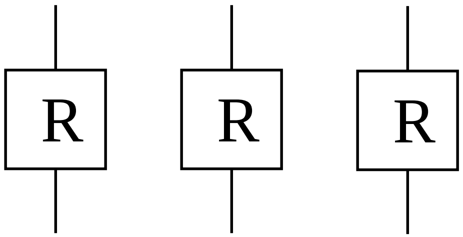
Summary of last lecture

- common composition patterns
  - converse, conjugate
  - repeated series composition, repeated parallel composition
- types
  - primitive and composite types
  - types for primitive and composite blocks
- triangular-shaped architectures
  - tree-shaped array
  - triangular array
- grid components and combinators
  - beside and below
  - transposed conjugate
  - row and column

Reasoning and specialisation

- key features in Ruby
  - parametrisation: capture collection of related designs, e.g. R^n
  - composition: build design from components
  - reasoning: develop design rigorously using simple maths
  - specialisation: produce special cases of general design patterns
- parametrisation
  - make descriptions general: capture patterns of composition
  - get different designs: instantiate parameters with different values
  - can generate designs with different performance trade-offs
- inductive definition
  - base case: case 0 or 1
  - induction case: assume case n, derive case (n+1)
  - use binary operator, e.g. (;) or [,] or ($\leftrightarrow$) to construct case (n+1)
  - key: check the **type** of the interface variables – brackets!

Recap: map - repeated parallel composition

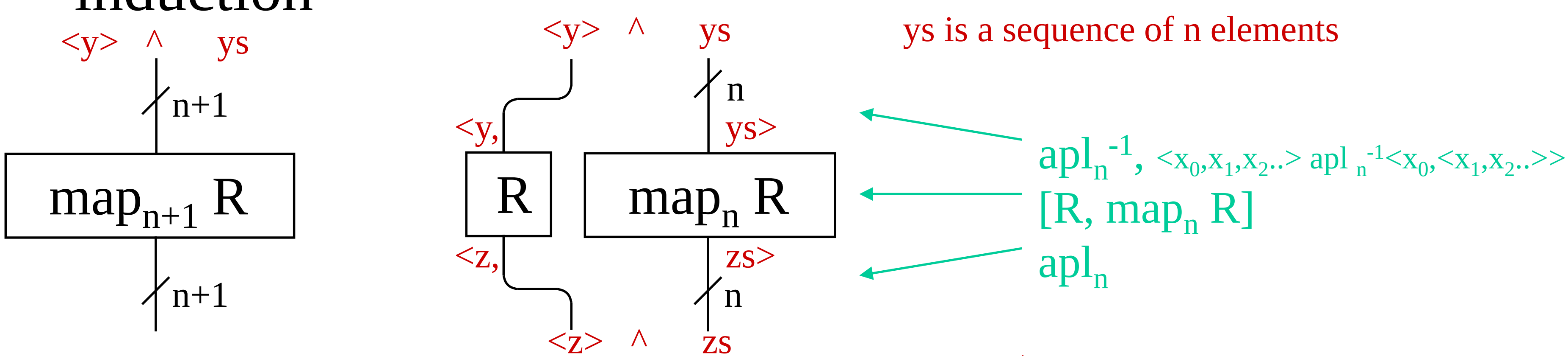
- 

  $\mathrm{map}_3\ R\ =\ [R, R, R]$

- recursive description:

  base case: $\mathrm{map}_0\ R = []$, where <> [] <>

- induction



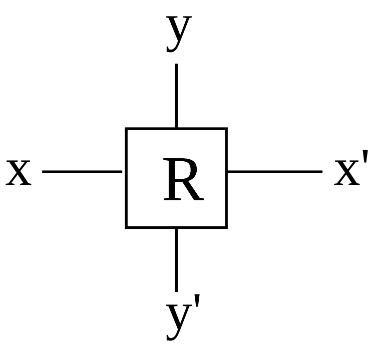
<y> ^ ys <y, ys> <z, zs> <z> ^ zs

$$\mathrm{map}_{n+1}\ R\ =\ \mathrm{apl}_n^{-1} ; [R, \mathrm{map}_n\ R] ; \mathrm{apl}_n$$

$$=\ [R, \mathrm{map}_n\ R] \setminus \mathrm{apl}_n \text{ where }\ Q \setminus P\ =\ P^{-1} ; Q ; P$$

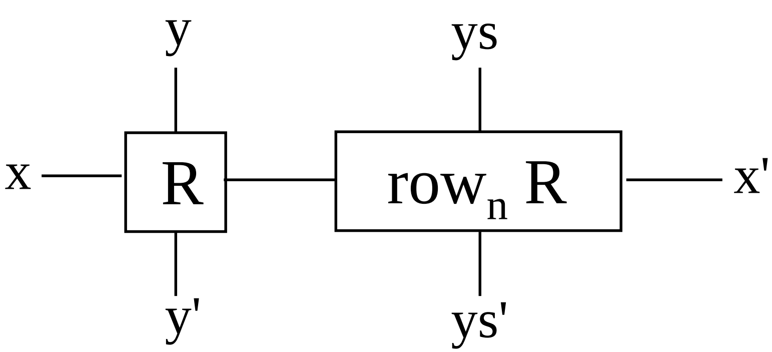
sequence concatenation: <a,b> ^ <c,d,e> = <a,b,c,d,e>

Recap: row – repeated beside composition

- 

<x, <y>> <x, y> <y' x'> <<y'>, x'>

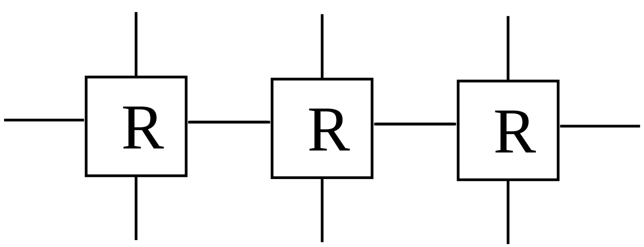
$$\mathrm{row}_1\ R \stackrel{def}{=} \mathrm{snd}\ [-]^{-1} ; R ; \mathrm{fst}\ [-]$$
$$= R \setminus\setminus \mathrm{fst}\ [-]$$

- 

sequence concatenation

<x, <y> ^ ys> <x, <y, ys>>

$$\mathrm{row}_{n+1}\ R \stackrel{def}{=} \mathrm{snd}\ \mathrm{apl}_n^{-1} ; R \leftrightarrow (\mathrm{row}_n\ R) ; \mathrm{fst}\ \mathrm{apl}_n$$
$$= (R \leftrightarrow \mathrm{row}_n\ R) \setminus\setminus \mathrm{fst}\ \mathrm{apl}_n$$

- 

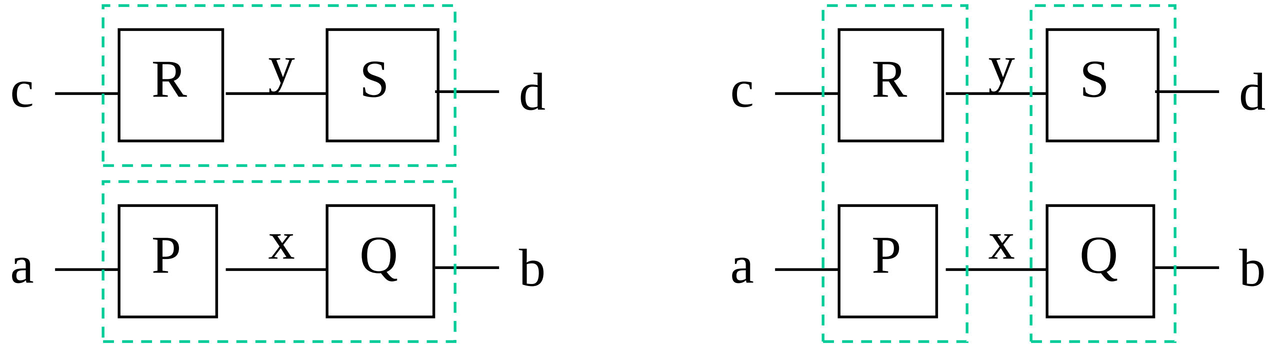
$\mathrm{row}_3\ R$

Reasoning about Ruby descriptions

- design steps: idea, description, diagram, simulation, proof
- why reasoning?
  - library designs: require high confidence of correctness
  - may also improve efficiency e.g. preconditions for transforms
  - understand design space, automate design development
- proof techniques
  - pointwise: introduce data objects of the right shape
  - pointfree: use algebraic laws to transform designs
  - pointfree: verify algebraic laws, possibly using induction
- maths: simple logic e.g. $\exists$, equal substitution, induction
- hints
  - get types right
  - check correctness using simulator
  - proofs can be guided by diagrams, but independent of them

Algebraic law: pointwise proof

- [(P ; Q), (R ; S)]   =   [P, R] ; [Q, S]



- a (P ; Q) b ⇔ (∃x . a P x ∧ x Q b) ***…there exists*** x ***such that*** a P x ***and***…
- proof (*pointwise* since it involves introducing a, b, c and d)

  <a, c> LHS <b, d>

  ⇔ a (P ; Q) b ∧ c (R ; S) d   (def. //el)

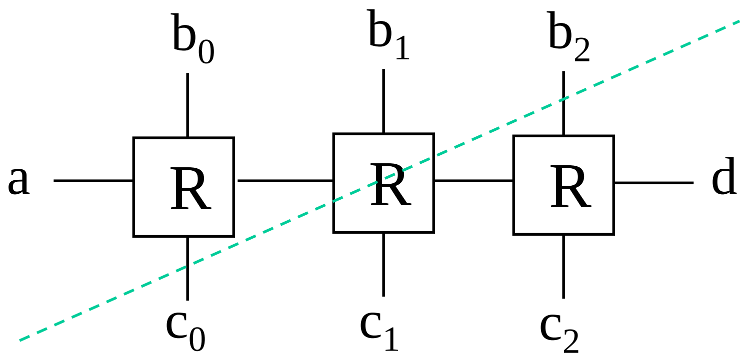
  ⇔ (∃x . a P x ∧ x Q b) ∧ (∃y . c R y ∧ y S d)   (def. ;)

  ⇔ ∃x,y . ((a P x) ∧ (c R y)) ∧ ((x Q b) ∧ (y S d))   (maths)

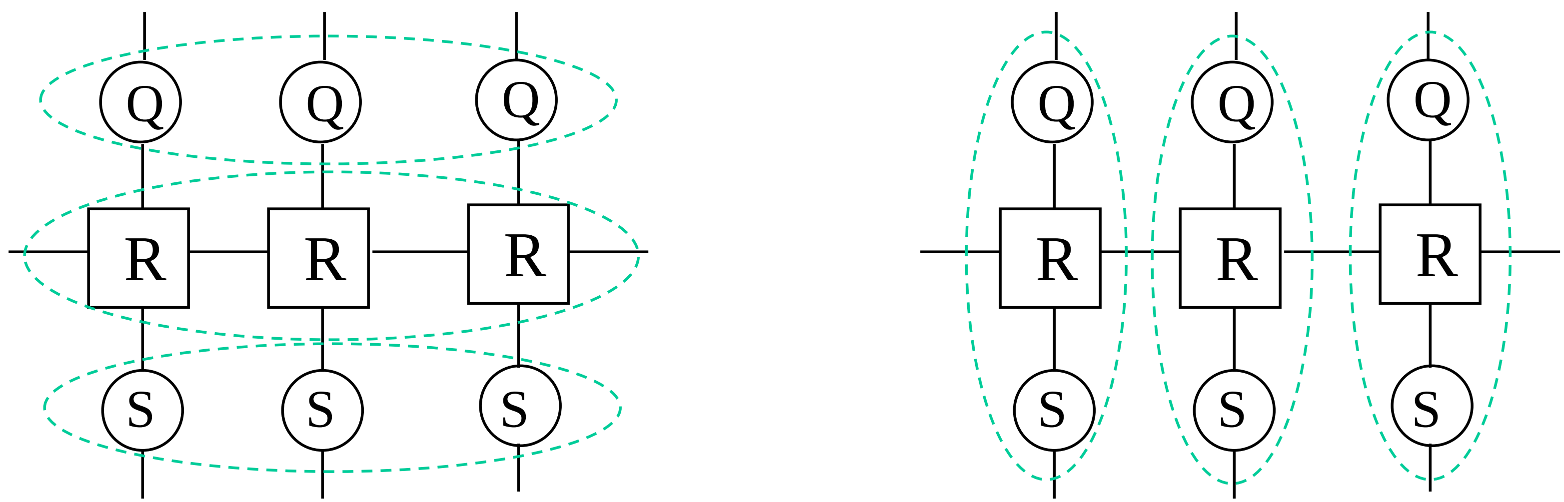
  ⇔ ∃x,y . <a, c> [P, R] <x, y> ∧ <x, y> [Q, S] <b, d>   (def. //el)

  ⇔ <a, c> RHS <b, d>

Algebraic laws for row and column

- 

  $(\text{row}_n\ R)^{-1} = \text{col}_n\ (R^{-1})$

- 

$$\text{snd}\ (\text{map}_n\ Q)\ ;\ \text{row}_n\ R\ ;\ \text{fst}\ (\text{map}_n\ S) \quad = \quad \text{row}_n\ (\text{snd}\ Q\ ;\ R\ ;\ \text{fst}\ S)$$

Pointfree proof of a law: by induction

- to show:

  $\text{snd}\ (\text{map}_n\ Q);\ \text{row}_n\ R;\ \text{fst}\ (\text{map}_n\ S) = \text{row}_n\ (\text{snd}\ Q;\ R;\ \text{fst}\ S)$ (1)

- base case definition: map and row:

  $\text{map}_0\ R = [],\quad \text{row}_0\ R = \text{snd}\ []\ ;\ \text{swap}\ ;\ \text{fst}\ [] = \text{swap} \backslash\backslash (\text{fst}\ [])$ (2)

- base case: LHS of (1)

  $= \{$ def. of $\text{map}_0\ Q,\ \text{row}_0\ R$ $\}$

  $\text{snd}\ []\ ;\ (\text{snd}\ []\ ;\ \text{swap}\ ;\ \text{fst}\ []);\ \text{fst}\ []$

  $= \{\ \text{snd}\ A\ ;\ \text{snd}\ B = \text{snd}\ (A;B)$ and $[];[] = []\ \}$

  $\text{snd}\ []\ ;\ \text{swap}\ ;\ \text{fst}\ []$

  $= \{$ def. of $\text{row}_0\ P$ $\}$

  RHS of (1)

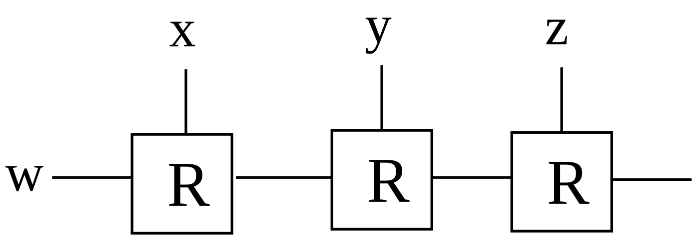
Proof by induction (induction case)

- induction case: need
  - snd [A, B] ; C $\leftrightarrow$ D ; fst [E, F]
    = (snd A ; C ; fst E) $\leftrightarrow$ (snd B ; D ; fst F) (3)

- snd (map_{n+1} Q); row_{n+1} R; fst (map_{n+1} S) ……. *LHS of* (1)
  = { def. of map_{n+1} Q, row_{n+1} R }
  snd (apl_n^{-1} ; [Q, map_n Q] ; apl_n) ;
  snd apl_n^{-1} ; (R $\leftrightarrow$ row_n R) ; fst apl_n ;
  fst (apl_n^{-1} ; [S, map_n S] ; apl_n)
  = { apl_n ; apl_n^{-1} = id, and (3) }
  snd apl_n^{-1} ; (snd Q ; R ; fst S)
  $\leftrightarrow$ (snd map_n Q ; row_n R ; fst map_n S) ; fst apl_n
  = { induction hypothesis }
  snd apl_n^{-1} ; (snd Q ; R ; fst S)
  $\leftrightarrow$ (row_n (snd Q ; R ; fst S)) ; fst apl_n
  = { def. of row_{n+1} P }
  row_{n+1} (snd Q; R; fst S) ……. *RHS of* (1)

Reduction: specialisation of row and column

- drop the bottom-side connections of a row



$$\mathrm{rdl}_n\, R \;=\; \mathrm{row}_n\, (R \,;\, \pi_2^{-1}) \,;\, \pi_2$$ left reduction

<w, <x, y, z>> (rdl_3 subtr) (((w -x) - y) -z) (also called *fold left*)

- drop the right-side connections of a column

$$\mathrm{rdr}_n\, R \;=\; \mathrm{col}_n\, (R \,;\, \pi_1^{-1}) \,;\, \pi_1$$ right reduction

<<w, x, y>, z> (rdr_3 subtr) (w - (x - (y - z)))

- rdl and rdr functions can also be defined inductively

Example of reduction: inner product

- specification:

$$z = \sum_{i<n} x_i \times y_i$$

- obvious design:

$$IP0_n = map_n \ mult \ ; \pi_2^{-1} \ ; fst\ 0 \ ; rdl_n \ add$$

- more regular design:

$$IP1_n = \pi_2^{-1} \ ; fst\ 0 \ ; rdl_n \ (snd \ mult \ ; add)$$

Recap: triangles

- $\Delta_n\ R\ =\ [R^0, R^1, R^2, ..., R^{n-1}]$
  $=\ \text{if}\ \ n=0\ \ \text{then}\ []$
  $\text{else}\ \ [\Delta_{n-1}\ R,\ \ R^{n-1}]\ \backslash\ apr_{n-1}$
- flipped triangle
  - $\Delta^{\sim}_n\ R\ =\ [R^{n-1}, R^{n-2}, ..., R^0]$
- $\Delta_1\ R\ =\ ?$
  $\Delta_2\ R\ =\ ?$
- example: polynomial evaluation $\qquad y = \sum_{i<m} a_i \times x^i$
  - polyeval: <sreal>$_m$ ~ sreal
  - polyeval $= \Delta_n\ (\text{mult}\ x)\ ;\ (apl_{n-1})^{-1}\ ;\ rdl_{(n-1)}\ \text{add}$