A model of conceptual bootstrapping in human cognition

Bonan Zhao, Christopher G. Lucas, Neil R. Bramley

journal club at HMC Lab, David Nagy

MAX PLANCK GESELLSCHAFT

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

Contents lists available at SciVerse ScienceDirect

Cognition

journal homepage: www.elsevier.com/locate/COGNIT

Bootstrapping in a language of thought: A formal model of numerical concept learning

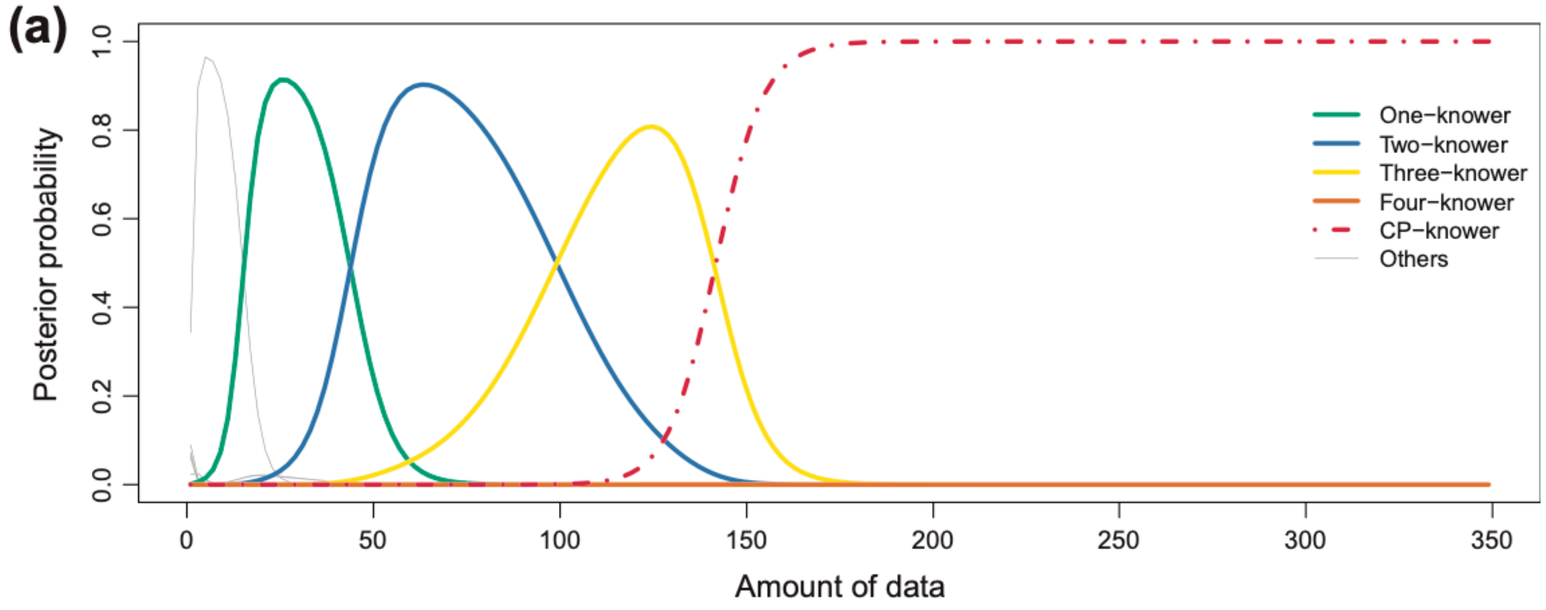
Steven T. Piantadosi [a,*], Joshua B. Tenenbaum [b], Noah D. Goodman [c]

[a] Department of Brain and Cognitive Sciences, University of Rochester, United States
[b] Department of Brain and Cognitive Sciences, MIT, United States
[c] Department of Psychology, Stanford University, United States

Primitive operations allowed in the LOT. All valid compositions of these primitives are potential hypotheses for the model.

Functions mapping sets to truth values	
(*singleton? X*)	Returns true iff the set *X* has exactly one element
(*doubleton? X*)	Returns true iff the set *X* has exactly two elements
(*tripleton? X*)	Returns true iff the set *X* has exactly three elements
Functions on sets	
(*set-difference X Y*)	Returns the set that results from removing *Y* from *X*
(*union X Y*)	Returns the union of sets *X* and *Y*
(*intersection X Y*)	Returns the intersect of sets *X* and *Y*
(*select X*)	Returns a set containing a single element from *X*
Logical functions	
(*and P Q*)	Returns TRUE if *P and Q* are both true
(*or P Q*)	Returns TRUE if either *P or Q* is true
(*not P*)	Returns TRUE iff *P* is false
(*if P X Y*)	Returns *X* iff *P* is true, *Y* otherwise
Functions on the counting routine	
(*next W*)	Returns the word after *W* in the counting routine
(*prev W*)	Returns the word before *W* in the counting routine
(*equal-word? W V*)	Returns TRUE if *W* and *V* are the same word
Recursion	
(*L S*)	Returns the result of evaluating the entire current lambda expression on set *S*

(Piantadosi et al., 2012)



One-knower

```
λ S . (if (singleton? S)
          "one"
          undef)
```

Two-knower

```
λ S . (if (singleton? S)
          "one"
          (if (doubleton? S)
              "two"
              undef))
```

Three-knower

```
λ S . (if (singleton? S)
          "one"
          (if (doubleton? S)
              "two"
              (if (tripleton? S)
                  "three"
                  undef))
```

CP-knower

```
λ S . (if (singleton? S)
          "one"
          (next (L (set-difference S
                                   (select S)))))
```

Singular-Plural

```
λ S . (if (singleton? S)
          "one"
          "two")
```

Mod-5

```
λ S . (if (or (singleton? S)
              (equal-word? (L (set-difference S)
                              (select S))
                           "five"))
          "one"
          (next (L (set-difference S
                                   (select S)))))
```

2-not-1-knower

```
λ S . (if (doubleton? S)
          "two"
          undef)
```

2N-knower

```
λ S . (if (singleton? S)
          "one"
          (next (next (L (set-difference S (select S))))))
```

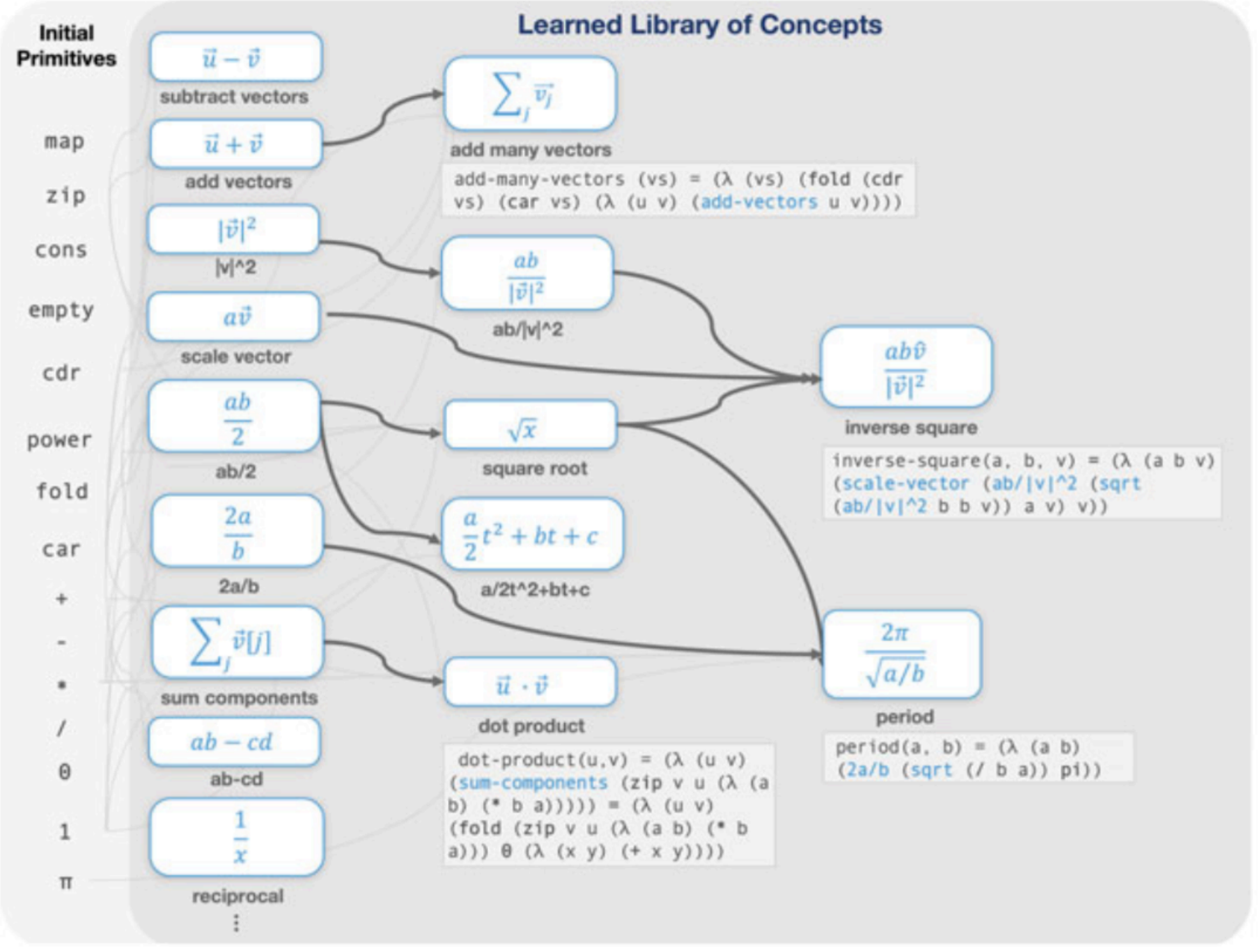
(Piantadosi et al., 2012)

D.S.: The numbers only go to million and ninety-nine.
Experimenter: What happens after million and ninety-nine?
D.S.: You go back to zero.
E: I start all over again? So, the numbers do have an end?
Or do the numbers go on and on?
D.S.: Well, everybody says numbers go on and on because you start over again with million and ninety-nine.

...

E: How about if I tell you that there is a number after that?
A million one hundred.
D.S.: Well, I wish there was a million and one hundred, but there isnt.

(Hartnett, 1991)



Discovered Physics Equations

Newton's Second Law

$$\vec{a} = \frac{1}{m}\sum_i \vec{F}_i$$

```
(scale-vector(reciprocal m)
(add-many-vectors Fs))
```

Parallel Resistors

$$R_{total} = \left(\Sigma_i \frac{1}{R_i}\right)^{-1}$$

```
(reciprocal (sum-components
(map (λ(r) (reciprocal r))
Rs)))
```

Work

$$U = \vec{F}\cdot\vec{d}$$

```
(dot-product F d)
```

Force in a Magnetic Field

$$|\vec{F}| = q|\vec{v}\times\vec{B}|$$

```
(* q (ab-cd v_x b_y v_y b_x))
```

Kinetic Energy

$$KE = \frac{1}{2}m|\vec{v}|^2$$

```
(ab/2 m (|v|^2 v))
```

Coulomb's Law

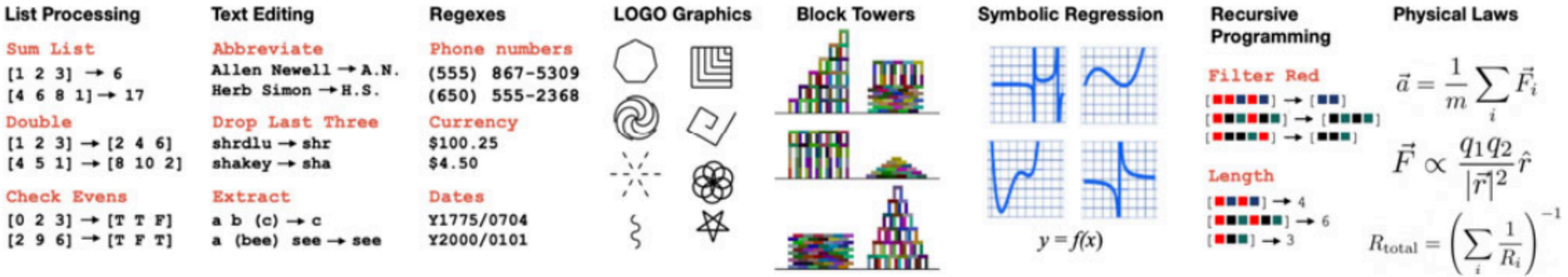
$$\vec{F} \propto \frac{q_1 q_2}{|\vec{r_1}-\vec{r_2}|^2}\widehat{r_1 - r_2}$$

```
(inverse-square q_1 q_2
(subtract-vectors r_1 r_2))
```

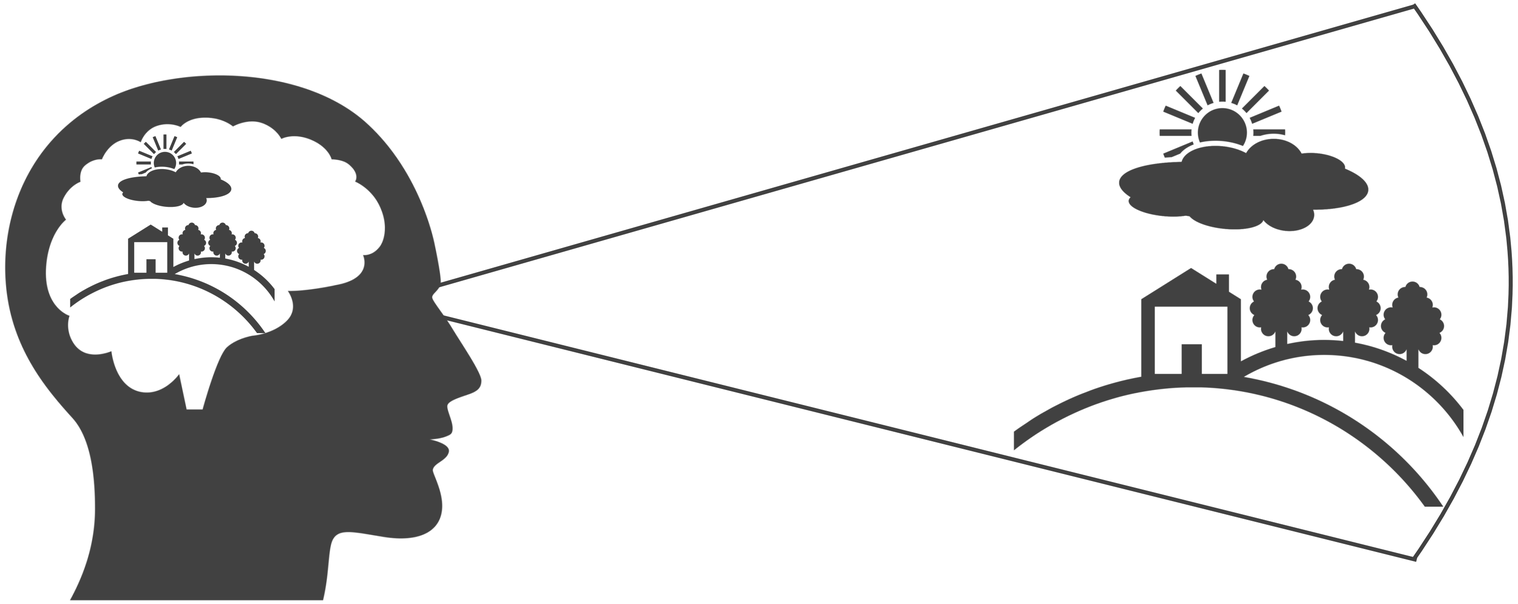
```
(λ (x y z u) (map (λ (v) (* (/
(* (power (/ (* x x) (fold (zip
z u (λ (w a) (- w a))) θ (λ (b
c) (+ (* b b)    c)))) (/ (* 1
1) (+ 1 1))) y) (fold (zip z u
(λ (d e)  (- d e))) θ (λ (f g)
(+ (* f f) g)))) v)) (zip z u
(λ (h i) (- h i)))))
```

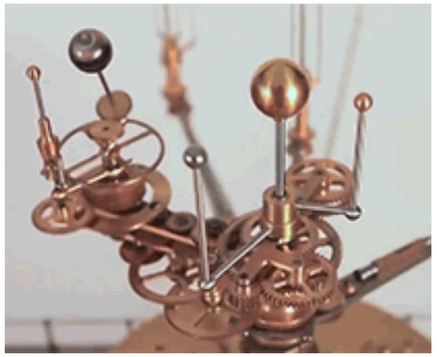
Solution to Coulomb's Law if expressed in initial primitives

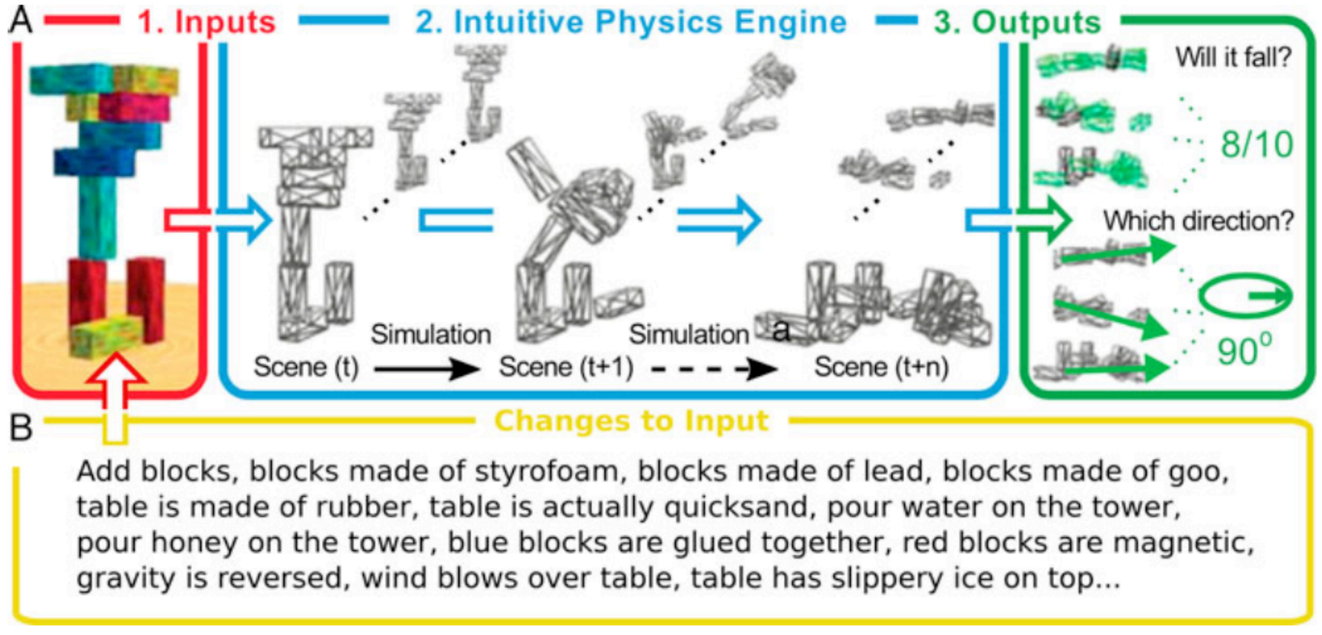
(Ellis et al., 2023)

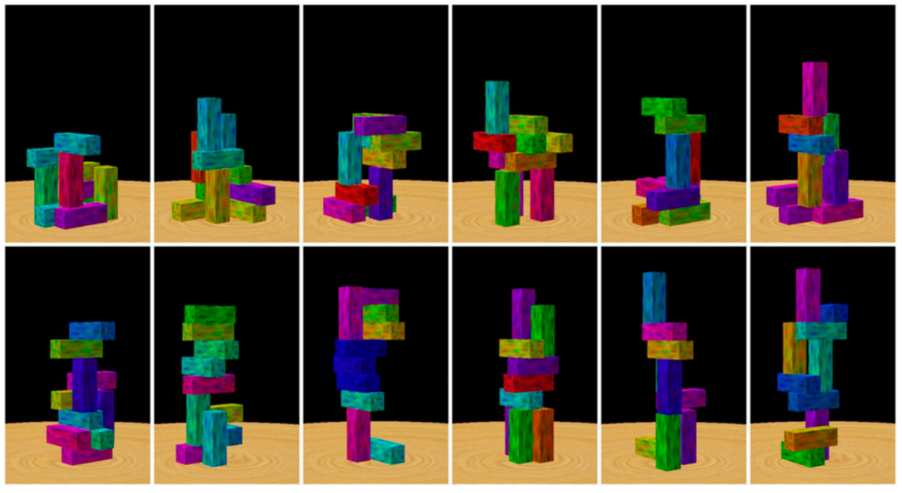


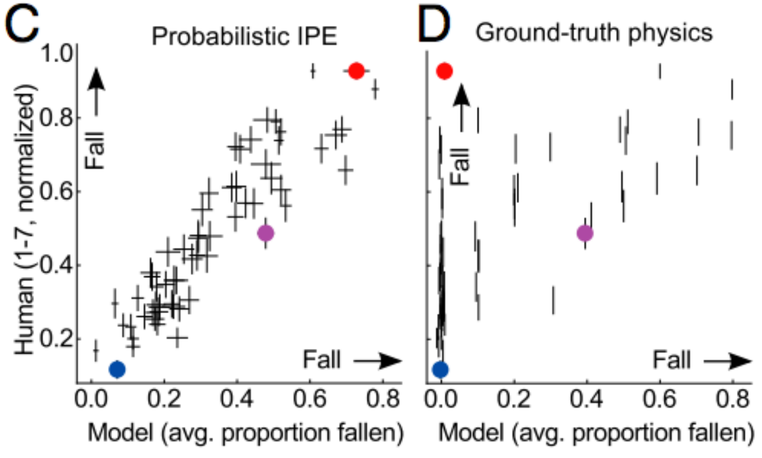
(Ellis et al., 2023)



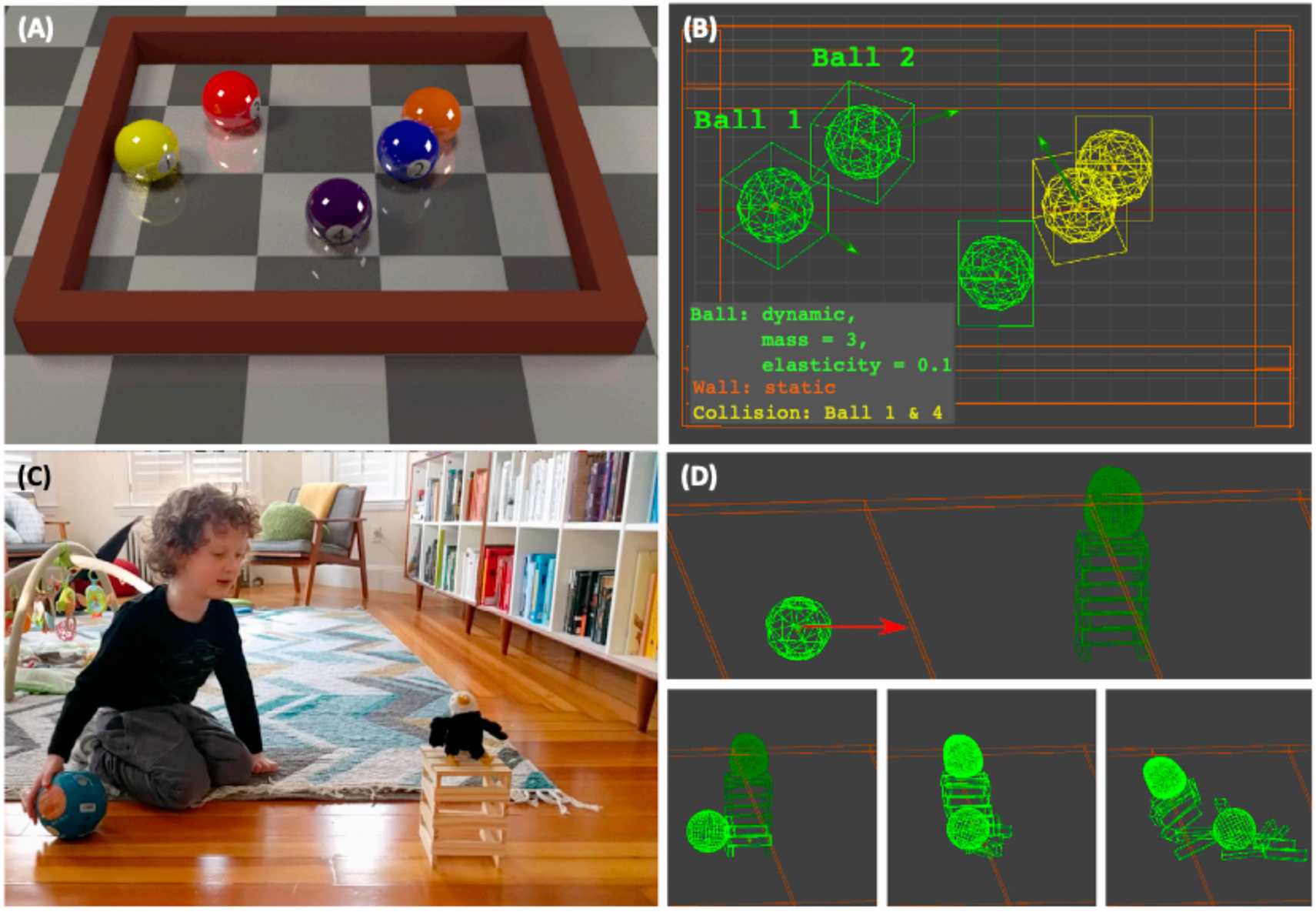








(Battaglia et al., 2013)



(Ullman et al., 2017)

Octodad™ Editor - Content/Levels/Church_SubReception.irr

File Edit View Options Tools Window Help

Scene Tree

Collapse | Go To ID

- root
  - (LevelInfo)_999
  - Reception
    - Architecture
    - ArchitectureTemp
    - Chairs
    - Door_183
    - Door_1184
    - GrabKnob
    - CakeTable
    - CakeTable
    - Vases
    - Tables
    - Cake
    - Spot_1001
    - Spot_1002
    - Party
    - Sign
    - TieGone_1066
    - TieGame_1065
    - GrabTie
    - ChangingRoom
    - Objectives
    - Cameras
      - Camera_269
      - CamSnap_1306
      - Camera_1312
      - Cam_1304
      - SplineCam_1305
      - Camera_1308
      - SplineCam_1307
      - DressCam_304
      - CamLeft
      - Cam_305
      - RightCam_1314
    - Bananers
  - Life
  - DressingSpot_8000
  - (light)_1004
  - (light)_1005
  - DanceMusic_1003
  - CelebrationSound_1068
  - ClothesChange_1070

Attributes

1- ID & Positions	
Name	Architecture
Id	-1
AbsolutePosition	1.783, 10.972, 1.703
Position	1.783, 10.972, 1.703
Rotation	0, 0, 0
Scale	1, 1, 1
1- Transform Settings	
IgnoreParentTransfo	False
2- General Flags	
Visible	True
VisibleInEditor	True
Pickable	True
VisibleInSublevelOn	False
3- Gameplay	
IsGrabbable	False
ReduceWeightOnGr	True
TurnDynamicOnGra	False
TurnDynamicOnTou	False
IsGold	False
IsButton	False
IsTripHazard	False
CanHangFrom	False
4- Graphics	
MeshFileName	Content/Models/Church/Reception.ob
AlphaOverride	1
HasTransparency	False
DisableAntialiasing	False
IsReflective	False
ReceiveCaustics	False
CastShadow	False
ReceiveShadow	True
UseRimShading	False
ClipObject	True
5- Physics	
BoundType	Mesh
ColMeshFileName	Content/Models/Church/ReceptionCO
PhysicsState	Static

Name
A string identifying the object.

Attributes | Materials | Animators

Scene Viewer

Content/Levels/Church_SubReception.irr

Log Window

```
Could not open material file: BananaBowlCOL.mtl
Loaded mesh: Content/Models/Church/BananaBowlCOL.obj
Loaded texture: Content/Models/Church/BananaWholeTexture.jpg
Loaded mesh: Content/Models/Church/BananaWhole.obj
Loaded texture: Content/Models/Editor/default_objective.png
Loaded mesh: Content/Models/Editor/Path.obj
Loaded texture: Content/Models/Editor/Gray.png
Loaded texture: C:\Users\Kevin\Desktop\Octodad 2\Octodad\aa\Octodad\Content\Models\Church\archway.
Loaded mesh: C:\Users\Kevin\Desktop\Octodad 2\Octodad\aa\Octodad\Content\Models\Church\archway.o
```

Asset Library

Models

- Church
  - Altar.obj
  - altoids.obj
  - archway.obj
  - archwayCOL.obj
  - balloon.obj
  - balloonCluster.obj
  - BananaBowl.obj
  - BananaBowlCOL.obj
  - BananaWhole.obj

Scene Notes

Level notes go here.

Scene Search

Name	ID	Type
DanceMusic	1003	SoundO...
DiningPlate	7002	Physics...
Door	183	Advanc...
Door	1184	Advanc...
DoorKey	1188	Physics...
DoorStay	-1	Trigger...
DressCam	304	Camera...
DressingSpot	8000	light
Figure	-1	Physics...
FlickerLight	-1	empty
FlickerLightOn	8001	Trigger...

FPS: 350 | Total Collision Polys: 0 | Total Polys Drawn: 54621| Objects Drawn: 169| Objects Culled: 12| Cam Pos: {37, 33, 0}


File Edit View Editors Camera Renderer Engine Tools DEV Help
Scene Entity List
Lights
Default - Light Directional
Characters
Knightress_153
Knightress_154
Knightress_152
Knightress_151
Knightress_154
Scene
City Wall
Fence
Misc
Rocks
Rock2_2
Rock1B_247
Rock4A_12
Rock2_46
Rock2_29
Rock2_75
Rock2_40
Rock2_8
Rock4A_120
Rock2_6
Rock2_26
Skyline Console
Finished Updating Billboards
Finished Updating Billboards
Finished Updating Billboards
LOADING XML: Terrain Road Settings
LOADING XML: CAMERA Settings
LOADING XML: Load Bookmarks
LOADING XML: Post Effects
LOADING XML: SHADOW Settings
LOADING XML: Navmesh Settings
LOAD LEVEL COMPLETED [XML]
Clear Log
Enter Commands Here....
World Space
Terrain Editor
Creation
Height Edit
Paint
Vegetation
Terrain Generator
Terrain Setup
Terrain Size (Resolution)
512
World Size (Size in World Units
1000
Texture Blend Map Size:
1024
Tiles X
0
Tiles Y
0
Generate
Terrain Options:
Generate
Remove
Apply Physics
Import Options:
Export Options:
Load Terrain file
Export Terrain file
Load HeightMap
Export HeightMap
Terrain Properties
Heightmap Multiplier
75
Use Surface Names
Update Surfaces
Open AM Heightmaps
Open
Lightmap
Material
Brush Size
Brush Strength
Scene Settings
Terrain Editor
Particle Editor

learning the ‘game engine in your head’

Trends in Cognitive Sciences

CellPress REVIEWS

Opinion

The Child as Hacker

Joshua S. Rule,[1,*] Joshua B. Tenenbaum,[1] and Steven T. Piantadosi[2]

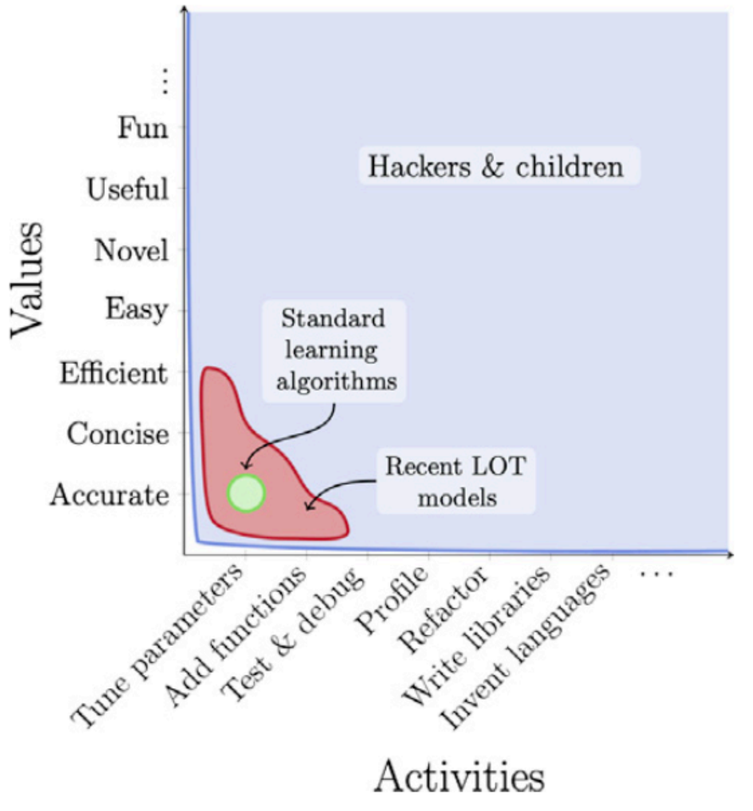
Table 2. Learners and Hackers Share Similar Techniques[a]

Tune parameters	Adjust constants in code to optimize an objective function.
Add functions	Write new procedures for the codebase, increasing its overall abilities by making new computations available for reuse.
Extract functions	Move existing code into its own named procedure to centrally define an already common computation.
Test and debug	Execute code to verify that it behaves as expected and fix problems that arise. Accumulating tests over time increases code's trustworthiness.
Handle errors	Recognize and recover from errors rather than failing before completion, thereby increasing robustness.
Profile	Observe a program's resource use as it runs to identify inefficiencies for further scrutiny.
Refactor	Restructure code without changing the semantics of the computations performed (e.g.,remove dead code, reorder statements).
Add types	Add code explicitly describing a program's semantics, so syntax better reflects semantics and supports automated reasoning about behavior.
Write libraries	Create a collection of related representations and procedures that serve as a toolkit for solving an entire family of problems.
Invent languages	Create new languages tuned to particular domains (e.g.,HTML, SQL, LaTeX) or approaches to problem solving (e.g.,Prolog, C, Scheme).

(Rule et al., 2020)

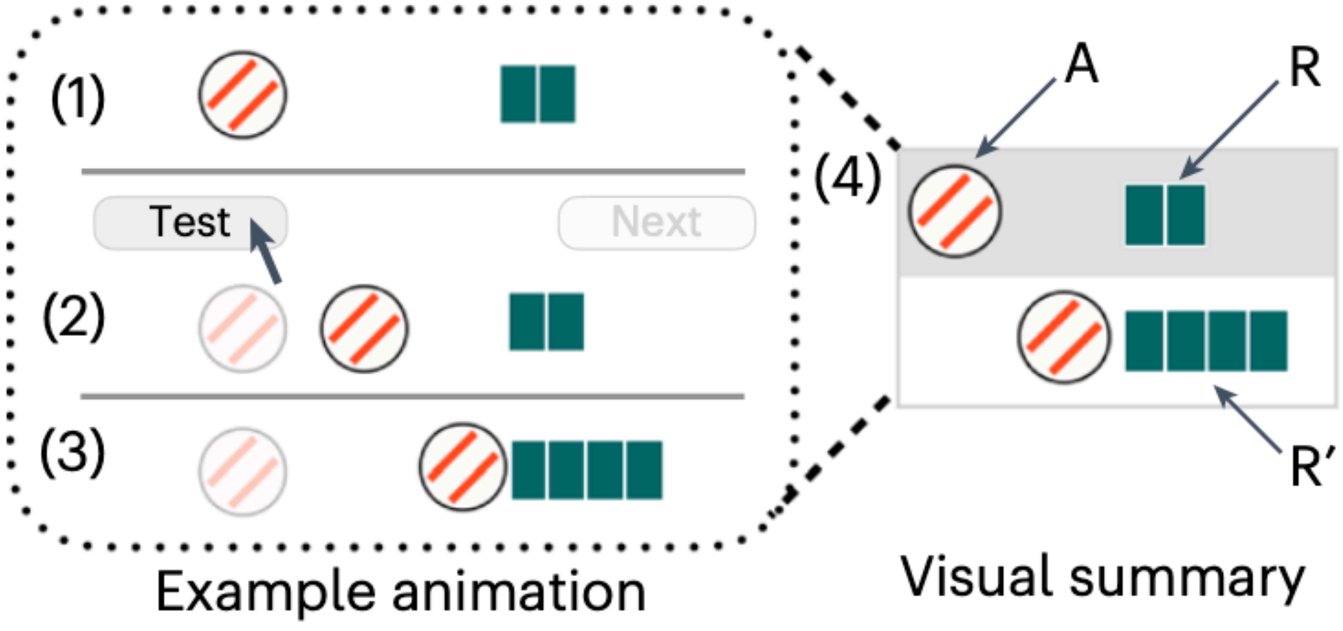
Table I. A Sampling of Domains Requiring Algorithmic Knowledge Formalizable as Programs, with Motivating Examples.

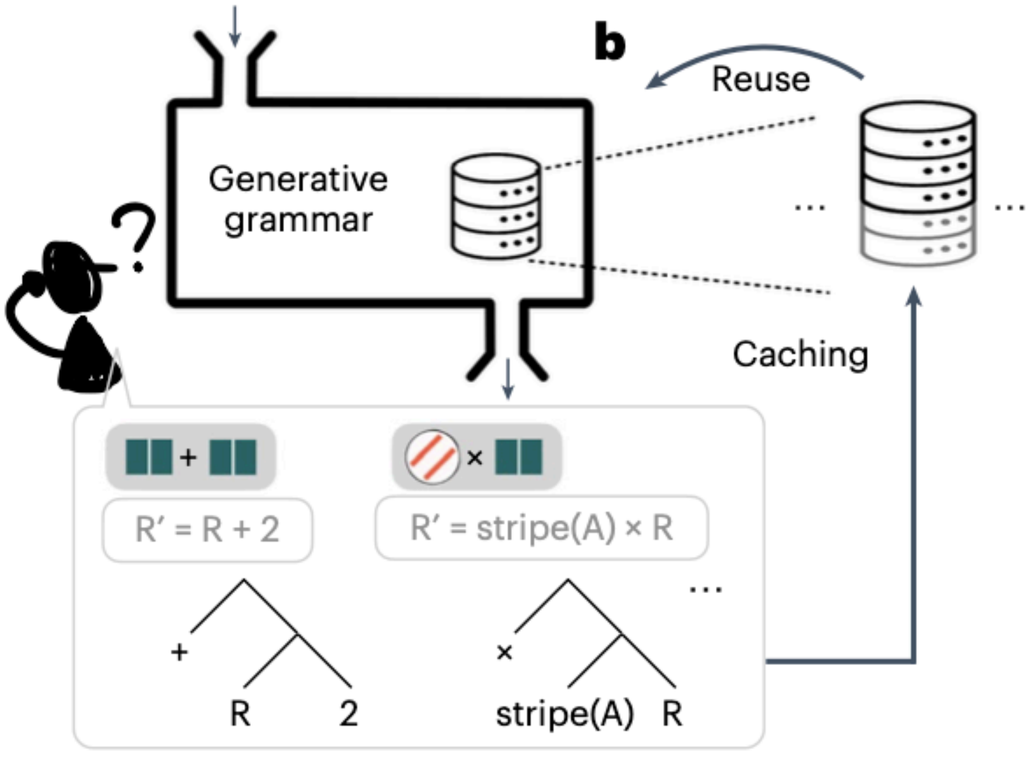
Logic	First-order, modal, deontic logic
Mathematics	Number systems, geometry, calculus
Natural language	Morphology, syntax, number grammars
Sense data	Audio, images, video, haptics
Computer languages	C, Lisp, Haskell, Prolog, LaTeX
Scientific theories	Relativity, game theory, natural selection
Operating procedures	Robert's rules, bylaws, checklists
Games and sports	Go, football, 8 queens, juggling, Lego
Norms and mores	Class systems, social cliques, taboos
Legal codes	Constitutions, contracts, tax law
Religious systems	Monastic orders, vows, rites and rituals
Kinship	Genealogies, clans/moieties, family trees
Mundane chores	Knotting ties, making beds, mowing lawns
Intuitive theories	Physics, biology, theory of mind
Domain theories	Cooking, lockpicking, architecture
Art	Music, dance, origami, color spaces



(Rule et al., 2020)

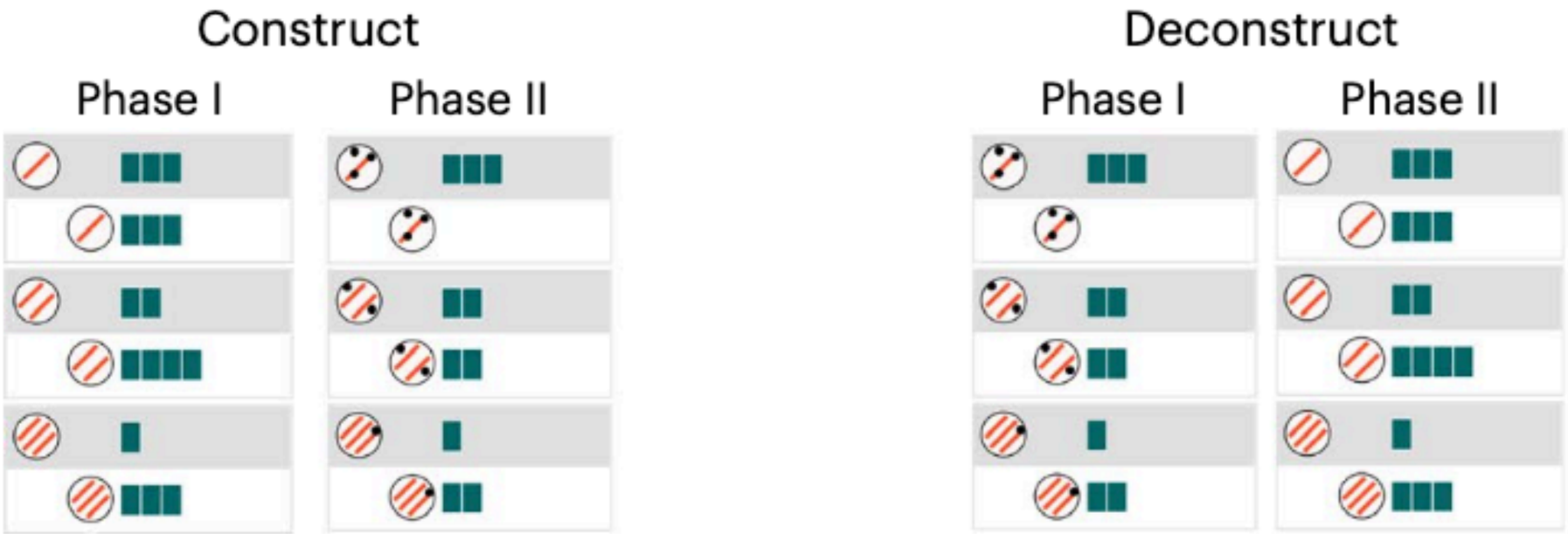
experiment


(1)
Test
Next
(2)
(3)
Example animation
(4)
A
R
R′
Visual summary

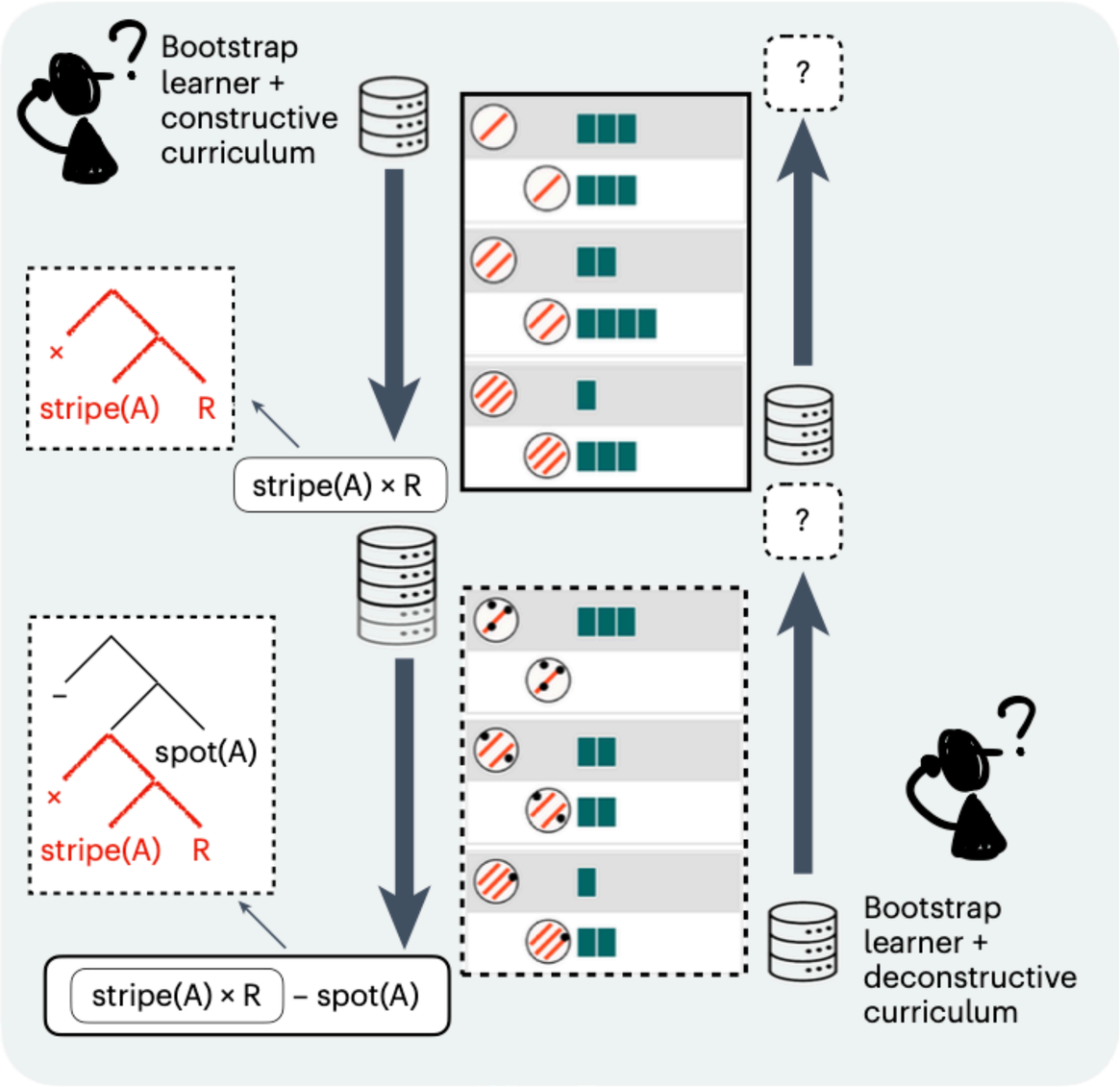


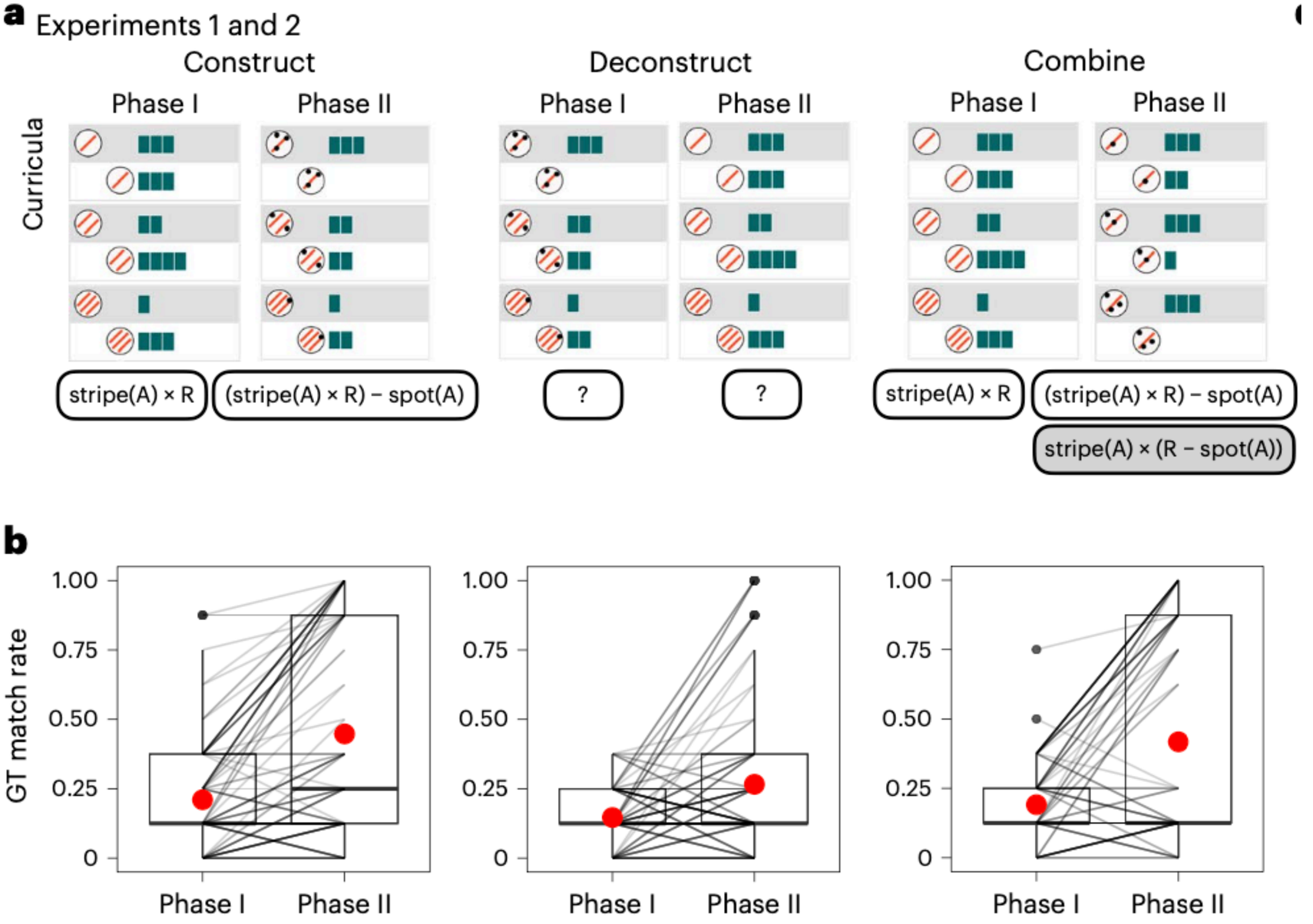
Example concepts

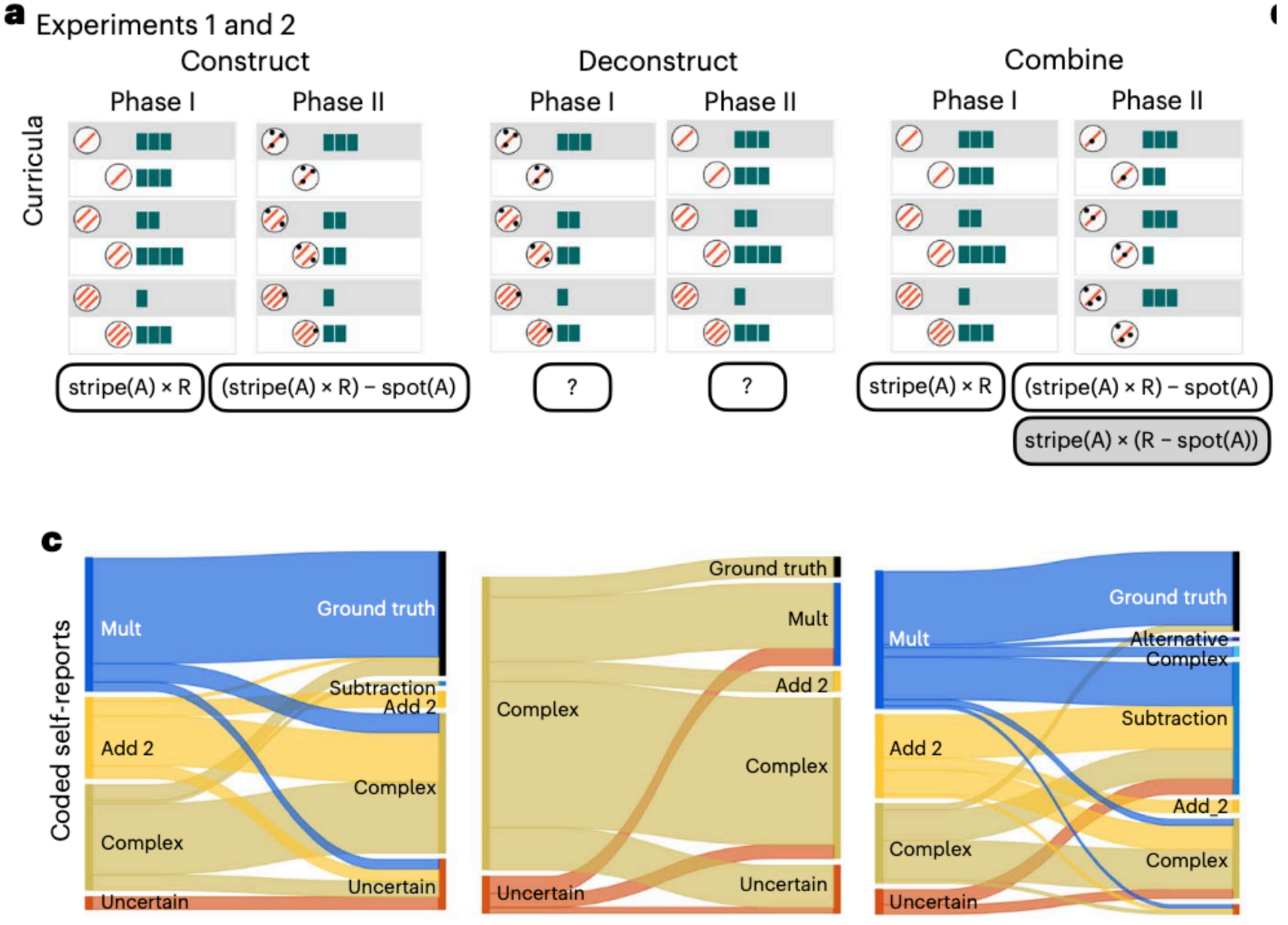
curricula

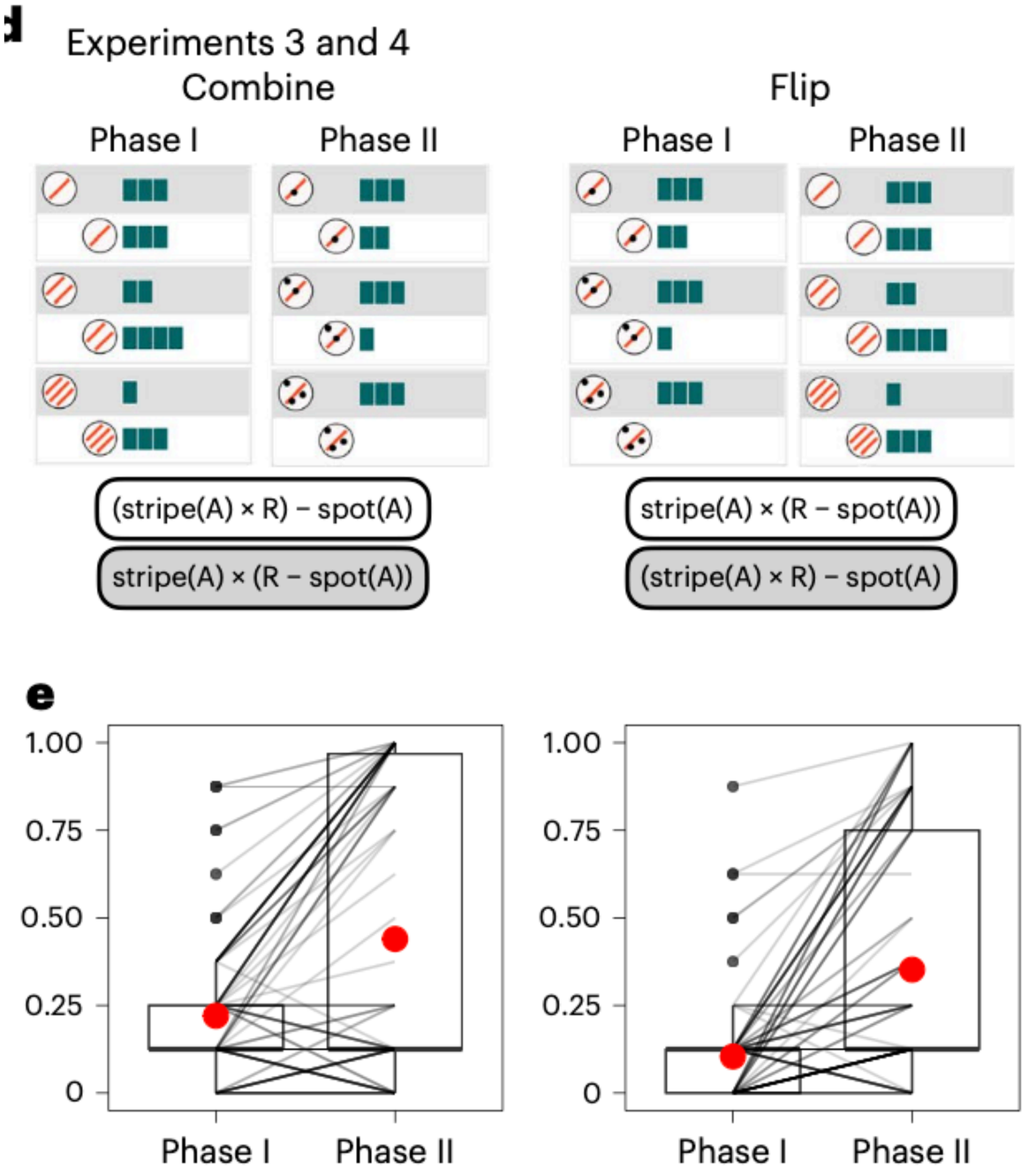


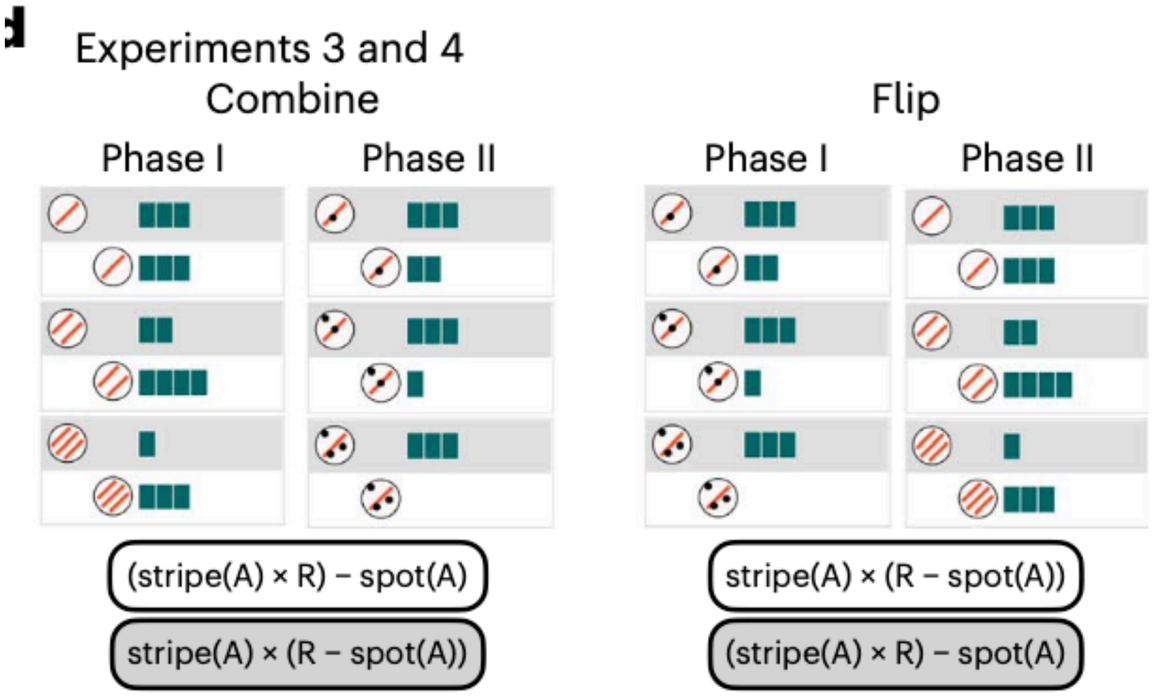
curricula

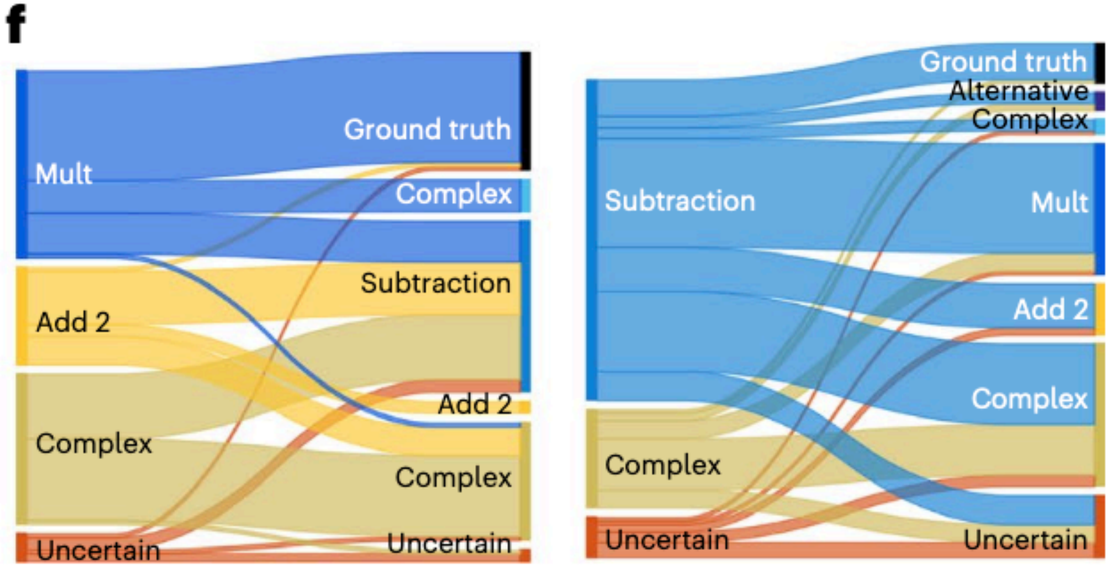

Bootstrap learner + constructive curriculum
?
×
stripe(A)
R
stripe(A) × R
?
?
–
spot(A)
×
stripe(A)
R
stripe(A) × R
– spot(A)
Bootstrap learner + deconstructive curriculum


a
Experiments 1 and 2
Construct
Deconstruct
Combine
Phase I
Phase II
Curricula
stripe(A) × R
(stripe(A) × R) − spot(A)
?
stripe(A) × (R − spot(A))
b
GT match rate
1.00
0.75
0.50
0.25
0


a
Experiments 1 and 2
Construct
Deconstruct
Combine
Phase I
Phase II
Curricula
stripe(A) × R
(stripe(A) × R) − spot(A)
?
stripe(A) × (R − spot(A))
c
Coded self-reports
Mult
Add 2
Complex
Uncertain
Ground truth
Subtraction
Alternative
Add_2


d
Experiments 3 and 4
Combine
Flip
Phase I
Phase II
Phase I
Phase II
(stripe(A) × R) − spot(A)
stripe(A) × (R − spot(A))
stripe(A) × (R − spot(A))
(stripe(A) × R) − spot(A)
e
1.00
0.75
0.50
0.25
0
Phase I
Phase II
1.00
0.75
0.50
0.25
0
Phase I
Phase II


Experiments 3 and 4
Combine
Phase I
Phase II
(stripe(A) × R) − spot(A)
stripe(A) × (R − spot(A))
Flip
Phase I
Phase II
stripe(A) × (R − spot(A))
(stripe(A) × R) − spot(A)


f
Mult
Add 2
Complex
Uncertain
Ground truth
Complex
Subtraction
Add 2
Complex
Uncertain
Subtraction
Complex
Uncertain
Ground truth
Alternative
Complex
Mult
Add 2
Complex
Uncertain

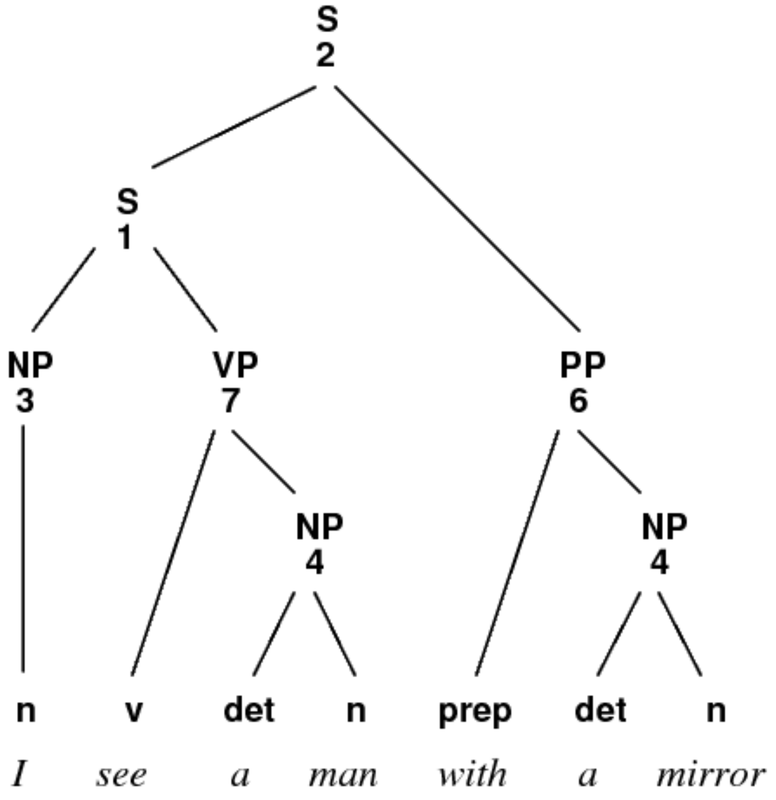
model

background

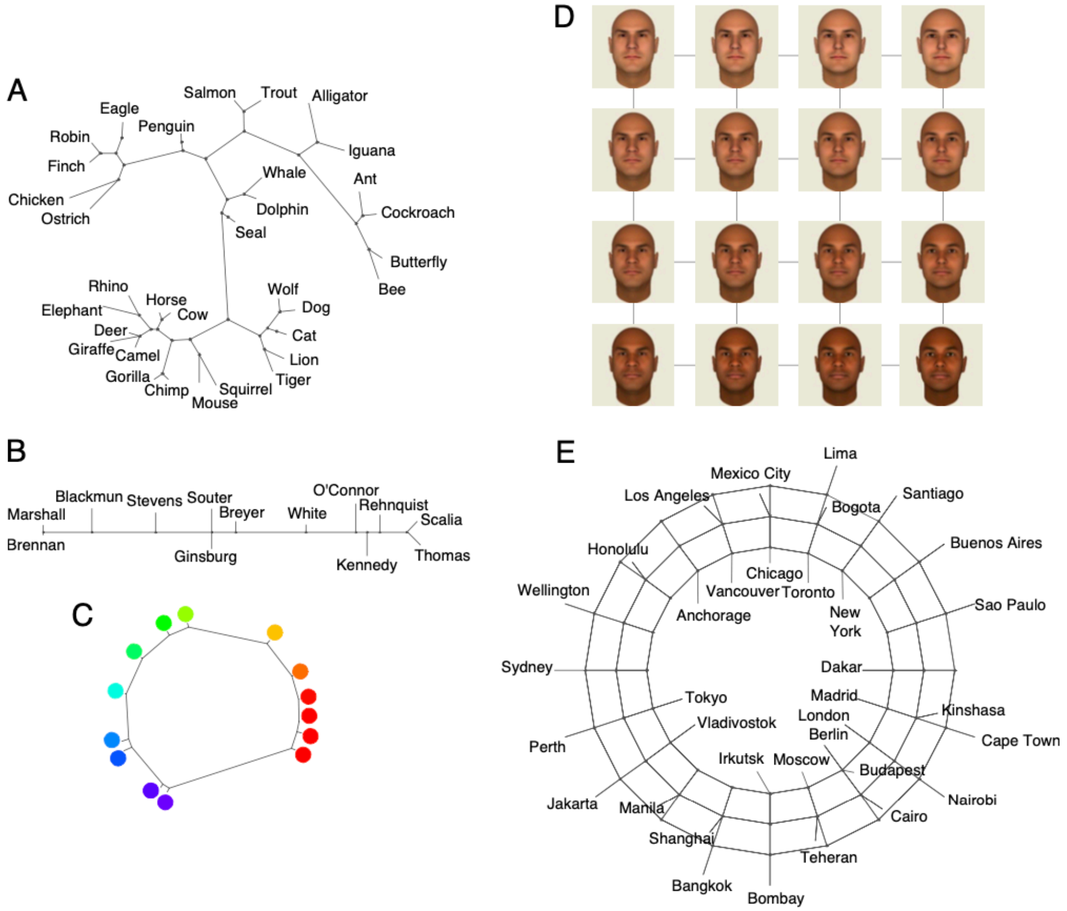
1. Probabilistic context free grammar (PCFG)
2. Adaptor Grammar (AG)
3. Pitman-Yor Adaptor Grammar (PYAG)
4. Combinatory Logic (CL)
5. CL with routers

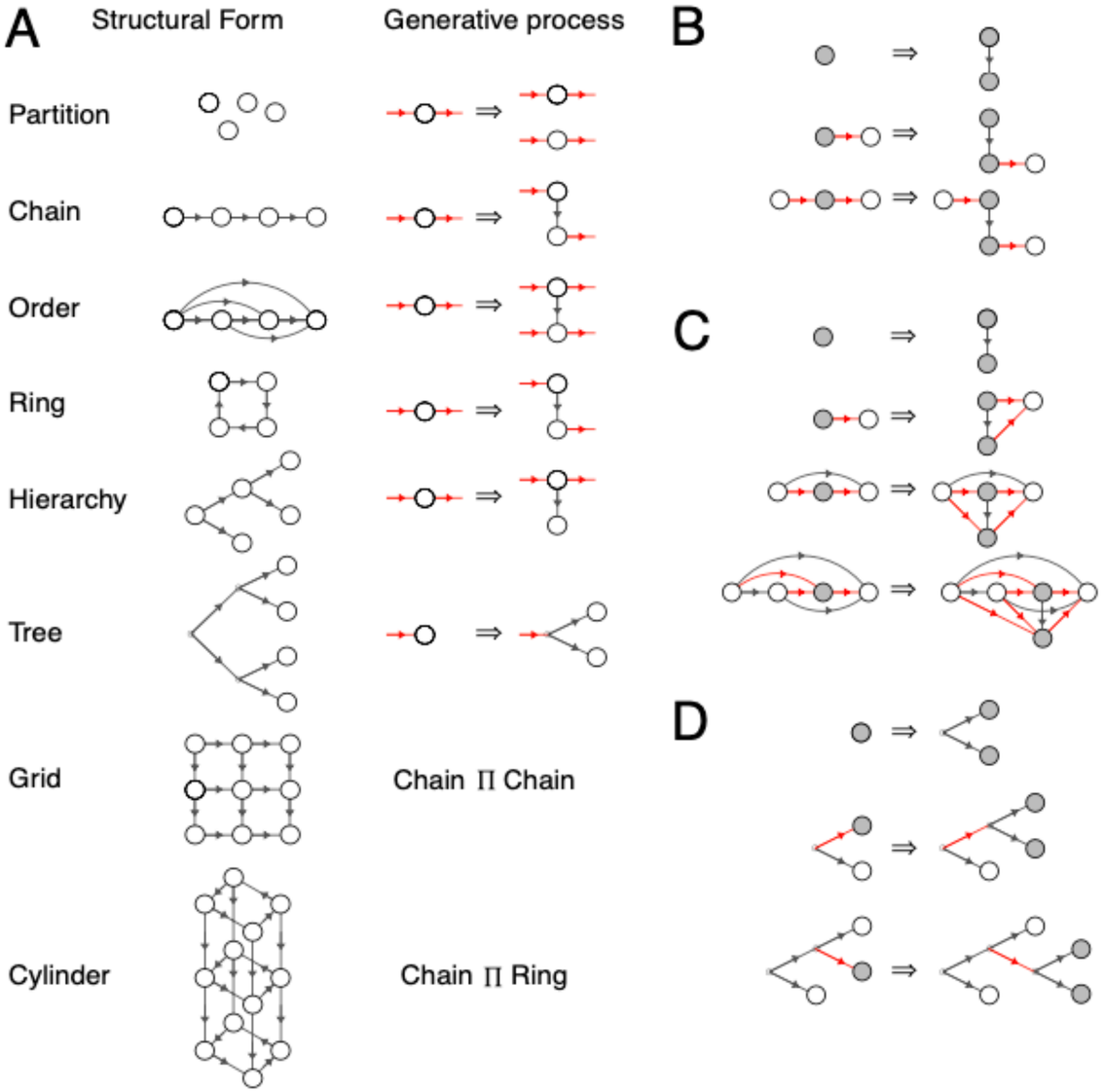
background

1. **Probabilistic context free grammar (PCFG)**
2. Adaptor Grammar (AG)
3. Combinatory Logic (CL)
4. CL with routers



(1)	S	::=	NP	VP
(2)	S	::=	S	PP
(3)	NP	::=	n	
(4)	NP	::=	det	n
(5)	NP	::=	NP	PP
(6)	PP	::=	prep	NP
(7)	VP	::=	v	NP

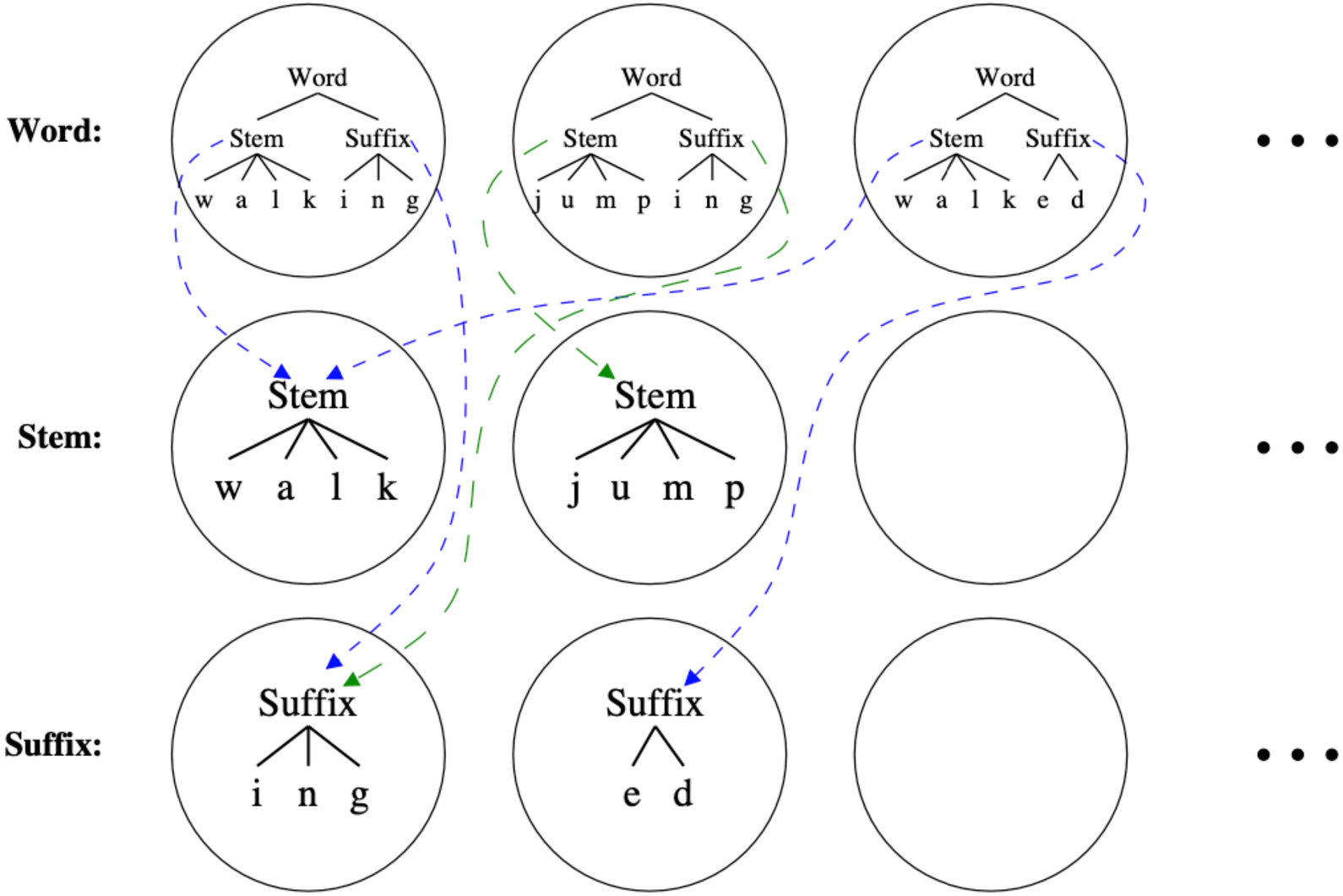




(Kemp & Tenenbaum 2008)

background

1. Probabilistic context free grammar (PCFG)
2. **Adaptor Grammar (AG)**
3. Combinatory Logic (CL)
4. CL with routers



(Johnson et al., 2007)

CRP


+1
+100
+1000
1
new
Total customers: 0.5
Function:
DirichletProcess(0.5,H)
1.0
0.5
0.0
Customers: 0.5
Probability: 1

Pitman-Yor

old table k

$$z_{n+1}|z_1,\ldots,z_n \sim \boxed{\frac{m\cancel{a}+b}{n+b}}\delta_{m+1} + \sum_{k=1}^{m}\boxed{\frac{n_k-\cancel{a}}{n+b}}\delta_k$$

new table

$$a = 0$$

background

1. Probabilistic context free grammar (PCFG)
2. Adaptor Grammar (AG)
3. **Combinatory Logic (CL)**
4. CL with routers

models of computation

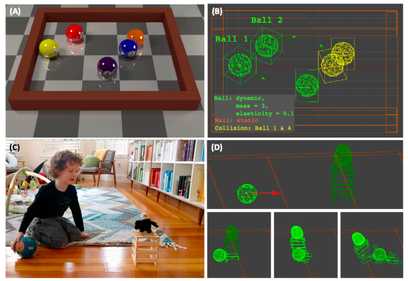
Turing machines

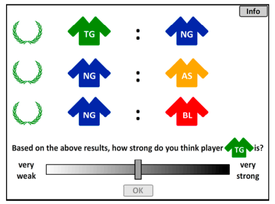
λ-calculus

Combinatory logic

Church (ψλ-calculus)

this is something else





Kinetic Energy

$$KE = \frac{1}{2} m |\vec{v}|^2$$

```
(ab/2 m (|v|^2 v))
```

Coulomb's Law

$$\vec{F} \propto \frac{q_1 q_2}{|\vec{r}_1 - \vec{r}_2|^2} \widehat{\vec{r}_1 - \vec{r}_2}$$

```
(inverse-square q_1 q_2
(subtract-vectors r_1 r_2))
```

```
(λ (x y z u) (map (λ (v) (* (/
(* (power (/ (* x x) (fold (zip
z u (λ (w a) (- w a))) 0 (λ (b
c) (+ (* b b)    c)))) (/ (* 1
1) (+ 1 1))) y) (fold (zip z u
(λ (d e)  (- d e))) 0 (λ (f g)
(+ (* f f) g)))) v)) (zip z u
(λ (h i) (- h i)))))
```

Solution to Coulomb's Law if expressed in initial primitives

Singular-Plural

λ *S* . (*if* (*singleton? S*)
"one"
"two")

Mod-5

λ *S* . (*if* (*or* (*singleton? S*)
(*equal-word?* (*L* (*set-difference S*)
(*select S*))
"five"))
"one"
(*next* (*L* (*set-difference S*
(*select S*)))))

2-not-1-knower

λ *S* . (*if* (*doubleton? S*)
"two"
undef)

2N-knower

λ *S* . (*if* (*singleton? S*)
"one"
(*next* (*next* (*L* (*set-difference S* (*select S*))))))

S combinator

```
s[x_][y_][z_] -> x[z][y[z]]]
```

K combinator

```
k[x_][y_] -> x
```

a[b[a][c]]

↕

```
s[s[k[s]][s[k[k]][s[k[s]][k]]]][s[k[s[s[k][k]]]][k]]
```

s[s[k[s]][s[k[k]][s[k[s]][k]]]][s[k[s[s[k][k]]]][k]][a][b][c]
s[k[s]][s[k[k]][s[k[s]][k]]][a][s[k[s[s[k][k]]]][k][a]][b][c]
k[s][a][s[k[k]][s[k[s]][k]][a]][s[k[s[s[k][k]]]][k][a]][b][c]
s[s[k[k]][s[k[s]][k]][a]][s[k[s[s[k][k]]]][k][a]][b][c]
s[k[k]][s[k[s]][k]][a][b][s[k[s[s[k][k]]]][k][a][b]][c]
k[k][a][s[k[s]][k][a]][b][s[k[s[s[k][k]]]][k][a][b]][c]
k[s[k[s]][k][a]][b][s[k[s[s[k][k]]]][k][a][b]][c]
s[k[s]][k][a][s[k[s[s[k][k]]]][k][a][b]][c]
k[s][a][k[a]][s[k[s[s[k][k]]]][k][a][b]][c]
s[k[a]][s[k[s[s[k][k]]]][k][a][b]][c]
k[a][c][s[k[s[s[k][k]]]][k][a][b][c]]
a[s[k[s[s[k][k]]]][k][a][b][c]]
a[k[s[s[k][k]]][a][k[a]][b][c]]
a[s[s[k][k]][k[a]][b][c]]
a[s[k][k][b][k[a][b]][c]]
a[k[b][k[b]][k[a][b]][c]]
a[b[k[a][b]][c]]
a[b[a][c]]

(Wolfram, 2020)

s[k[s]][s[k[s[k[s]][k]]]]

If we call this **plus**, then let's compute **plus**[1][2][s][k], where 1 and 2 are represented by combinators:

```
s[k[s]][s[k[s[k[s]][k]]]][s[s[k[s]][k]][s[k]]][s[s[k[s]][k]][s[s[k[s]][k]][s[k]]]][s]][k]
k[s][s[s[k[s]][k]][s[k]]][s[k[s[k[s]][k]]][s[s[k[s]][k]][s[k]]]][s[s[k[s]][k]][s[s[k[s]][k]][s[k]]]][s]][k]
s[s[k[s[k[s]][k]]][s[s[k[s]][k]][s[k]]]][s[s[k[s]][k]][s[s[k[s]][k]][s[k]]]][s]][k]
s[k[s[k[s]][k]]][s[s[k[s]][k]][s[k]]]][s][s[s[k[s]][k]][s[s[k[s]][k]][s[k]]]][s]][k]
k[s[k[s]][k]][s][s[s[k[s]][k]][s[k]]][s]][s[s[k[s]][k]][s[s[k[s]][k]][s[k]]]][s]][k]
s[k[s]][k][s[s[k[s]][k]][s[k]]][s]][s[s[k[s]][k]][s[s[k[s]][k]][s[k]]]][s]][k]
k[s][s[s[k[s]][k]][s[k]]][s]][k[s[s[k[s]][k]][s[k]]][s]]][s[s[k[s]][k]][s[s[k[s]][k]][s[k]]]][s]]][k]
s[k[s[s[k[s]][k]][s[k]]][s]]][s[s[k[s]][k]][s[s[k[s]][k]][s[k]]]][s]][k]
k[s[s[k[s]][k]][s[k]]][s]][k][s[s[k[s]][k]][s[s[k[s]][k]][s[k]]]][s]][k]]
s[s[k[s]][k]][s[k]][s]][s[s[k[s]][k]][s[s[k[s]][k]][s[k]]]][s]][k]]
s[k[s]][k]][s]][s[k]][s]][s[s[k[s]][k]][s[s[k[s]][k]][s[k]]]][s]][k]]
k[s][s]][k[s]][s[k]][s]][s[s[k[s]][k]][s[s[k[s]][k]][s[k]]]][s]][k]]
s[k[s]][s[k][s]][s[s[k[s]][k]][s[s[k[s]][k]][s[k]]]][s]][k]]
k[s][s[s[k[s]][k]][s[s[k[s]][k]][s[k]]]][s]][k]][s[k]][s]][s[s[k[s]][k]][s[s[k[s]][k]][s[k]]]][s]][k]]]
s[s[k][s]][s[s[k[s]][k]][s[s[k[s]][k]][s[k]]]][s]][k]]]
s[k[s[s[k[s]][k]][s[s[k[s]][k]][s[k]]]][s]][k]][s[s[s[k[s]][k]][s[s[k[s]][k]][s[k]]]][s]][k]]]]
s[s[s[k[s]][k]][s[s[k[s]][k]][s[k]]]][s]][k]]
s[s[k[s]][k][s]][s[s[k[s]][k]][s[k]]][s]][k]]
s[s[k[s]][k]][s[s[k[s]][k]][s[k]]][s]][k]]
s[k[s][s]][k[s]][s[s[k[s]][k]][s[k]]][s]][k]]
s[s[k[s]][s[s[k[s]][k]][s[k]][s]][k]]
s[k[s][k][s[s[k[s]][k]][s[k]][s][k]]]
s[s[s[s[k[s]][k]][s[k]][s][k]]]
s[s[s[k[s]][k][s][s[k][s]][k]]]
s[s[k[s][s][k[s]][s[k][s]][k]]]
s[s[s[k[s]][s[k][s]][k]]]
s[s[k[s][k][s[k][s][k]]]]
s[s[s[s[k][s][k]]]]
s[s[s[k[k][s[k]]]]]
s[s[s[k]]]
```

(Wolfram, 2020)

What about doing multiplication? There's a combinator for that too, and it's actually rather simple:

s[k[s]][k]

Here's the computation for 3 × 2—giving 6 after 58 steps:

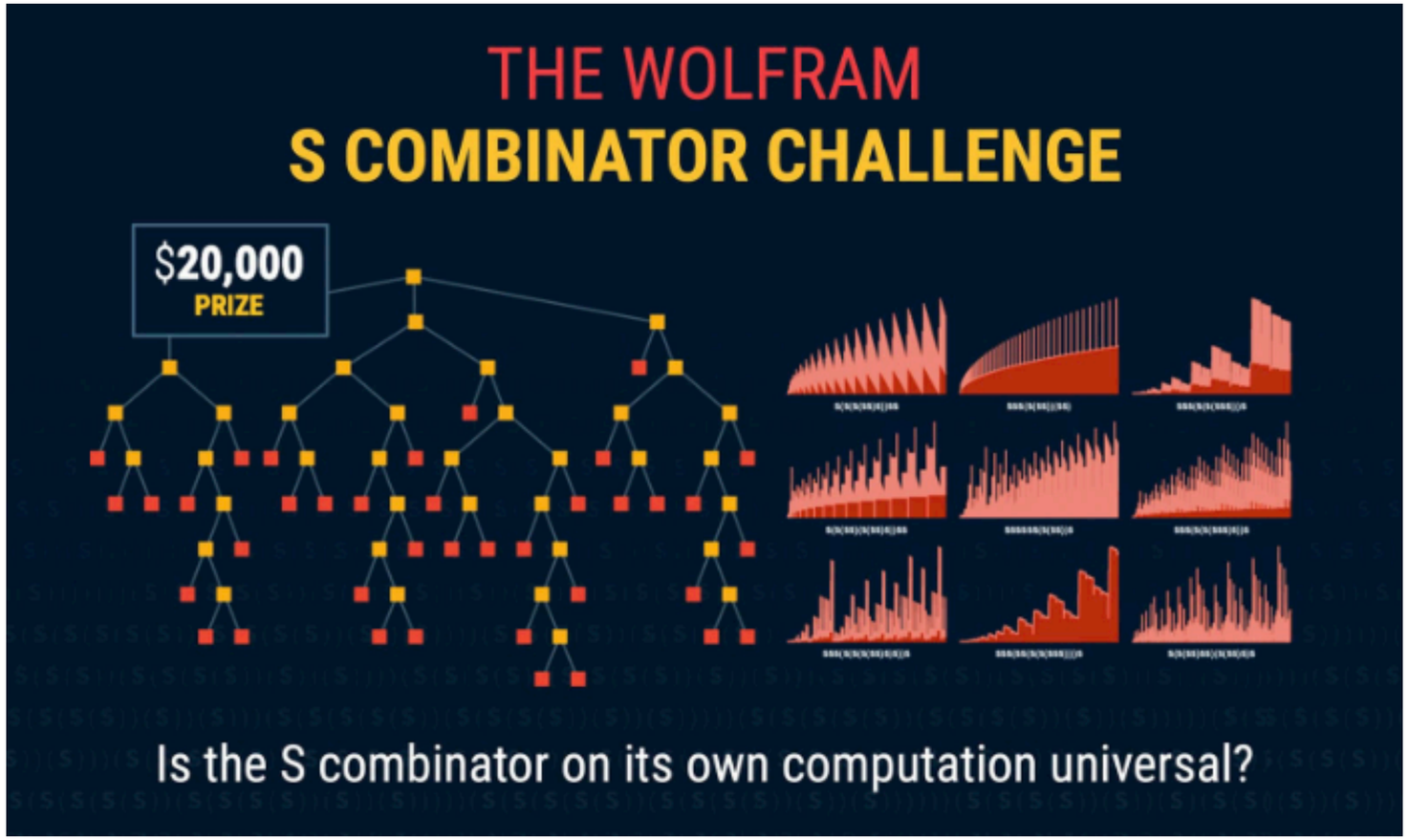
(Wolfram, 2020)

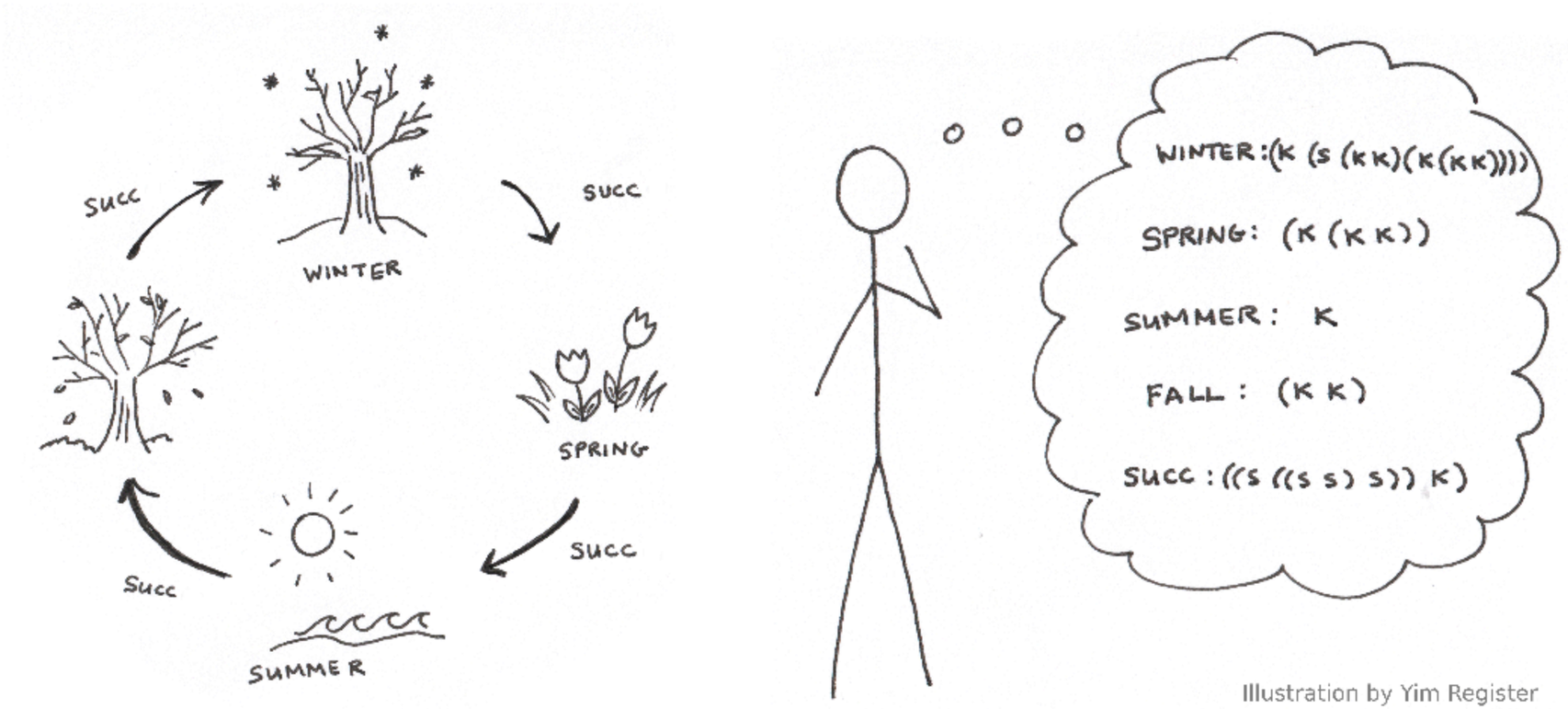
Here's a combinator for power:

s[k[s[s[k][k]]]][k]

And here's the computation of 3^2 using it (which takes 116 steps):

(Wolfram, 2020)


THE WOLFRAM
S COMBINATOR CHALLENGE
$20,000
PRIZE
Is the S combinator on its own computation universal?

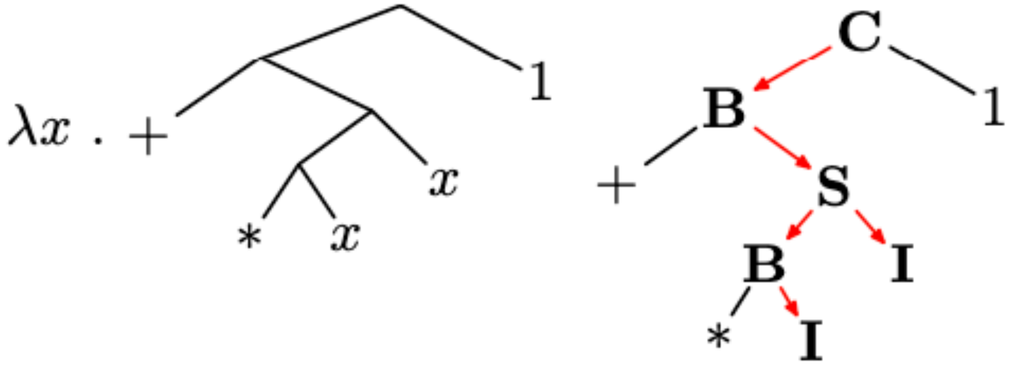


(Piantadosi, 2020)

background

1. Probabilistic context free grammar (PCFG)
2. Adaptor Grammar (AG)
3. Pitman-Yor Adaptor Grammar (PYAG)
4. Combinatory Logic (CL)
5. **CL with routers**

To strike a balance between minimality and practical usability, we make two modifications to **SK** combinatory logic: (1) we introduce higher-order combinators to capture the intuition of routing; and (2) we place these combinators at internal nodes.

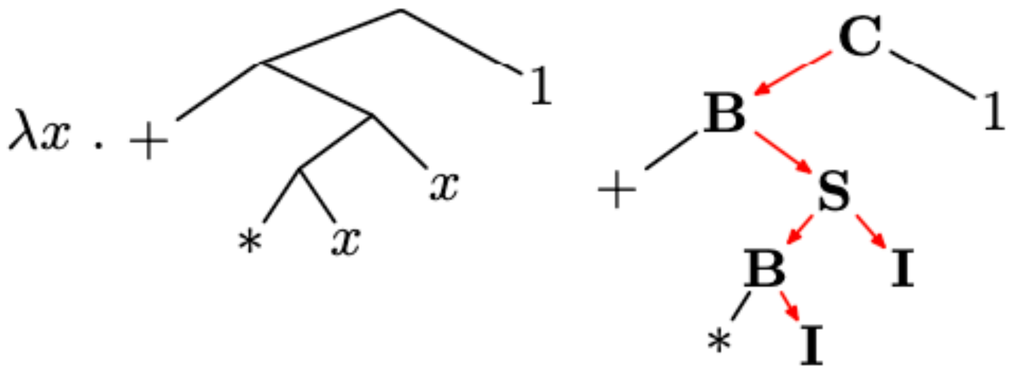


(a) lambda calculus (b) no variables

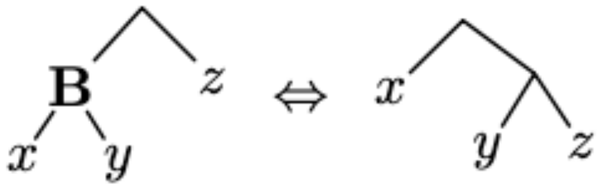
Replace the variable x with $\mathbf{I}$ and label internal nodes of the tree with a *router* (for now, one of $\mathbf{B}, \mathbf{C}, \mathbf{S}$) depending on whether x appeared in the right subtree, left subtree, or both, respectively. The result is Figure 1(b). To apply this function on an argument, we start the argument at the root of the transformed tree. The router at each node determines to which subtrees the argument should be sent. When the argument reaches $\mathbf{I}$, it replaces $\mathbf{I}$.

One significance of the variable-free formalism is that we have eliminated the distinction between program and subprogram. Each subtree is now a valid standalone program, and thus a candidate for multi-task sharing. For example, $(\mathbf{S}\ (\mathbf{B}\ *\ \mathbf{I})\ \mathbf{I})$ denotes the square function, which could be useful elsewhere.

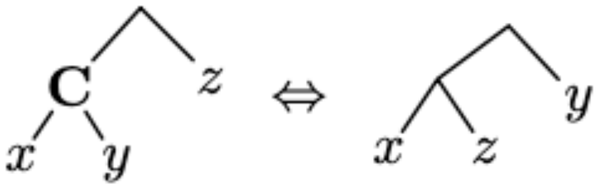
(Liang et al., 2010)



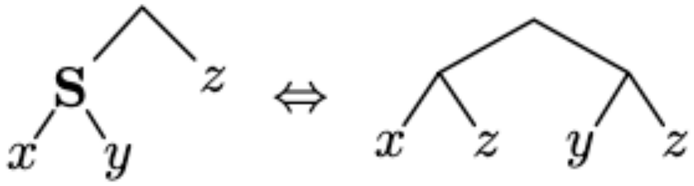
(a) lambda calculus (b) no variables



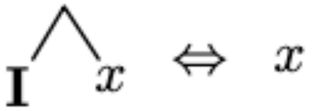
route right



route left

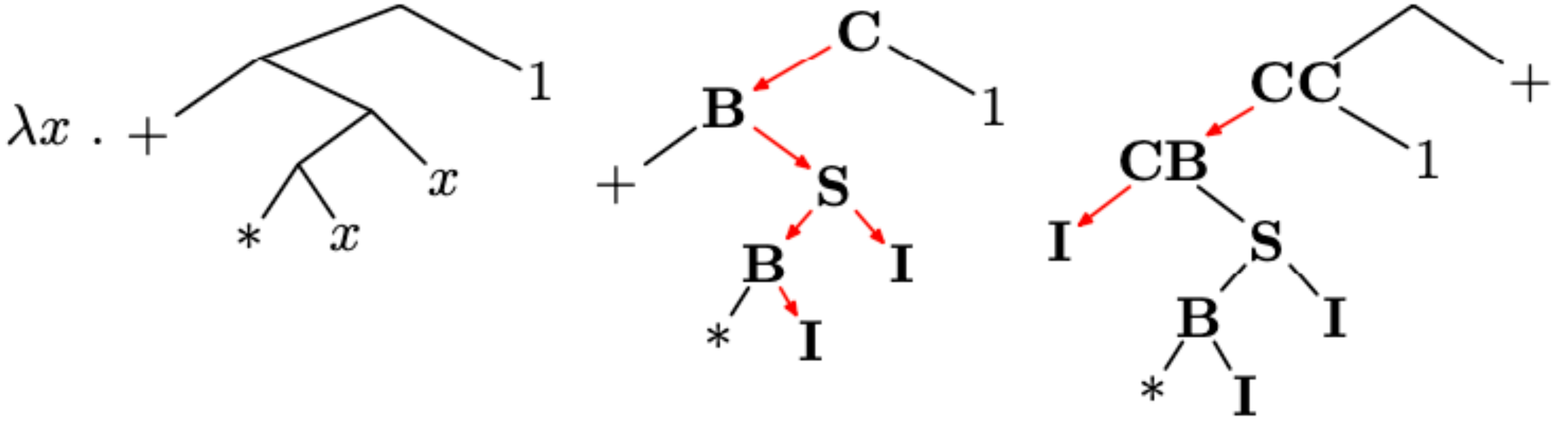


route left and right



destination

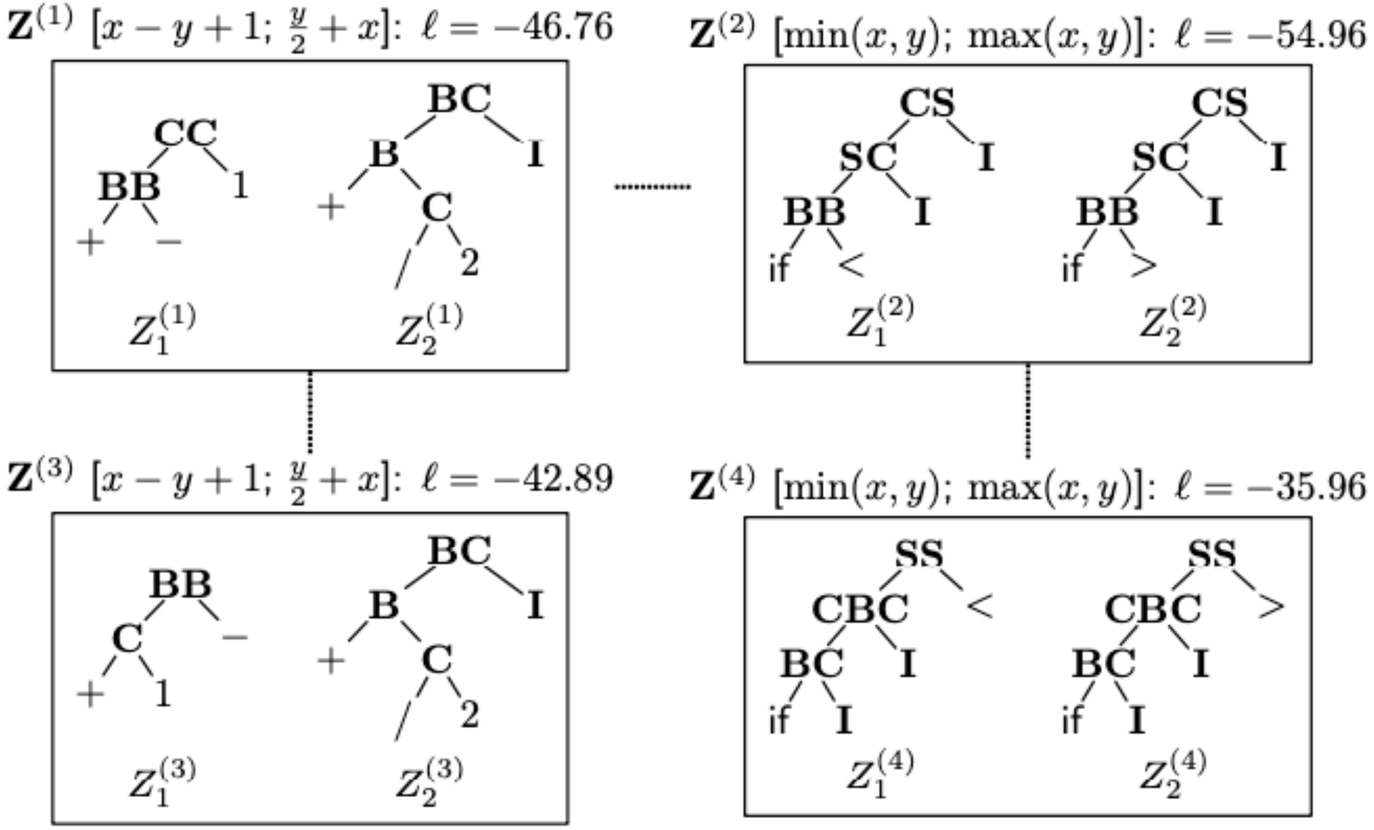
(Liang et al., 2010)



(a) lambda calculus (b) no variables (c) refactored

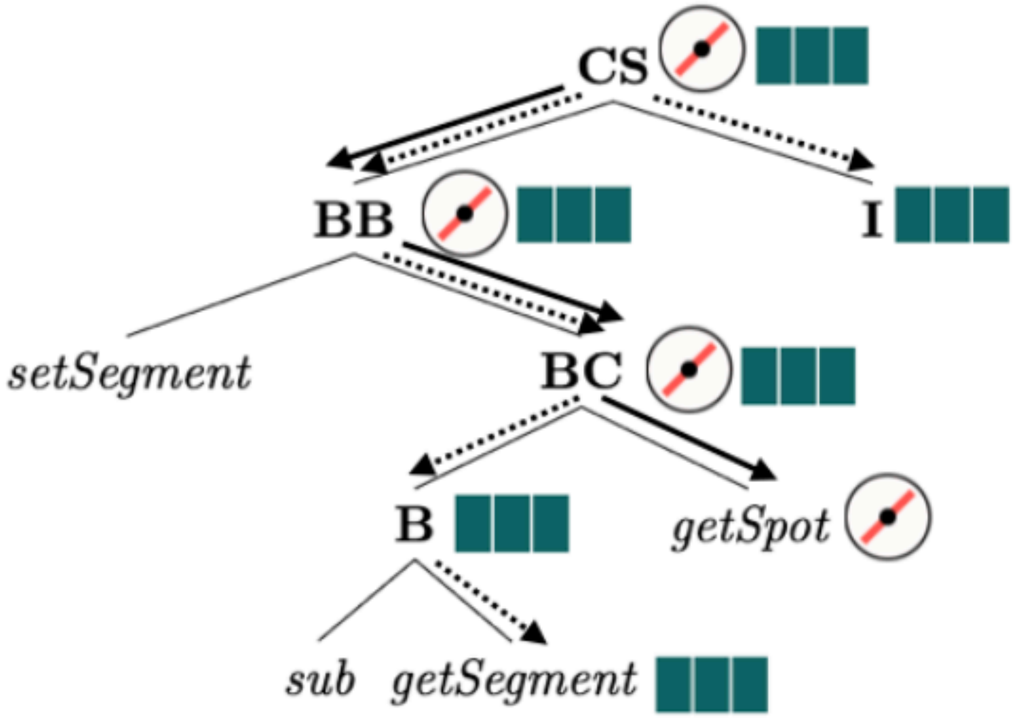
However, sometimes the desired unit of sharing does not appear as a subtree. For example, functions $x \mapsto x^2 + 1$ and $x \mapsto x^2 - 1$ have identical trees except for one leaf (which is $+$ or $-$). To address this, we can pull the $+$ ($-$) leaf to the top, augmenting the routers along the path. Figure 1(c) is the result of this refactoring operation. The left subtree of the root now denotes a higher-order function, which when applied to $+$, produces the function $x \mapsto x^2 + 1$. Refactoring creates new sharable modular subprograms, analogous to how a good programmer might.

Task 1 examples: $X_{11} = (3, 2), Y_{11} = 2, X_{12} = (7, 4), Y_{12} = 4$
Task 2 examples: $X_{21} = (3, 6), Y_{21} = 6, X_{22} = (2, 4), Y_{22} = 4$



(Liang et al., 2010)

c



$S \rightarrow \text{add}(A, A) \mid \text{sub}(A, A) \mid \text{mult}(A, A)$

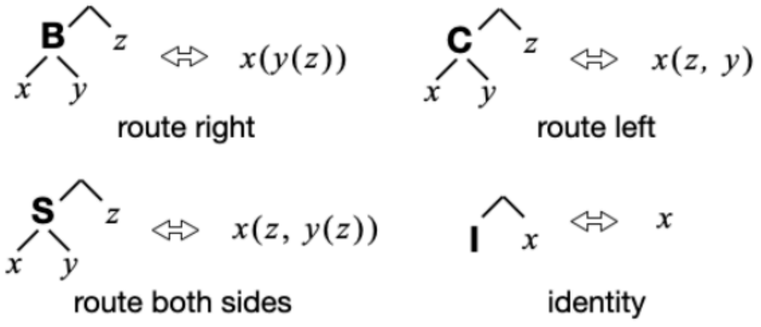
$A \rightarrow S \mid B$

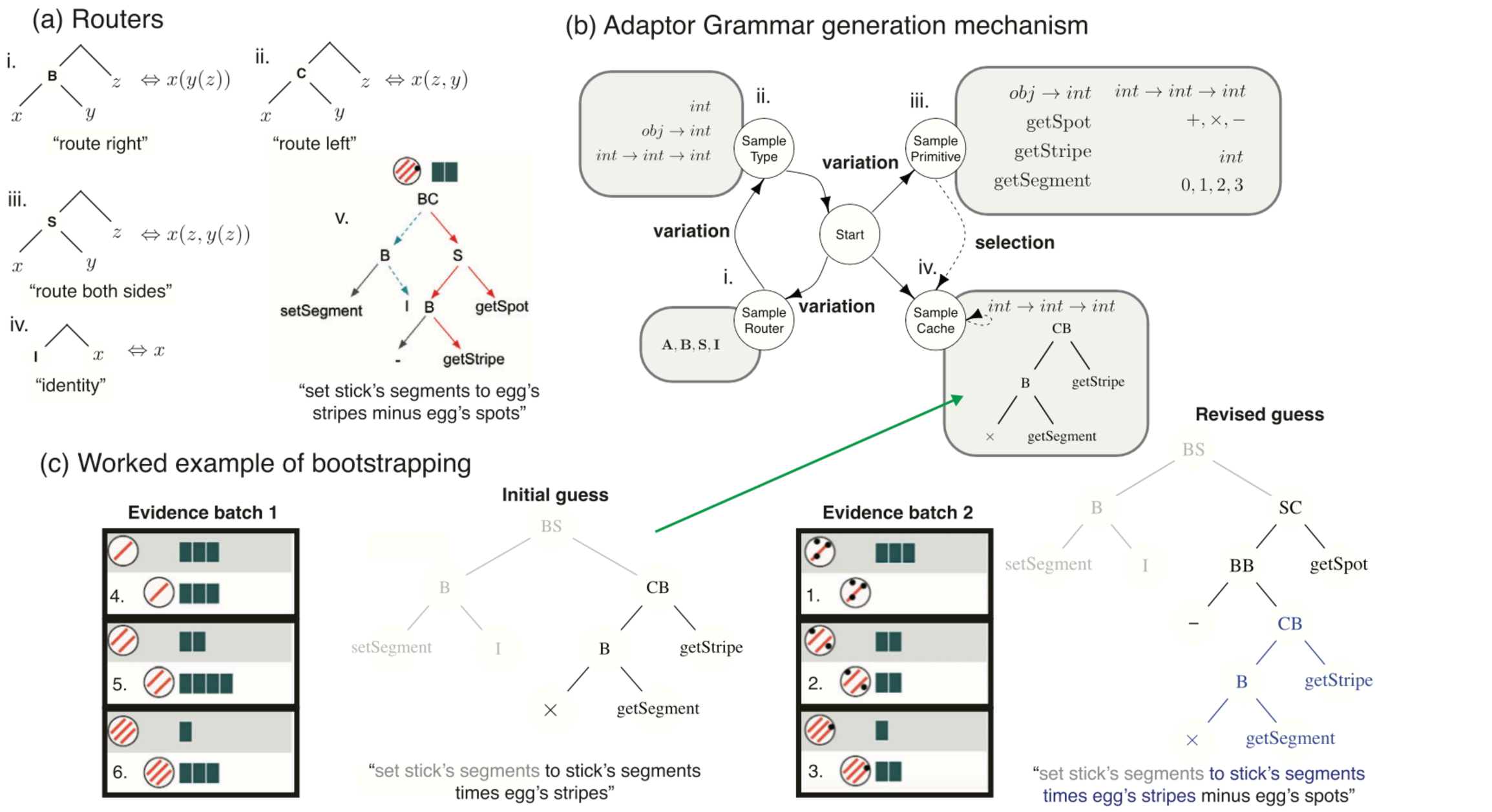
$B \rightarrow C \mid D$

$C \rightarrow \text{stripe} \mid \text{spot} \mid \text{segment}$

$D \rightarrow 0 \mid 1 \mid 2 \mid 3$

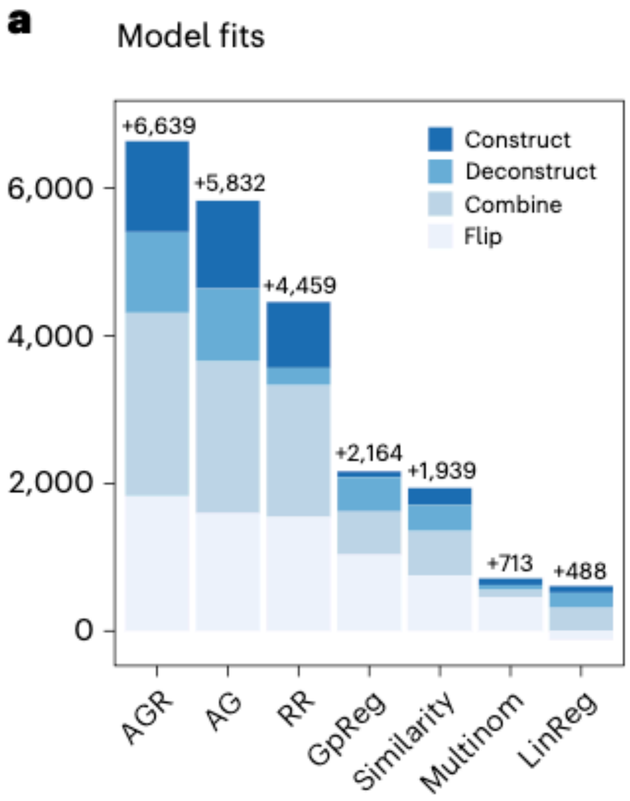
b



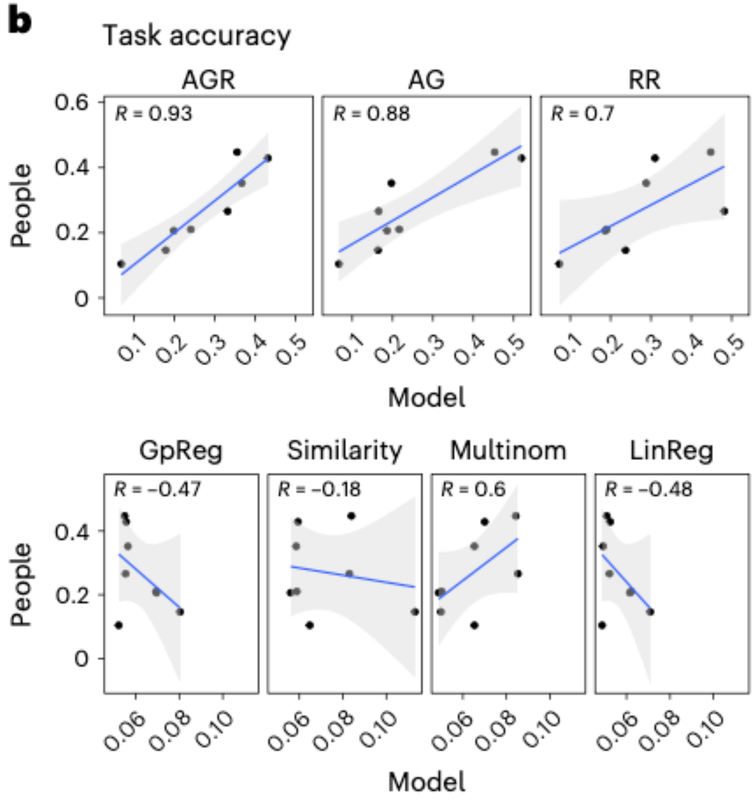


(Bramley et al., 2023)

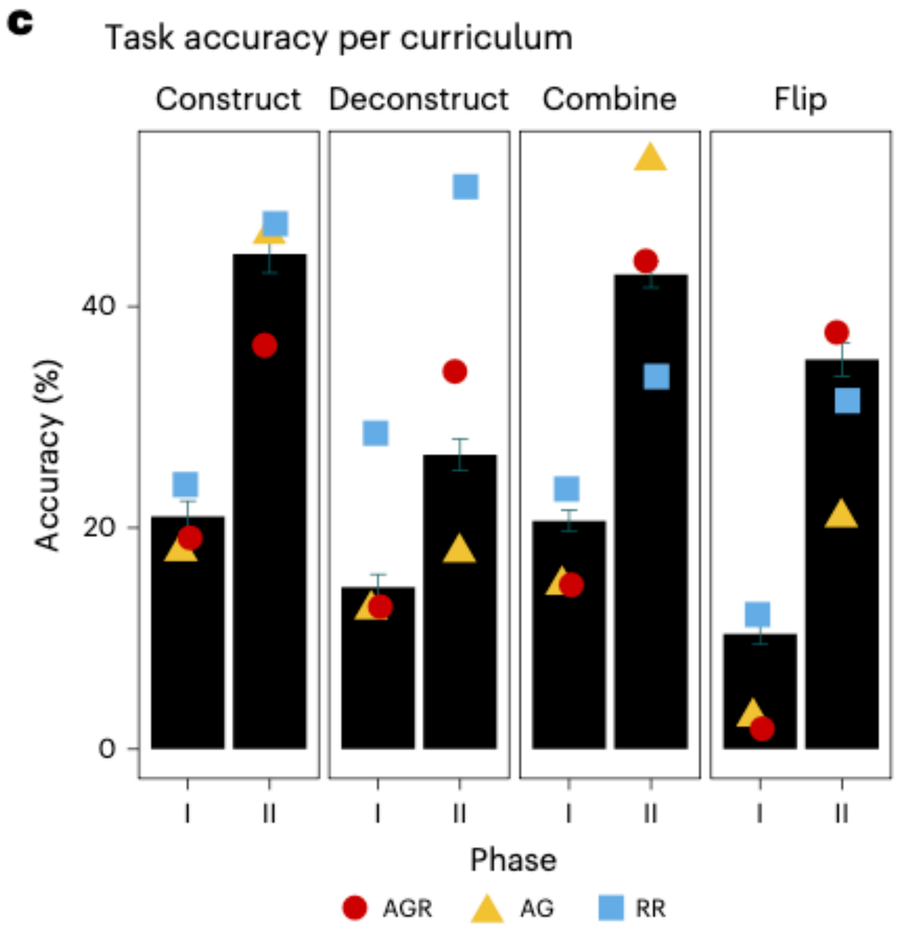
model comparison

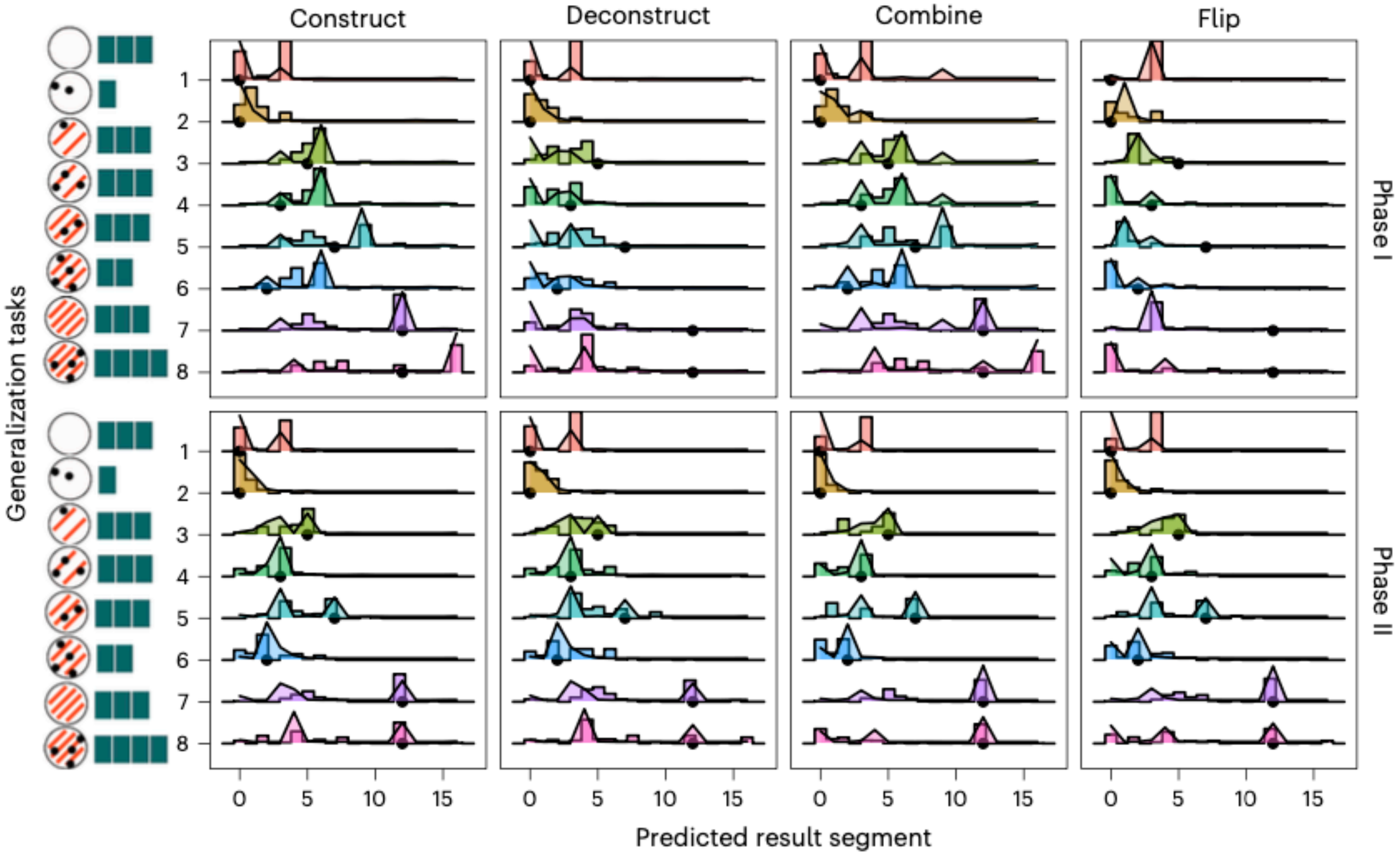


AGR: $\hat{y}_r \propto \theta \times \hat{y}_{\rightarrow} + (1-\theta) \times \hat{y}_{\leftarrow}$



model comparison




Construct
Deconstruct
Combine
Flip
Phase I
Phase II
Generalization tasks
1
2
3
4
5
6
7
8
0
5
10
15
Predicted result segment

models of computation

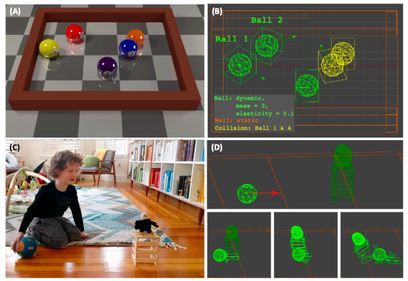
Turing machines

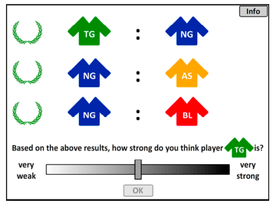
λ-calculus

Combinatory logic

Church (ψλ-calculus)

this is something else





Kinetic Energy

$$KE = \frac{1}{2} m |\vec{v}|^2$$

```
(ab/2 m (|v|^2 v))
```

Coulomb's Law

$$\vec{F} \propto \frac{q_1 q_2}{|\vec{r}_1 - \vec{r}_2|^2} \widehat{\vec{r}_1 - \vec{r}_2}$$

```
(inverse-square q_1 q_2
(subtract-vectors r_1 r_2))
```

```
(λ (x y z u) (map (λ (v) (* (/
(* (power (/ (* x x) (fold (zip
z u (λ (w a) (- w a))) 0 (λ (b
c) (+ (* b b)    c)))) (/ (* 1
1) (+ 1 1))) y) (fold (zip z u
(λ (d e)  (- d e))) 0 (λ (f g)
(+ (* f f) g)))) v)) (zip z u
(λ (h i) (- h i)))))
```

Solution to Coulomb's Law if expressed in initial primitives

Singular-Plural

λ *S* . (*if* (*singleton? S*)
"one"
"two")

Mod-5

λ *S* . (*if* (*or* (*singleton? S*)
(*equal-word?* (*L* (*set-difference S*)
(*select S*))
"five"))
"one"
(*next* (*L* (*set-difference S*
(*select S*)))))

2-not-1-knower

λ *S* . (*if* (*doubleton? S*)
"two"
undef)

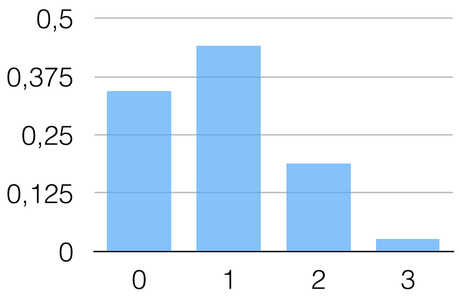
2N-knower

λ *S* . (*if* (*singleton? S*)
"one"
(*next* (*next* (*L* (*set-difference S* (*select S*))))))

```
a = flip(0.3)
b = flip(0.3)
c = flip(0.3)
a + b + c
```

1 0 0
0 0 0
1 0 1
2 0 1

sampling



$$P(n) = \binom{3}{n} 0.3^n 0.7^{3-n}$$

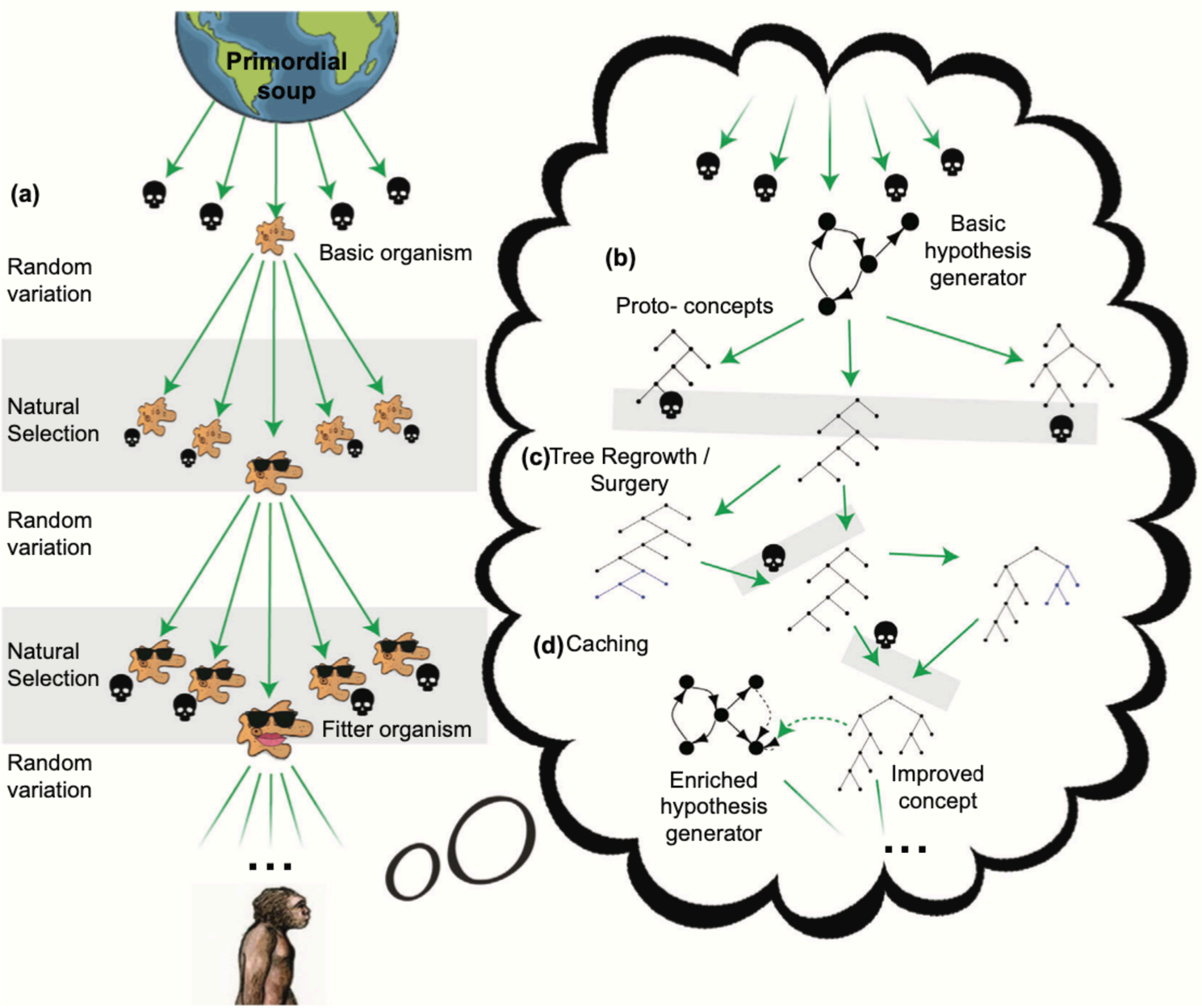
distribution

- Turing-complete language
- random choice operator

```
flip (0.5)
```

0 or 1

- conditioning (inference) as language primitive


Primordial soup
(a)
Random variation
Basic organism
Natural Selection
Random variation
Natural Selection
Fitter organism
Random variation
...
(b)
Basic hypothesis generator
Proto- concepts
(c)Tree Regrowth / Surgery
(d)Caching
Enriched hypothesis generator
Improved concept
...

Tree regrowth / Tree surgery