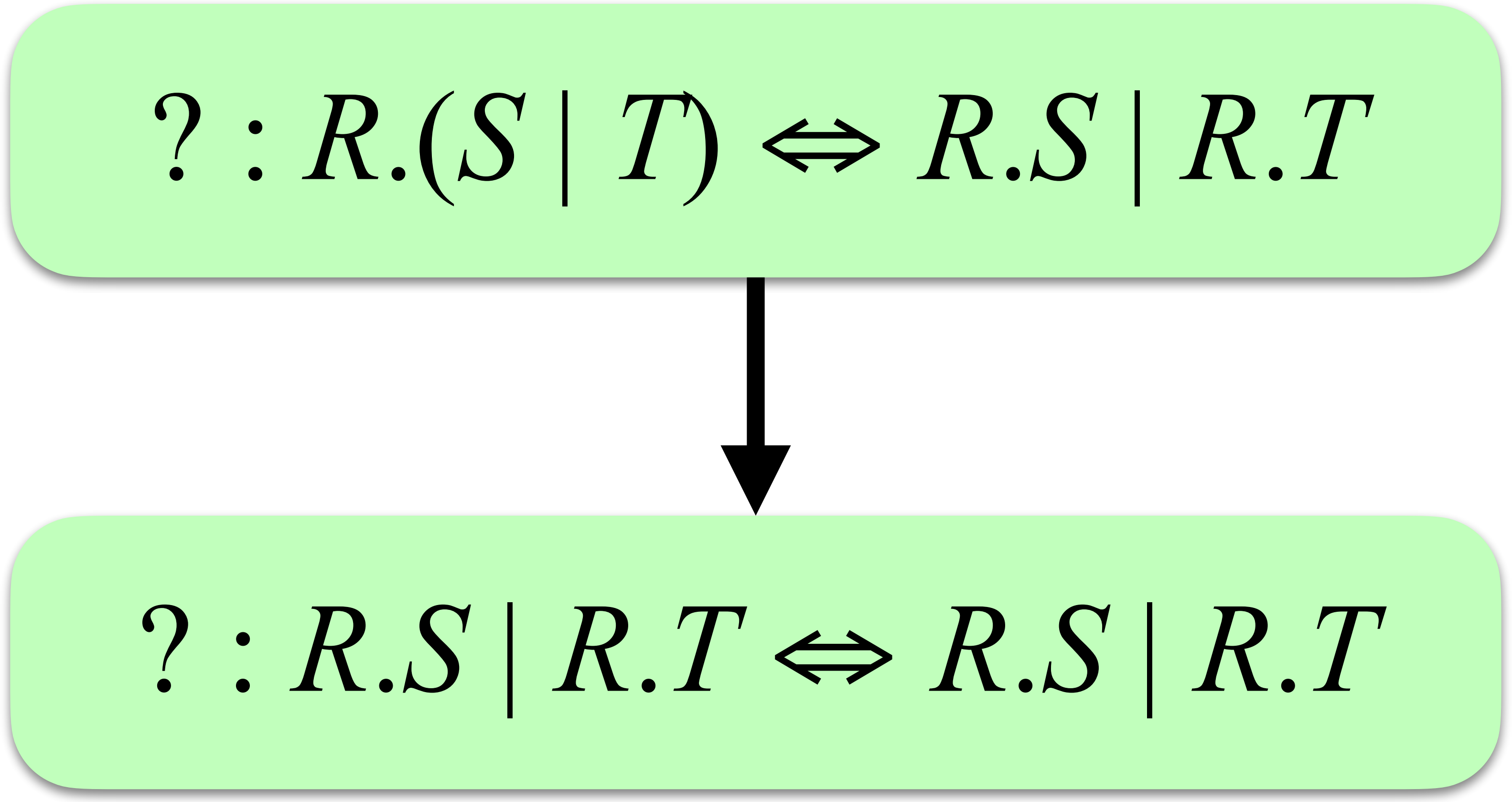
Example Type Equivalence Rule

$$? : R.(S \mid T) \Leftrightarrow R.S \mid R.T$$

Example Type Equivalence Rule

TYPE EQUIVALENCE

$$\frac{l : R \Leftrightarrow S \qquad R \equiv^s R' \qquad S \equiv^s S'}{l : R' \Leftrightarrow S'}$$



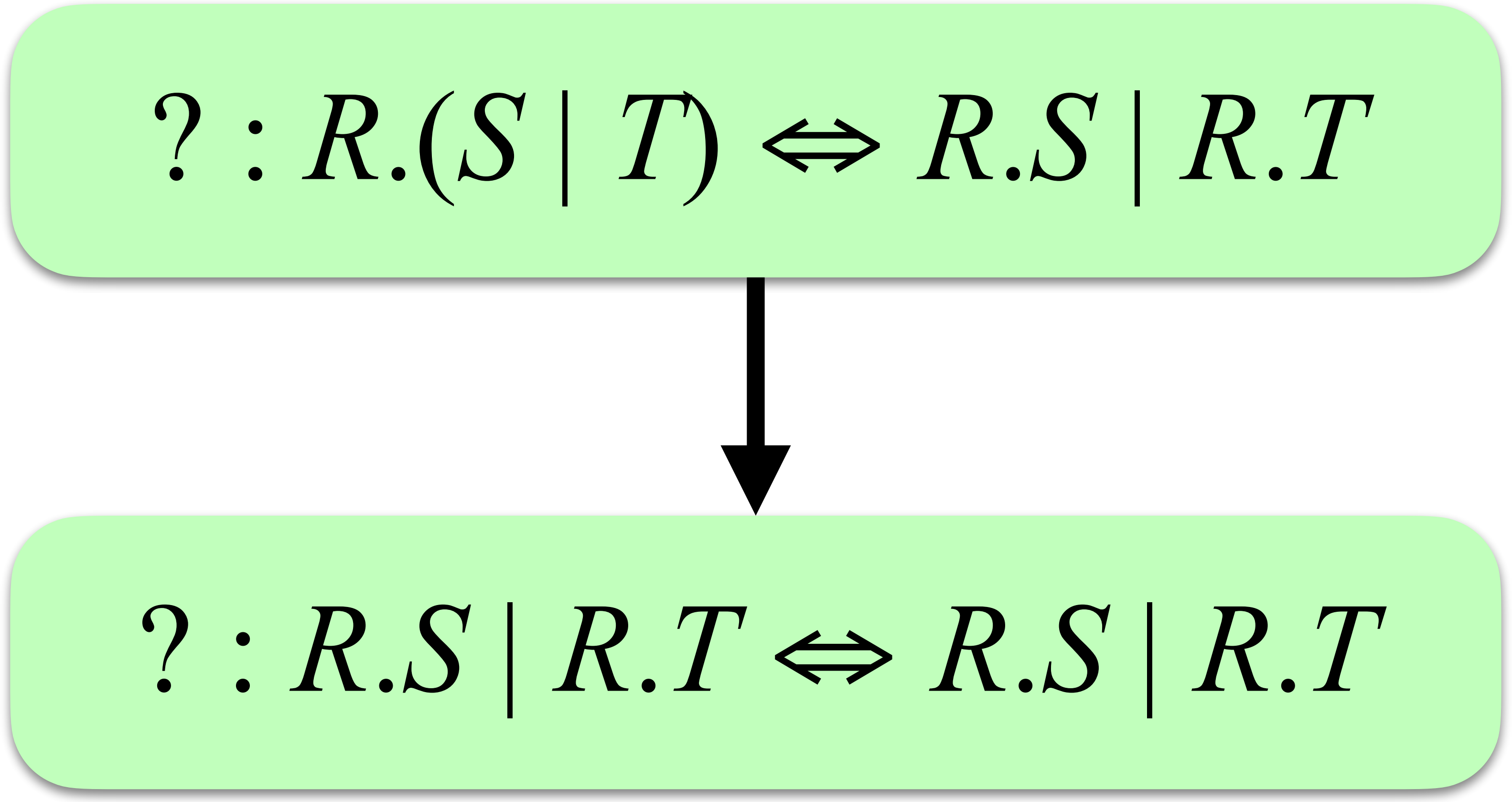
$? : R.(S \mid T) \Leftrightarrow R.S \mid R.T$

$? : R.S \mid R.T \Leftrightarrow R.S \mid R.T$

Example Type Equivalence Rule

$$\textsc{Type Equivalence}$$

$$\frac{l : R \Leftrightarrow S \qquad R \equiv^s R' \qquad S \equiv^s S'}{l : R' \Leftrightarrow S'}$$



$$? : R.(S \mid T) \Leftrightarrow R.S \mid R.T$$

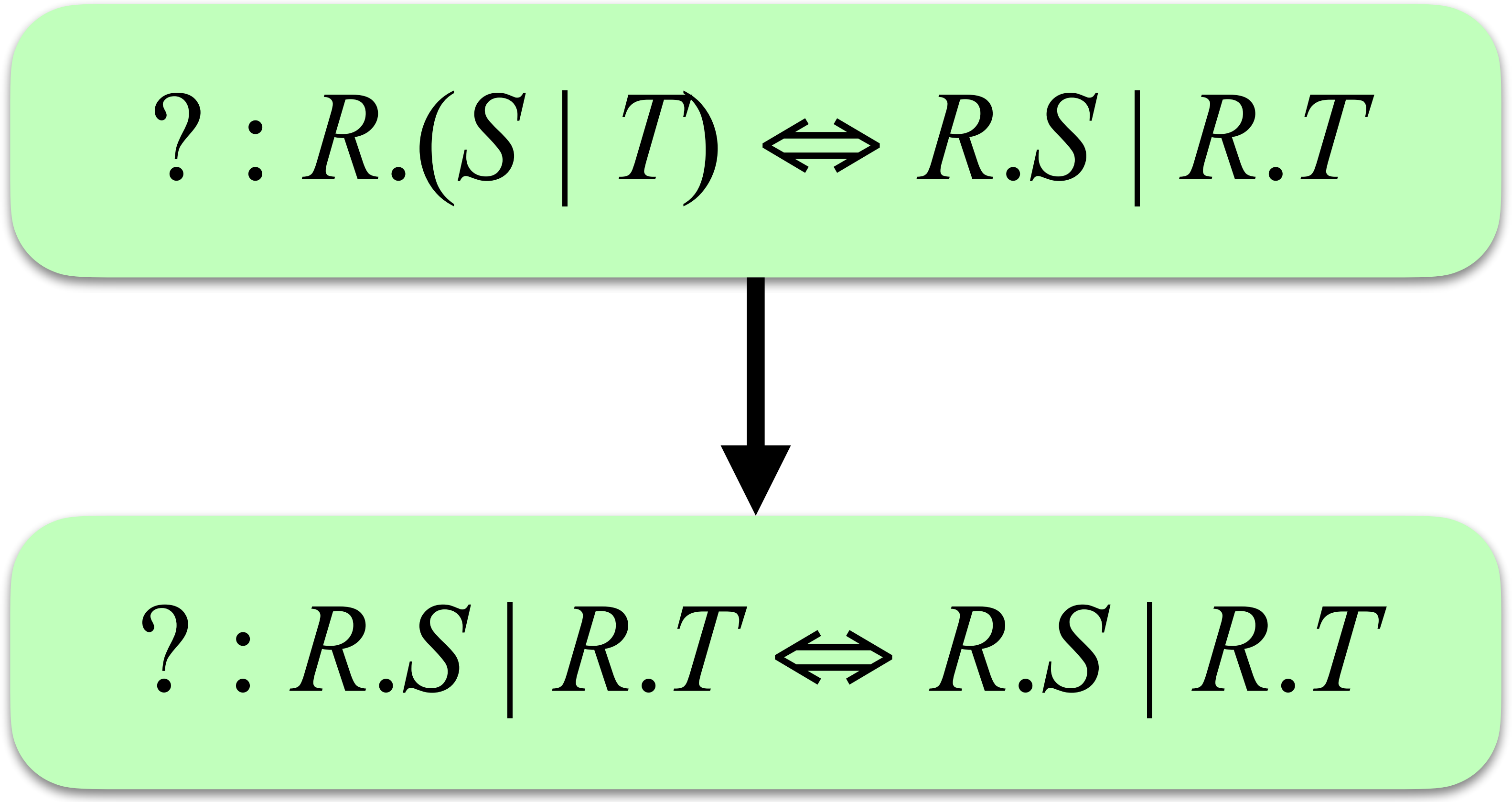
$$? : R.S \mid R.T \Leftrightarrow R.S \mid R.T$$

Search through equivalent regular expression pairs

Example Type Equivalence Rule

Type Equivalence

$$\frac{l : R \Leftrightarrow S \qquad R \equiv^{s} R' \qquad S \equiv^{s} S'}{l : R' \Leftrightarrow S'}$$



$$? : R.(S \mid T) \Leftrightarrow R.S \mid R.T$$

$$? : R.S \mid R.T \Leftrightarrow R.S \mid R.T$$

Search through **equivalent regular expression pairs**

Solution:
Use Alternative Language

DNF Regular Expressions

Solution:
Use Alternative Language

DNF Regular Expressions

- Pseudo-canonical form

Solution:
Use Alternative Language

DNF Regular Expressions

- Pseudo-canonical form

Disjunctions
↓
Concatenations
↓
Iterations, Variables, and Strings

Solution:
Use Alternative Language

DNF Regular Expressions

- Pseudo-canonical form

Disjunctions
↓
Concatenations
↓
Iterations, Variables,
and Strings

`“a” . (“b” | “c”)`

`(“a”.“b”) | (“a”.“c”)`

Solution:
Use Alternative Language

DNF Regular Expressions

- Pseudo-canonical form

Disjunctions
↓
Concatenations
↓
Iterations, Variables,
and Strings

DNF Lenses are actually typed between DNF regular expression pairs

`“a” . (“b” | “c”)`

`(“a”.“b”) | (“a”.“c”)`

Example Type Equivalence Rule

$$R^* = \text{“”} \mid RR^*$$
$$R^* = \text{“”} \mid R^*R$$

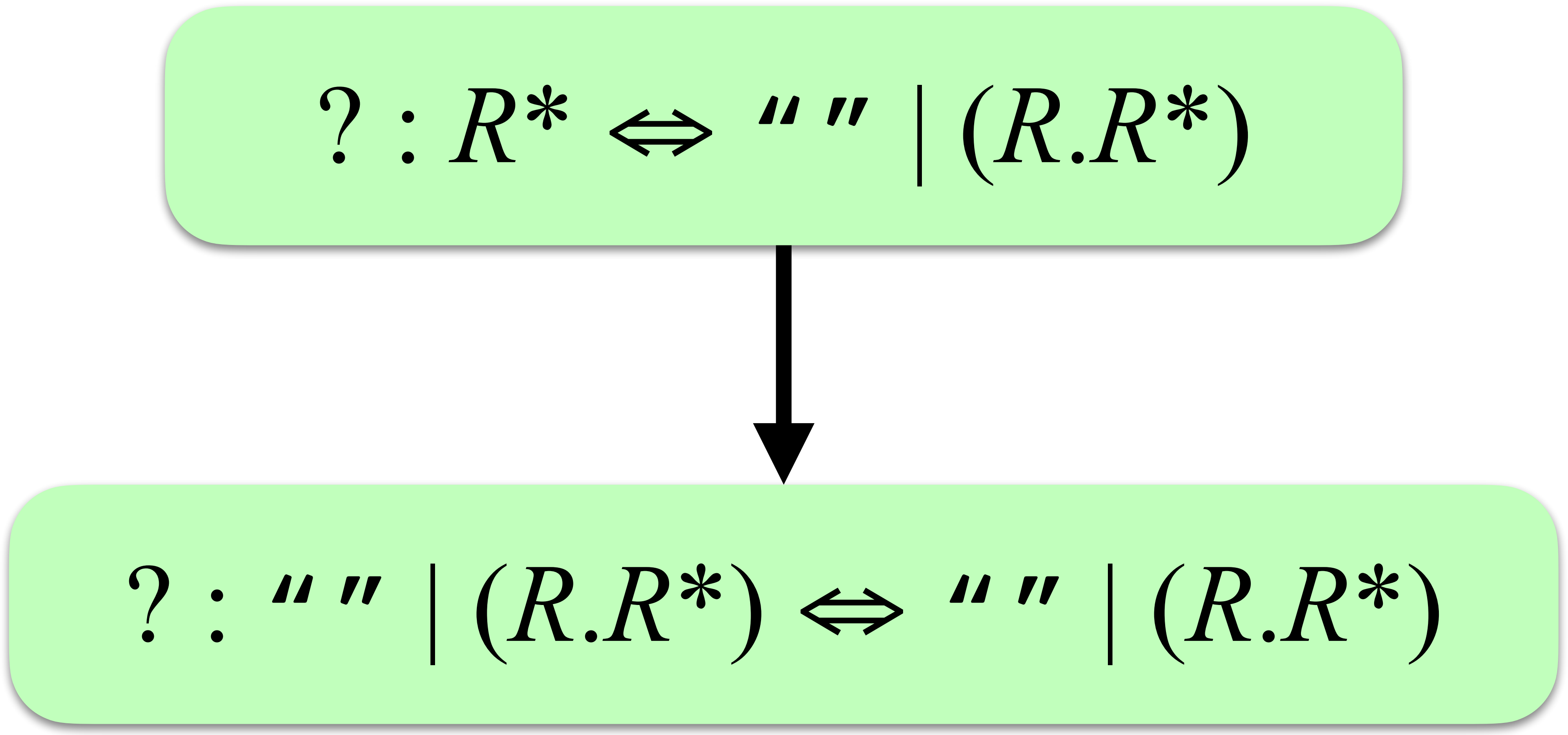
Example Type Equivalence Rule

$$\begin{aligned} R^* &= \text{“”} \mid RR^* \\ R^* &= \text{“”} \mid R^*R \end{aligned}$$

$$\begin{tikzcd} ?: R^* \Leftrightarrow \text{“”} \mid (R.R^*) \arrow[d] \\ ?: \text{“”} \mid (R.R^*) \Leftrightarrow \text{“”} \mid (R.R^*) \end{tikzcd}$$

Example Type Equivalence Rule

$$
\begin{aligned}
R^* &= \text{“”} \mid RR^* \\
R^* &= \text{“”} \mid R^*R
\end{aligned}
$$



$$? : R^* \Leftrightarrow \text{“”} \mid (R.R^*)$$

$$? : \text{“”} \mid (R.R^*) \Leftrightarrow \text{“”} \mid (R.R^*)$$

NOT Syntactically Equal in DNF Form!

DNF Lens Typing Judgement(s)

DNF Lens Typing

$$dl : DR \Leftrightarrow DS$$

Rewriteless DNF Lens Typing

$$dl \mathrel{\tilde{:}} DR \Leftrightarrow DS$$

DNF Lens Typing Judgement(s)

Rewrite DNF Regex Lens

$$\frac{DR' \rightarrow^{*} DR \qquad DS' \rightarrow^{*} DS \qquad dl \mathrel{\tilde{:}} DR \Leftrightarrow DS}{dl : DR' \Leftrightarrow DS'}$$

DNF Lens Typing

$$dl : DR \Leftrightarrow DS$$

Addresses Rewrites

Rewriteless DNF Lens Typing

$$dl \mathrel{\tilde{:}} DR \Leftrightarrow DS$$

Addresses Syntax-Directed Rules

We have a complete search algorithm for DNF lenses…

We can convert DNF lenses into bijective lenses…

Is this a complete search procedure for bijective lenses?

Completeness Theorem

For all

$$l : R \Leftrightarrow S$$

Completeness Theorem

For all

$$l : R \Leftrightarrow S$$

There exists

$$dl : DR \Leftrightarrow DS$$

Where DR and DS are R and S in DNF form

Completeness Theorem

For all

$l : R \Leftrightarrow S$

There exists

$dl : DR \Leftrightarrow DS$

Where DR and DS are R and S in DNF form

Such that

$[[l]] = [[dl]]$

By Induction over the derivation of $l : R \Leftrightarrow S$

Whole Bunch of Cases

- Syntax-Directed Rules
  - Conjunction
  - Disjunction
  - …
- Composition
- Type Equivalence

Whole Bunch of Cases

- Syntax-Directed Rules
  - Conjunction
  - Disjunction
  - …
- Composition
- **Type Equivalence**

Case: Type Equivalence

$$\textsc{Type Equivalence}\quad \frac{l : R \Leftrightarrow S \qquad R \equiv^s R' \qquad S \equiv^s S'}{l : R' \Leftrightarrow S'}$$

Case: Type Equivalence

$$\textsc{Type Equivalence}\quad \frac{l : R \Leftrightarrow S \qquad R \equiv^s R' \qquad S \equiv^s S'}{l : R' \Leftrightarrow S'}$$

By induction assumption there exists dl, DR_1, DS_1, such that

$$dl : DR_1 \Leftrightarrow DS_1$$

With equivalent semantics to l

Case: Type Equivalence

Type Equivalence

$$\frac{l : R \Leftrightarrow S \qquad R \equiv^s R' \qquad S \equiv^s S'}{l : R' \Leftrightarrow S'}$$

$$dl \;: DR_1 \Leftrightarrow DS_1$$

By induction assumption there exists dl, DR_1, DS_1, such that

$$dl : DR_1 \Leftrightarrow DS_1$$

With equivalent semantics to l

Case: Type Equivalence

Type Equivalence

$$\frac{l : R \Leftrightarrow S \qquad R \equiv^s R' \qquad S \equiv^s S'}{l : R' \Leftrightarrow S'}$$

$$dl\ : DR_1 \Leftrightarrow DS_1$$

Rewrite DNF Regex Lens

$$\frac{DR_1 \rightarrow^* DR_2 \qquad DS_1 \rightarrow^* DS_2 \qquad dl \mathrel{\tilde{:}} DR_1 \Leftrightarrow DS_1}{dl : DR_2 \Leftrightarrow DS_2}$$

Case: Type Equivalence

Type Equivalence

$$\frac{l : R \Leftrightarrow S \qquad R \equiv^s R' \qquad S \equiv^s S'}{l : R' \Leftrightarrow S'}$$

$$\begin{tikzcd}
& & dl \;\tilde{:}\; DR_2 & \Leftrightarrow & DS_2 & & \\
& \cdots \arrow[ur] & & & & \cdots \arrow[ul] & \\
DR_1 \arrow[ur] & & & & & & DS_1 \arrow[ul]
\end{tikzcd}$$

Rewrite DNF Regex Lens

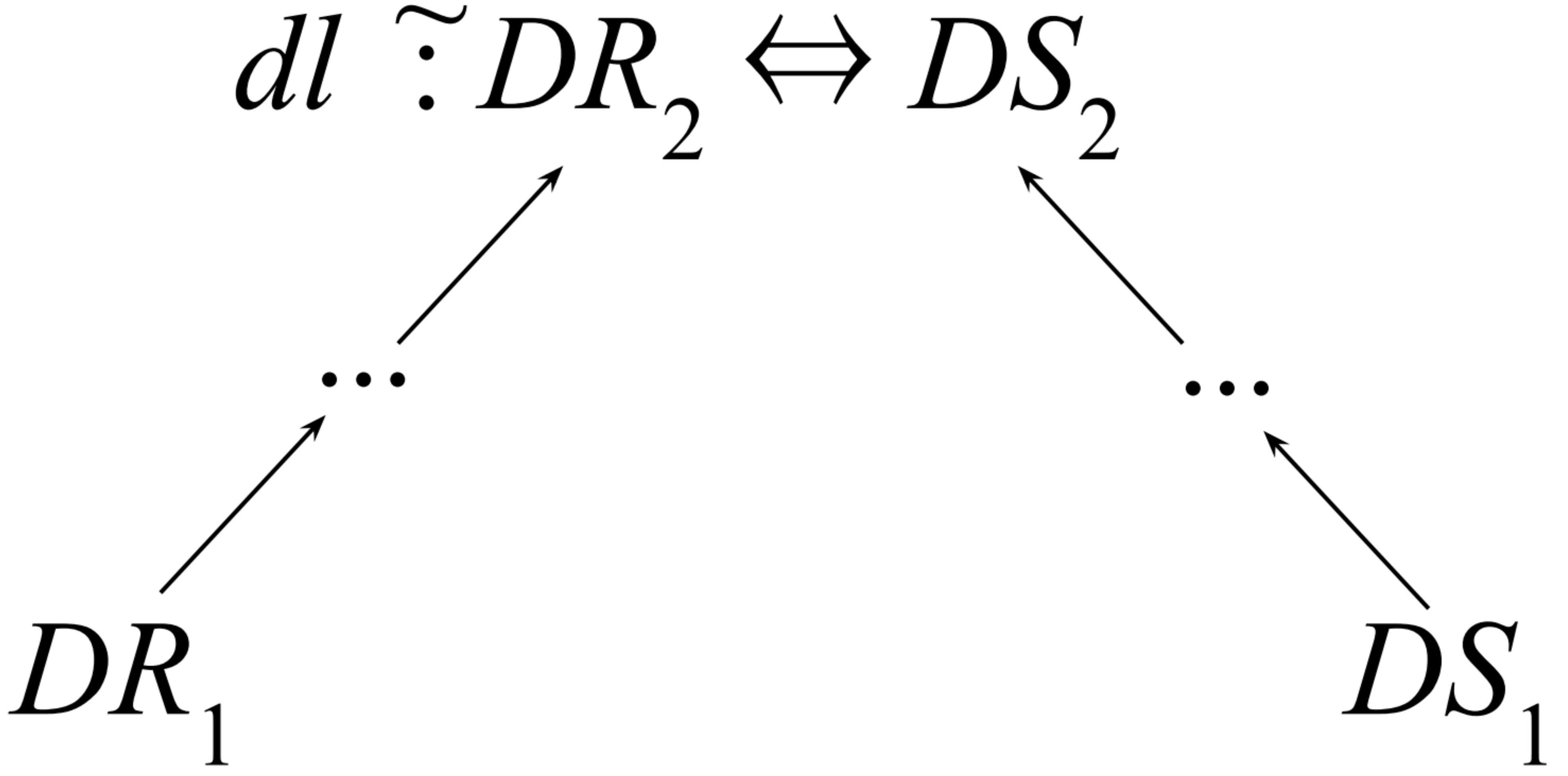
$$\frac{DR_1 \rightarrow^* DR_2 \qquad DS_1 \rightarrow^* DS_2 \qquad dl \;\tilde{:}\; DR_1 \Leftrightarrow DS_1}{dl : DR_2 \Leftrightarrow DS_2}$$

Case: Type Equivalence

Type Equivalence

$$\frac{l : R \Leftrightarrow S \qquad R \equiv^s R' \qquad S \equiv^s S'}{l : R' \Leftrightarrow S'}$$

$R \equiv^s R' \Rightarrow DR \equiv_{\rightarrow} DR'$



Rewrite DNF Regex Lens

$$\frac{DR_1 \rightarrow^* DR_2 \qquad DS_1 \rightarrow^* DS_2 \qquad dl \mathrel{\tilde{:}} DR_1 \Leftrightarrow DS_1}{dl : DR_2 \Leftrightarrow DS_2}$$

Case: Type Equivalence

Type Equivalence

$$\frac{l : R \Leftrightarrow S \qquad R \equiv^s R' \qquad S \equiv^s S'}{l : R' \Leftrightarrow S'}$$

$R \equiv^s R' \Rightarrow DR \equiv_{\rightarrow} DR'$

$$\begin{tikzcd}
DR_1' & & & dl \;\tilde{:}\; DR_2 \Leftrightarrow DS_2 & & & DS_1' \\
& & \cdots \arrow[ur] & & \cdots \arrow[ul] & & \\
& DR_1 \arrow[uul, Rrightarrow, no head] \arrow[ur] & & & & DS_1 \arrow[ul] \arrow[uur, Rrightarrow, no head] &
\end{tikzcd}$$

$DR_1 \equiv_{\rightarrow} DR_1'$
$DR_2 \equiv_{\rightarrow} DR_2'$

Rewrite DNF Regex Lens

$$\frac{DR_1 \rightarrow^* DR_2 \qquad DS_1 \rightarrow^* DS_2 \qquad dl \;\tilde{:}\; DR_1 \Leftrightarrow DS_1}{dl : DR_2 \Leftrightarrow DS_2}$$

Case: Type Equivalence

- Prove → satisfies the diamond property
- This implies the existence of such a DR_3 and DS_3

$$\begin{tikzcd}
 & & DR_3 & & & & & DS_3 & & \\
 & \cdots \arrow[ur] & & \cdots \arrow[ul] & & & \cdots \arrow[ur] & & \cdots \arrow[ul] & \\
DR_1{'} \arrow[ur] & & & & dl \overset{\sim}{:} DR_2 \arrow[ul] \arrow[r, Leftrightarrow] & DS_2 \arrow[ur] & & & & DS_1{'} \arrow[ul] \\
 & & & \cdots \arrow[ur] & & & \cdots \arrow[ul] & & & \\
 & & DR_1 \arrow[ur] \arrow[uull, phantom, "\equiv" description] & & & & & DS_1 \arrow[ul] \arrow[uurr, phantom, "\equiv" description] & &
\end{tikzcd}$$

Case: Type Equivalence

- Need dl_2 equivalent to dl
- Need $DR_3 \to^* DR_4$ and $DS_3 \to^* DS_4$

$$\begin{tikzcd}
 & & & & dl_2 \mathrel{\tilde{:}} DR_4 \arrow[r, phantom, "\Leftrightarrow"] & DS_4 & & & & \\
 & & & \cdots \arrow[ur] & & & \cdots \arrow[ul] & & & \\
 & & DR_3 \arrow[ur] & & & & & DS_3 \arrow[ul] & & \\
 & \cdots \arrow[ur] & & \cdots \arrow[ul] & & & \cdots \arrow[ur] & & \cdots \arrow[ul] & \\
DR_1' \arrow[ur] & & & & dl \mathrel{\tilde{:}} DR_2 \arrow[ul] \arrow[r, phantom, "\Leftrightarrow"] & DS_2 \arrow[ur] & & & & DS_1' \arrow[ul] \\
 & & & \cdots \arrow[ur] & & & \cdots \arrow[ul] & & & \\
 & & DR_1 \arrow[ur] \arrow[ull, phantom, sloped, "\equiv"] & & & & & DS_1 \arrow[ul] \arrow[urr, phantom, sloped, "\equiv"] & &
\end{tikzcd}$$

“Looks” a lot like confluence!

Let’s abstract this problem

$$\begin{tikzcd}
 & & dl_2 \overset{\sim}{:} DR_4 & \Leftrightarrow & DS_4 & & \\
 & \cdots \arrow[ur] & & & & \cdots \arrow[ul] & \\
DR_3 \arrow[ur] & & & & & & DS_3 \arrow[ul] \\
 & \cdots \arrow[ul] & & & & \cdots \arrow[ur] & \\
 & & dl \overset{\sim}{:} DR_2 \arrow[ul] & \Leftrightarrow & DS_2 \arrow[ur] & &
\end{tikzcd}$$

Let’s abstract this problem

$$\begin{tikzcd}
 & & dl_2 \overset{\sim}{:} DR_4 \Leftrightarrow DS_4 & & \\
 & \cdots \arrow[ur] & & \cdots \arrow[ul] & \\
DR_3 \arrow[ur] & & & & DS_3 \arrow[ul] \\
 & \cdots \arrow[ul] & & \cdots \arrow[ur] & \\
 & & dl \overset{\sim}{:} DR_2 \Leftrightarrow DS_2 \arrow[ul] \arrow[ur] & &
\end{tikzcd}$$

$$\begin{tikzcd}
 & & R(s_3 , t_3) & & \\
 & \cdots \arrow[ur] & & \cdots \arrow[ul] & \\
s_2 \arrow[ur] & & & & t_2 \arrow[ul] \\
 & \cdots \arrow[ul] & & \cdots \arrow[ur] & \\
 & & R(s_1 , t_1) \arrow[ul] \arrow[ur] & &
\end{tikzcd}$$

Let’s abstract this problem

$$\begin{tikzcd}
 & & dl_2 \overset{\sim}{:} DR_4 \Leftrightarrow DS_4 & & \\
 & \dots \arrow[ur] & & \dots \arrow[ul] & \\
DR_3 \arrow[ur] & & & & DS_3 \arrow[ul] \\
 & \dots \arrow[ul] & & \dots \arrow[ur] & \\
 & & dl \overset{\sim}{:} DR_2 \Leftrightarrow DS_2 \arrow[ul] \arrow[ur] & &
\end{tikzcd}$$

$$\begin{tikzcd}
 & & R(s_3 , t_3) & & \\
 & \dots \arrow[ur] & & \dots \arrow[ul] & \\
s_2 \arrow[ur] & & & & t_2 \arrow[ul] \\
 & \dots \arrow[ul] & & \dots \arrow[ur] & \\
 & & R(s_1 , t_1) \arrow[ul] \arrow[ur] & &
\end{tikzcd}$$

$R(DR,DS)$ if there exists a lens dl' such that $dl' \overset{\sim}{:} DR \Leftrightarrow DS$, and dl is equivalent to dl'

R-Confluence

If $R(s_1,t_1)$ and $s_1 \to^* s_2$ and $t_1 \to^* t_2$
Then there exists s_3 and t_3 such that

- $s_2 \to^* s_3$
- $t_2 \to^* t_3$
- $R(s_3,t_3)$

$$\begin{tikzcd}
 & & R(s_3 , t_3) & & \\
 & \cdots \arrow[ur] & & \cdots \arrow[ul] & \\
s_2 \arrow[ur] & & & & t_2 \arrow[ul] \\
 & \cdots \arrow[ul] & & \cdots \arrow[ur] & \\
 & & R(s_1 , t_1) \arrow[ul] \arrow[ur] & &
\end{tikzcd}$$

Sufficient Condition?

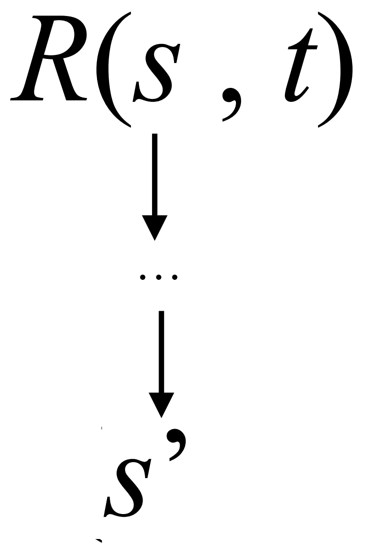
If → is confluent, and $(R, \to^*)$ is bisimilar,
Then → is R-confluent

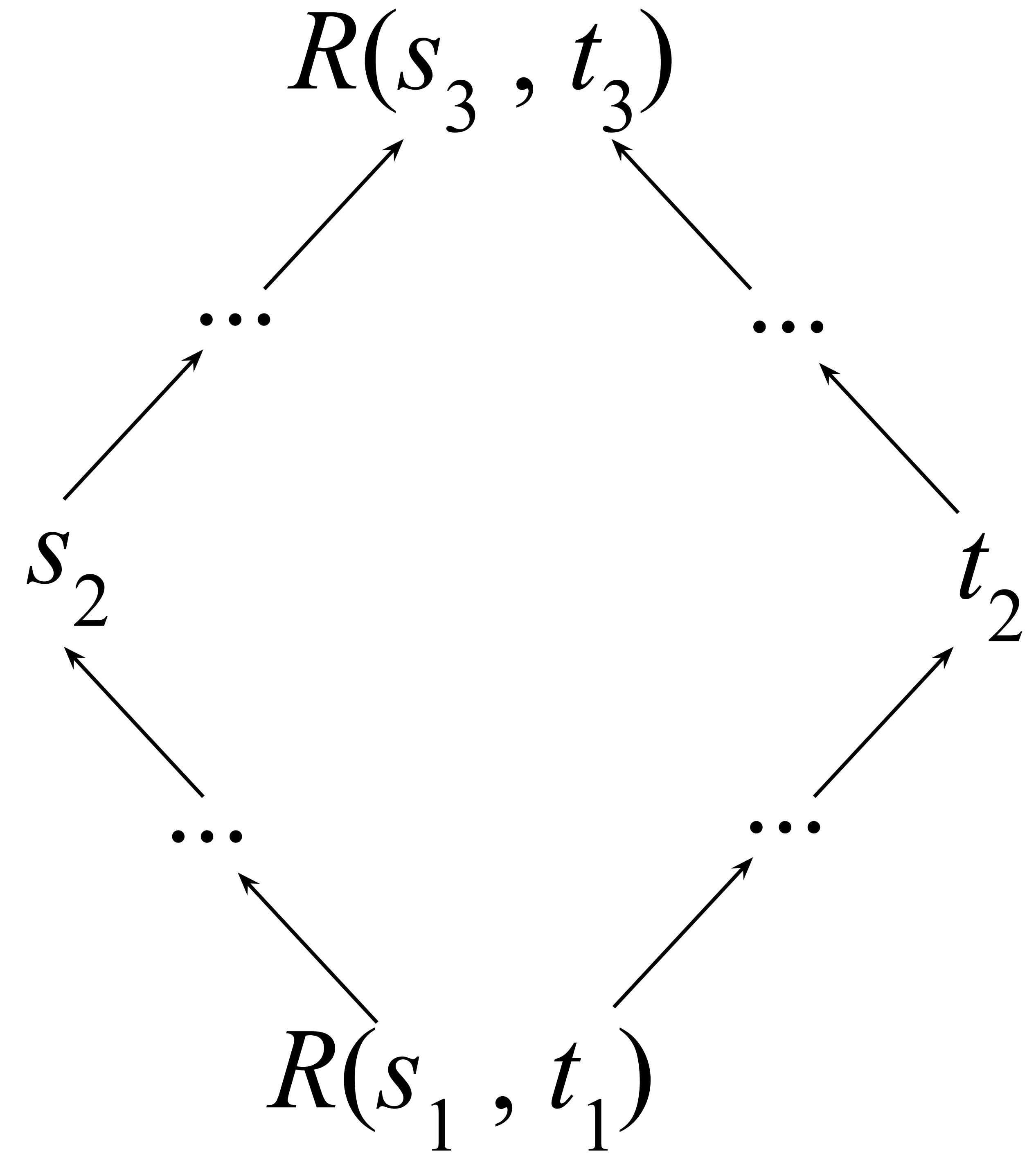
$$\begin{tikzcd}
 & & R(s_3\ ,\ t_3) & & \\
 & \cdots \arrow[ur] & & \cdots \arrow[ul] & \\
s_2 \arrow[ur] & & & & t_2 \arrow[ul] \\
 & \cdots \arrow[ul] & & \cdots \arrow[ur] & \\
 & & R(s_1\ ,\ t_1) \arrow[ul] \arrow[ur] & &
\end{tikzcd}$$

Sufficient Condition?

If → is confluent, and $(R, \rightarrow^*)$ is bisimilar,
Then → is R-confluent

$(R, \rightarrow^*)$ is bisimilar if, for all s,t where $R(s,t)$
- If $s \rightarrow^* s'$, then there exists t' such that $t \rightarrow^* t'$ and $R(s',t')$
- If $t \rightarrow^* t'$, then there exists s' such that $s \rightarrow^* s'$ and $R(s',t')$





Sufficient Condition?

If → is confluent, and $(R, \rightarrow^*)$ is bisimilar,
Then → is R-confluent

$(R, \rightarrow^*)$ is bisimilar if, for all s,t where $R(s,t)$
- If $s \rightarrow^* s'$, then there exists t' such that
  $t \rightarrow^* t'$ and $R(s',t')$
- If $t \rightarrow^* t'$, then there exists s' such that
  $s \rightarrow^* s'$ and $R(s',t')$

$$\begin{tikzcd}[column sep=small] R(s \arrow[d] & {,\ t)} \arrow[d] \\ \cdots \arrow[d] & \cdots \arrow[d] \\ R(s' & {,\ t')} \end{tikzcd}$$

$$\begin{tikzcd} & & R(s_3\ ,\ t_3) & & \\ & \cdots \arrow[ur] & & \cdots \arrow[ul] & \\ s_2 \arrow[ur] & & & & t_2 \arrow[ul] \\ & \cdots \arrow[ul] & & \cdots \arrow[ur] & \\ & & R(s_1\ ,\ t_1) \arrow[ul] \arrow[ur] & & \end{tikzcd}$$

Sufficient Condition?

If → is confluent, and (R,→*) is bisimilar,
Then → is R-confluent

(R,→*) is bisimilar if, for all s,t where $R(s,t)$
- If s →* s’, then there exists t’ such that
  t →* t’ and $R(s’,t’)$
- If t →* t’, then there exists s’ such that
  s →* s’ and $R(s’,t’)$

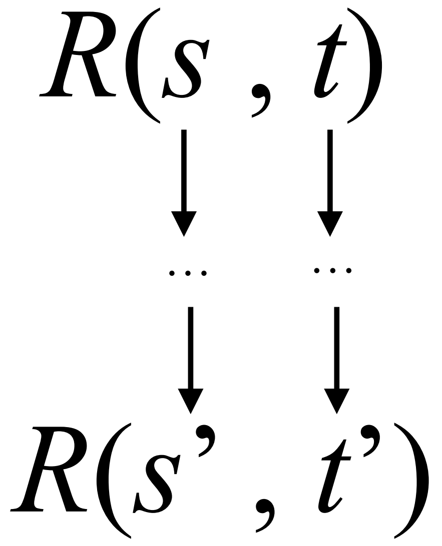
$$\begin{tikzcd}
R(s\ ,\ t) \arrow[d] \\
\cdots \arrow[d] \\
t'
\end{tikzcd}$$

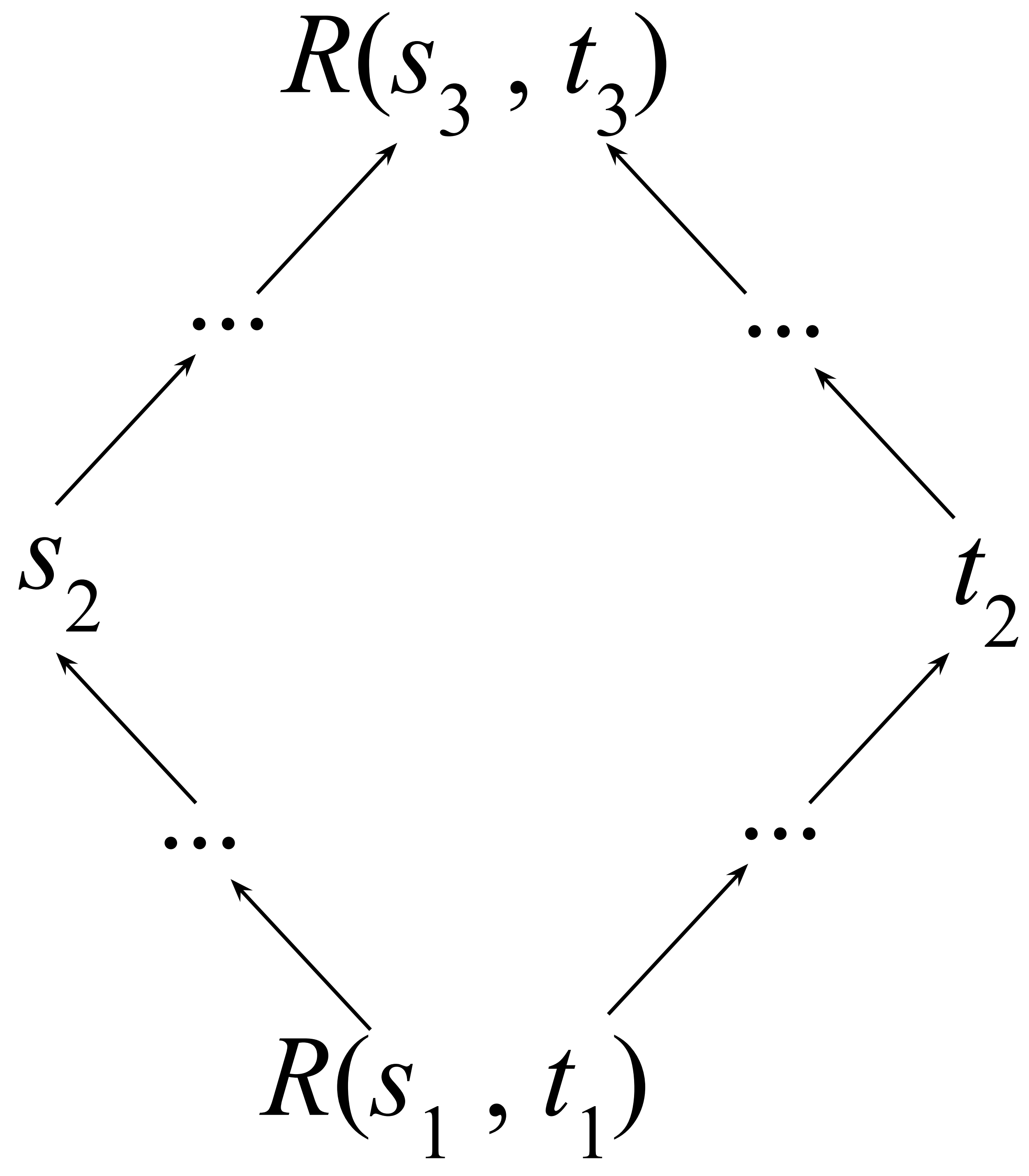
$$\begin{tikzcd}
 & & R(s_3\ ,\ t_3) & & \\
 & \cdots \arrow[ur] & & \cdots \arrow[ul] & \\
s_2 \arrow[ur] & & & & t_2 \arrow[ul] \\
 & \cdots \arrow[ul] & & \cdots \arrow[ur] & \\
 & & R(s_1\ ,\ t_1) \arrow[ul] \arrow[ur] & &
\end{tikzcd}$$

Sufficient Condition?

If → is confluent, and $(R, \to^*)$ is bisimilar,
Then → is R-confluent

$(R, \to^*)$ is bisimilar if, for all s,t where $R(s,t)$
- If $s \to^* s'$, then there exists t' such that $t \to^* t'$ and $R(s',t')$
- If $t \to^* t'$, then there exists s' such that $s \to^* s'$ and $R(s',t')$





Sufficiency Proof

If → is confluent, and (R,→*) is bisimilar,
Then → is R-confluent

By induction on the length of the reduction of s1 →* s2

$$\begin{tikzcd}
 & & R(s_3 , t_3) & & \\
 & \cdots \arrow[ur] & & \cdots \arrow[ul] & \\
s_2 \arrow[ur] & & & & t_2 \arrow[ul] \\
 & \cdots \arrow[ul] & & \cdots \arrow[ur] & \\
 & & R(s_1 , t_1) \arrow[ul] \arrow[ur] & &
\end{tikzcd}$$

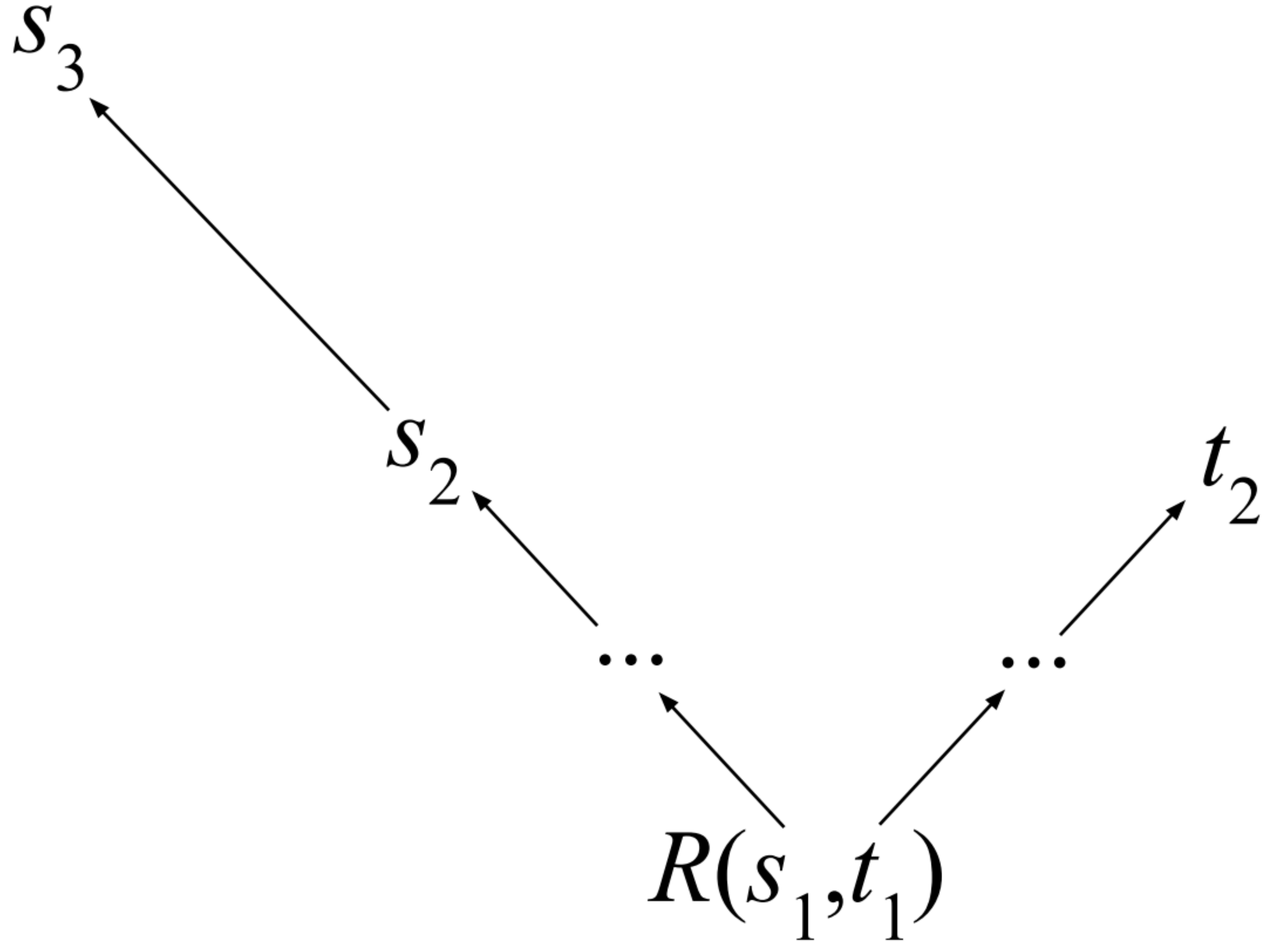
Sufficiency Proof Base Case

$$\begin{tikzcd}
R(s\ ,\ t) \arrow[d] \\
\cdots \arrow[d] \\
t'
\end{tikzcd}$$

Sufficiency Proof Base Case

$$\begin{tikzcd}[column sep=tiny] R(& s \arrow[d] & , & t \arrow[d] &) \\ & \cdots \arrow[d] & & \cdots \arrow[d] & \\ R(& s' & , & t' &) \end{tikzcd}$$

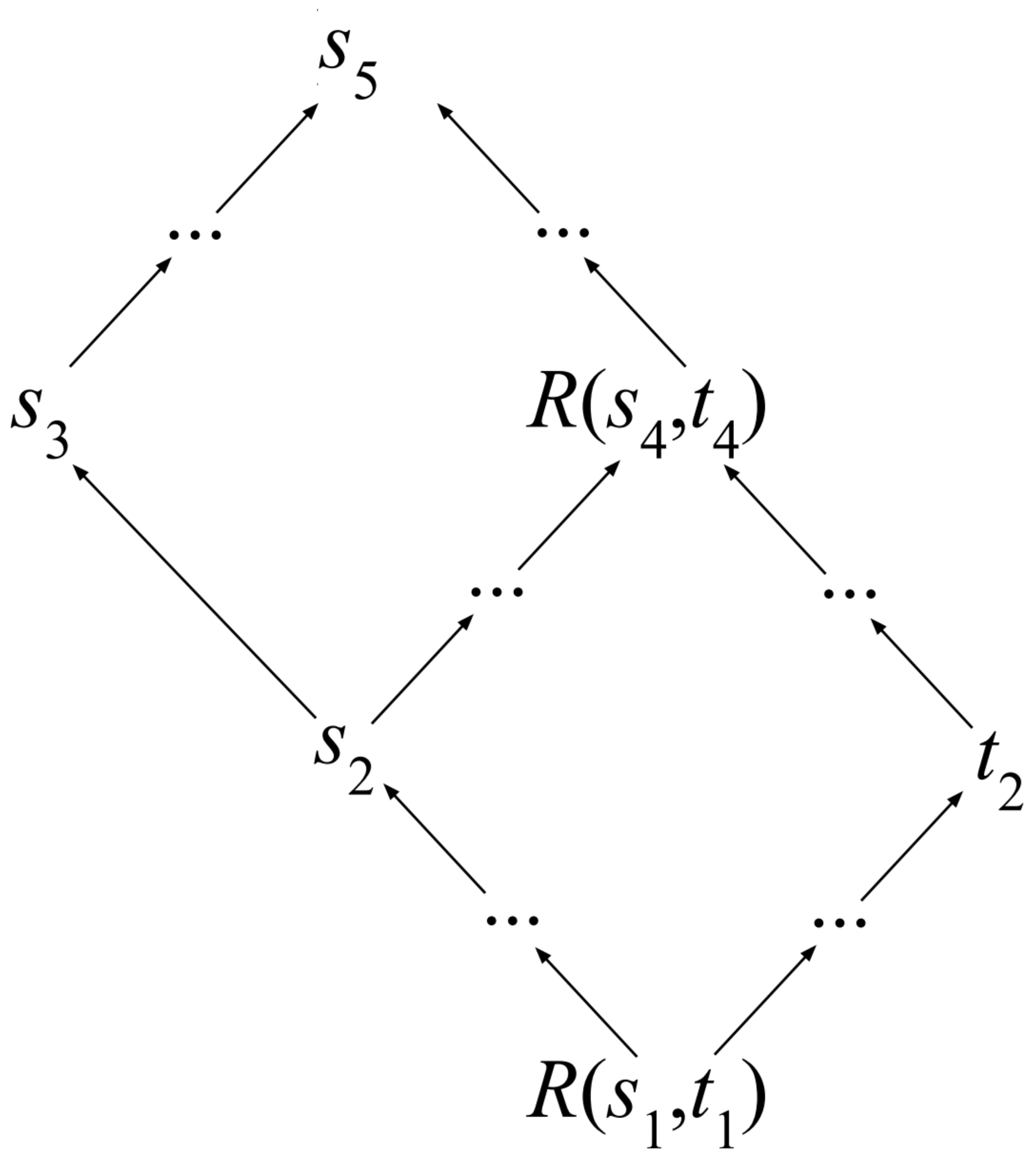
Sufficiency Proof Inductive Step



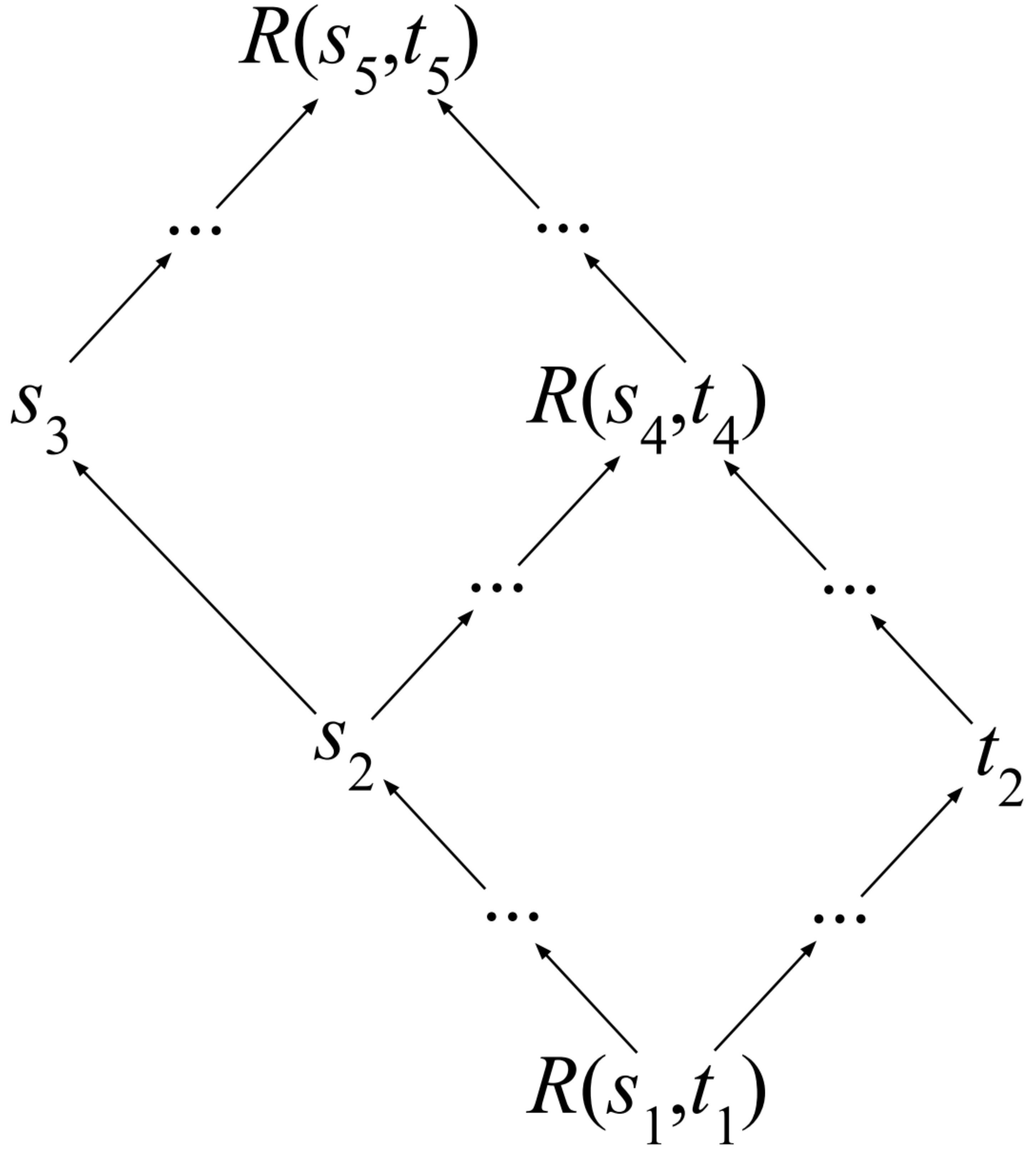
Sufficiency Proof Inductive Step

$$\begin{tikzcd}
s_3 & & & R(s_4,t_4) & & \\
& & \cdots \arrow[ur] & & \cdots \arrow[ul] & \\
& s_2 \arrow[uul] \arrow[ur] & & & & t_2 \arrow[ul] \\
& & \cdots \arrow[ul] & & \cdots \arrow[ur] & \\
& & & R(s_1,t_1) \arrow[ul] \arrow[ur] & &
\end{tikzcd}$$

Sufficiency Proof Inductive Step



Sufficiency Proof Inductive Step



Case: Type Equivalence

$$\begin{tikzcd}
 & & DR_3 & & & & & DS_3 & & \\
 & \cdots \arrow[ur] & & \cdots \arrow[ul] & & & \cdots \arrow[ur] & & \cdots \arrow[ul] & \\
DR_1' \arrow[ur] & & & & dl \overset{\sim}{:} DR_2 \arrow[ul] \arrow[r, phantom, "\Leftrightarrow"] & DS_2 \arrow[ur] & & & & DS_1' \arrow[ul] \\
 & & & \cdots \arrow[ur] & & & \cdots \arrow[ul] & & & \\
 & & DR_1 \arrow[uull, Rrightarrow, no head] \arrow[ur] & & & & & DS_1 \arrow[ul] \arrow[uurr, Rrightarrow, no head] & &
\end{tikzcd}$$

Case: Type Equivalence

$$\begin{tikzcd}
 & & DR_3 & & & & & & DS_3 & & \\
 & \cdots \arrow[ur] & & \cdots \arrow[ul] & & & & \cdots \arrow[ur] & & \cdots \arrow[ul] & \\
DR_1' \arrow[ur] \arrow[ddrr, Rrightarrow, no head] & & & & dl \mathrel{\tilde{:}} DR_2 \arrow[ul] & \Leftrightarrow & DS_2 \arrow[ur] & & & & DS_1' \arrow[ul] \arrow[ddll, Rrightarrow, no head] \\
 & & & \cdots \arrow[ur] & & & & \cdots \arrow[ul] & & & \\
 & & DR_1 \arrow[ur] & & & & & & DS_1 \arrow[ul] & &
\end{tikzcd}$$

$$\begin{tikzcd}
 & & & & dl_2 \mathrel{\tilde{:}} DR_4 & \Leftrightarrow & DS_4 & & & & \\
 & & & \cdots \arrow[ur] & & & & \cdots \arrow[ul] & & & \\
 & & DR_3 \arrow[ur] & & & & & & DS_3 \arrow[ul] & & \\
 & \cdots \arrow[ur] & & \cdots \arrow[ul] & & & & \cdots \arrow[ur] & & \cdots \arrow[ul] & \\
DR_1' \arrow[ur] \arrow[ddrr, Rrightarrow, no head] & & & & dl \mathrel{\tilde{:}} DR_2 \arrow[ul] & \Leftrightarrow & DS_2 \arrow[ur] & & & & DS_1' \arrow[ul] \arrow[ddll, Rrightarrow, no head] \\
 & & & \cdots \arrow[ur] & & & & \cdots \arrow[ul] & & & \\
 & & DR_1 \arrow[ur] & & & & & & DS_1 \arrow[ul] & &
\end{tikzcd}$$

Case: Type Equivalence

$R(DR,DS)$ if there exists a lens dl' such that $dl' : DR \Leftrightarrow DS$, and dl is equivalent to dl'

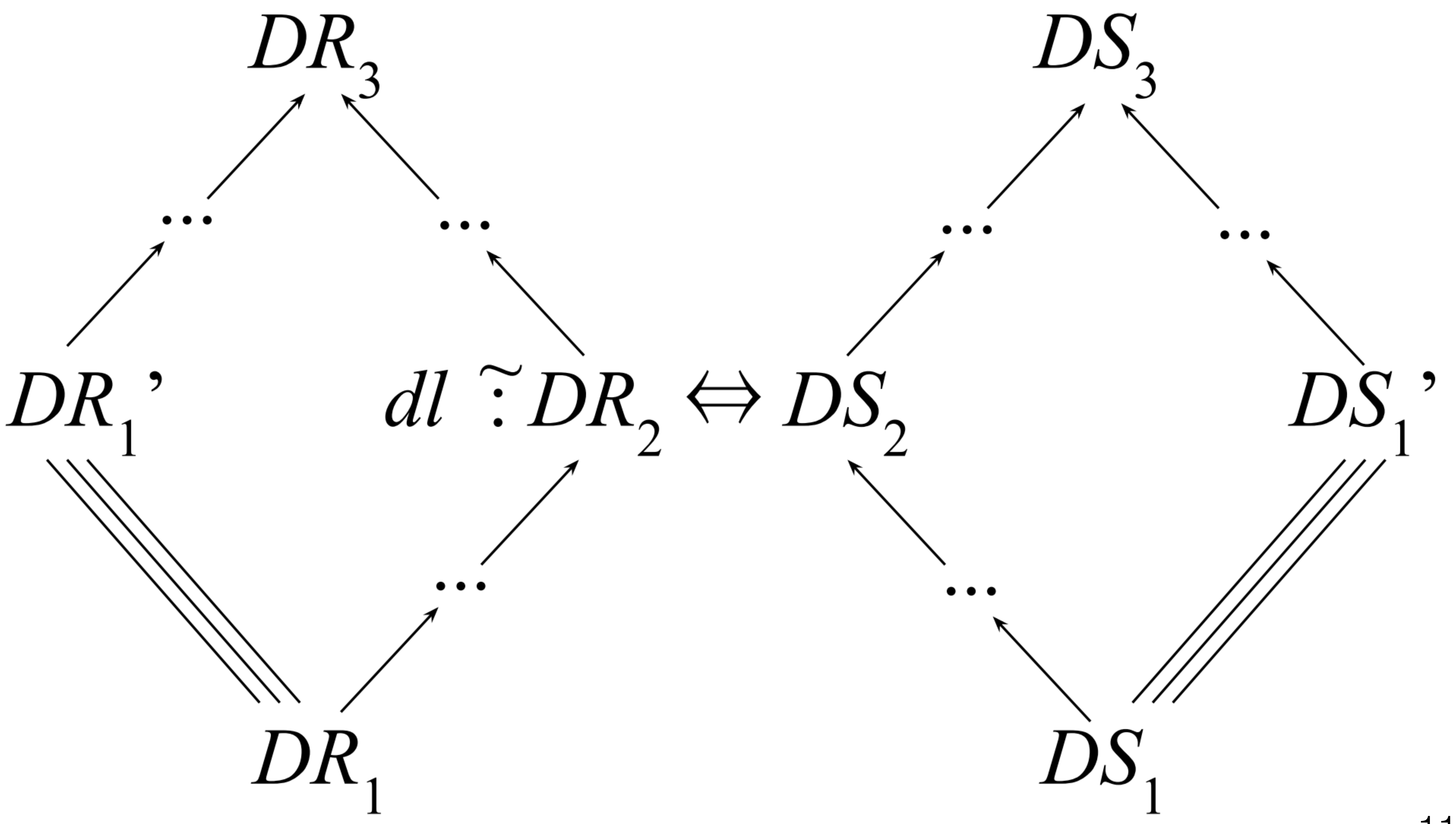
$$\begin{tikzcd}
 & & DR_3 & & & & DS_3 & & \\
 & \cdots \arrow[ur] & & \cdots \arrow[ul] & & \cdots \arrow[ur] & & \cdots \arrow[ul] & \\
DR_1' \arrow[ur] \arrow[ddrr, equal] & & & & dl \mathrel{\tilde{:}} DR_2 \Leftrightarrow DS_2 \arrow[ul] \arrow[ur] & & & & DS_1' \arrow[ul] \arrow[ddll, equal] \\
 & & & \cdots \arrow[ur] & & \cdots \arrow[ul] & & & \\
 & & DR_1 \arrow[ur] & & & & DS_1 \arrow[ul] & &
\end{tikzcd}$$

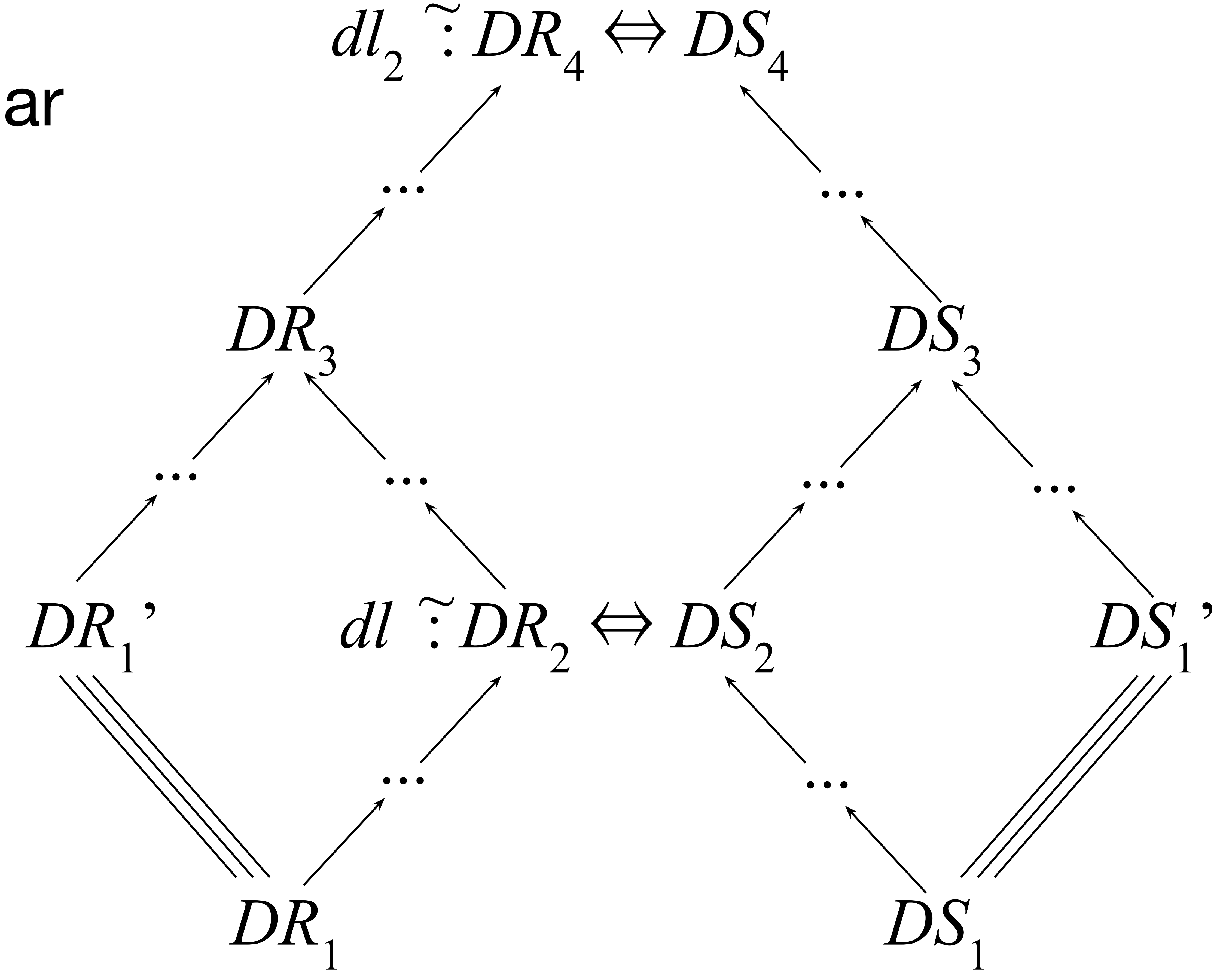
$$\begin{tikzcd}
 & & & & dl_2 \mathrel{\tilde{:}} DR_4 \Leftrightarrow DS_4 & & & & \\
 & & & \cdots \arrow[ur] & & \cdots \arrow[ul] & & & \\
 & & DR_3 \arrow[ur] & & & & DS_3 \arrow[ul] & & \\
 & \cdots \arrow[ur] & & \cdots \arrow[ul] & & \cdots \arrow[ur] & & \cdots \arrow[ul] & \\
DR_1' \arrow[ur] \arrow[ddrr, equal] & & & & dl \mathrel{\tilde{:}} DR_2 \Leftrightarrow DS_2 \arrow[ul] \arrow[ur] & & & & DS_1' \arrow[ul] \arrow[ddll, equal] \\
 & & & \cdots \arrow[ur] & & \cdots \arrow[ul] & & & \\
 & & DR_1 \arrow[ur] & & & & DS_1 \arrow[ul] & &
\end{tikzcd}$$

Case: Type Equivalence

R(*DR*,*DS*) if there exists a lens *dl*' such that *dl*' : $DR \Leftrightarrow DS$, and *dl* is equivalent to *dl*'

We prove → is confluent, and (*R*,→*) is bisimilar





Related Notions

Confluence Modulo Equivalence

$$\begin{tikzcd}
 & & s_3 \sim t_3 & & \\
 & \cdots \arrow[ur] & & \cdots \arrow[ul] & \\
s_2 \arrow[ur] & & & & t_2 \arrow[ul] \\
 & \cdots \arrow[ul] & & \cdots \arrow[ur] & \\
 & & s_1 \sim t_1 \arrow[ul] \arrow[ur] & &
\end{tikzcd}$$

$$\begin{tikzcd}
 & & R(s_3\,,\, t_3) & & \\
 & \cdots \arrow[ur] & & \cdots \arrow[ul] & \\
s_2 \arrow[ur] & & & & t_2 \arrow[ul] \\
 & \cdots \arrow[ul] & & \cdots \arrow[ur] & \\
 & & R(s_1\,,\, t_1) \arrow[ul] \arrow[ur] & &
\end{tikzcd}$$

[Huet 1980]

Related Notions

Confluence Modulo Equivalence

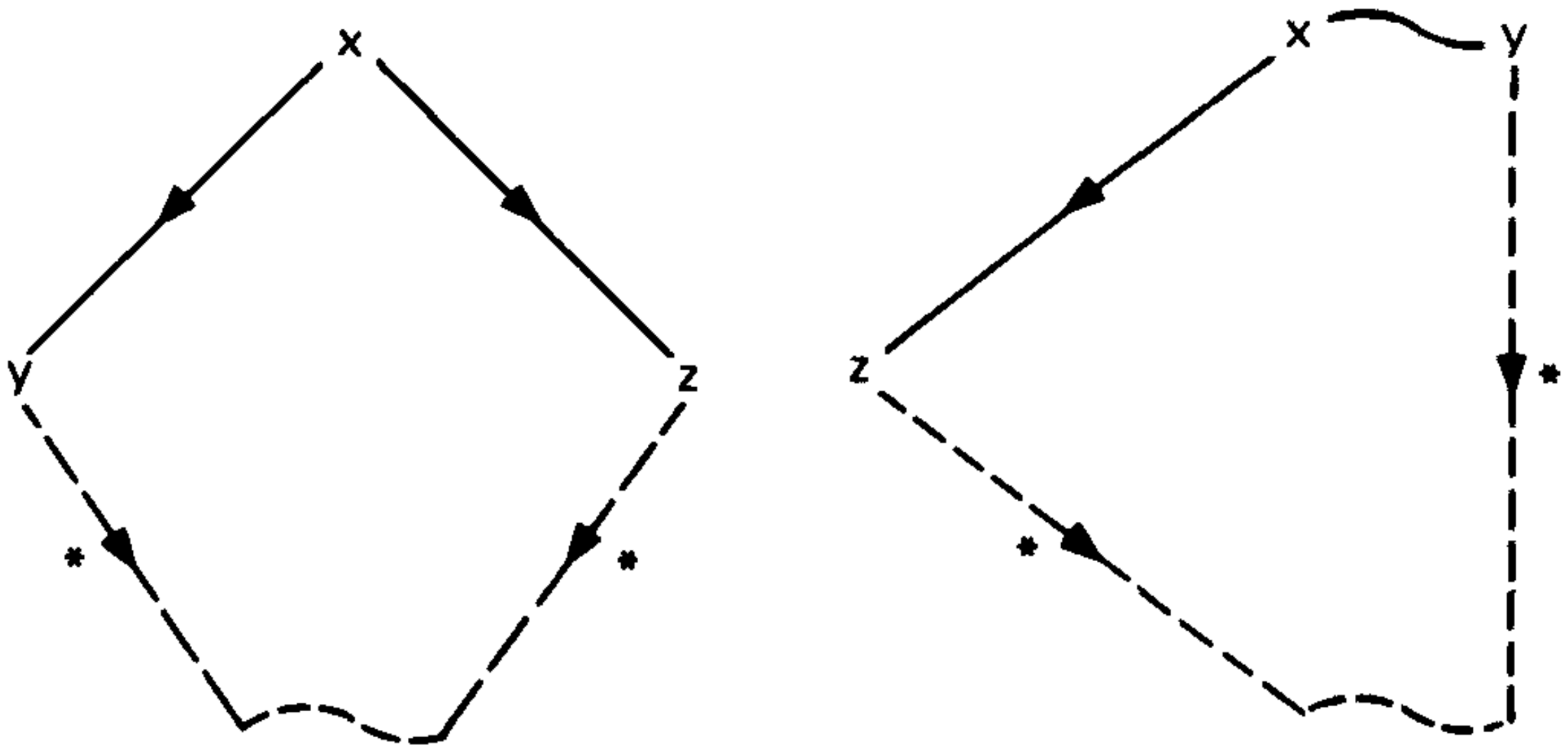
$$\begin{tikzcd}
 & & s_3 \arrow[r, phantom, "\sim"] & {} & t_3 & & \\
 & \cdots \arrow[ur] & & & & \cdots \arrow[ul] & \\
s_2 \arrow[ur] & & & & & & t_2 \arrow[ul] \\
 & \cdots \arrow[ul] & & & & \cdots \arrow[ur] & \\
 & & s_1 \arrow[ul] \arrow[r, phantom, "\sim"] & {} & t_1 \arrow[ur] & &
\end{tikzcd}$$

Confluence

[Huet 1980]

Related Notions

Confluence Modulo Equivalence



Confluence

Bisimilarity

[Huet 1980]

Related Notions

Bisimilarity

Related Notions

Bisimilarity

Common in state transition systems

Related Notions

Bisimilarity

Common in state transition systems

Related to commuting rewrites

[Toyama 1988]

Related Notions

Bisimilarity

Common in state transition systems

Related to commuting rewrites

$$\begin{tikzcd} R(s \arrow[d] & , & t) \arrow[d] \\ \cdots \arrow[d] & & \cdots \arrow[d] \\ R(s' & , & t') \end{tikzcd}$$

[Toyama 1988]

Related Notions

Bisimilarity

Common in state transition systems

Related to commuting rewrites

$$\begin{tikzcd}
s \arrow[r, "R"] \arrow[d] & \cdots \arrow[r, "R"] & t \arrow[d] \\
\cdots \arrow[d] & & \cdots \arrow[d] \\
s' \arrow[r, "R"] & \cdots \arrow[r, "R"] & t'
\end{tikzcd}$$

$$\begin{tikzcd}
R(s \arrow[d] & , \; t) \arrow[d] \\
\cdots \arrow[d] & \cdots \arrow[d] \\
R(s' & , \; t')
\end{tikzcd}$$

[Toyama 1988]

Conclusions

We can synthesize lenses, by synthesizing in an alternative form

This alternative form is equivalent

- Proof requires a confluence-like property, R-confluence
- Confluence + Bisimilarity with R ⇒ R-confluence