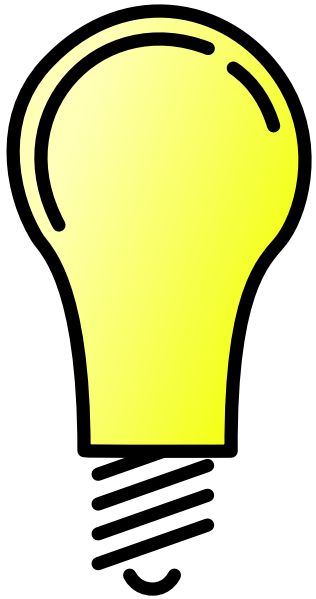
Linguistic modularity and side effects

Chung-chieh Shan
October 26, 2012

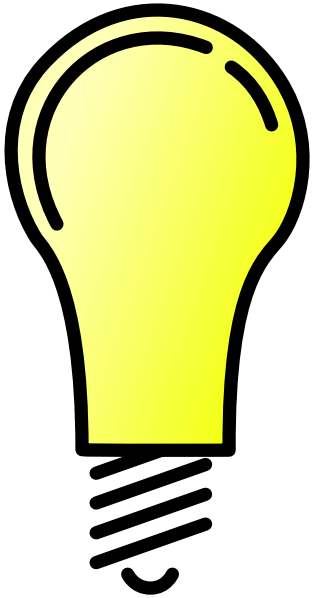
Human concepts

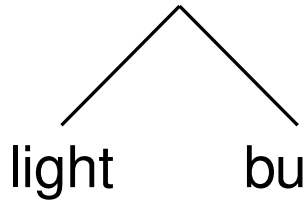
Line up representations and what they represent

Functional modularity



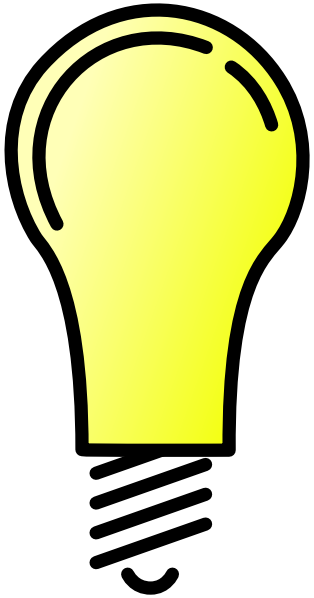
Functional modularity

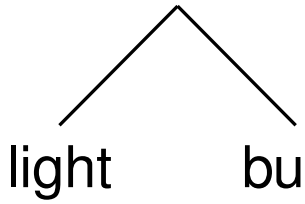




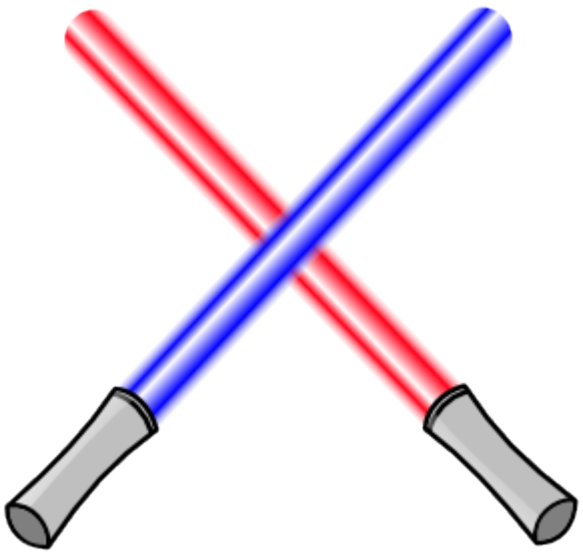

```
object {sphere ...
  interior {media {emission <.4,.3,.2>}}}
object {cylinder ...}
```

Functional modularity

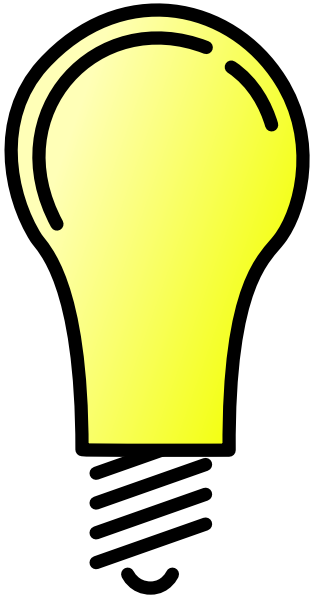


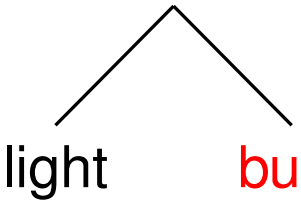



```
object {sphere ...
  interior {media {emission <.4,.3,.2>}}}
object {cylinder ...}
```

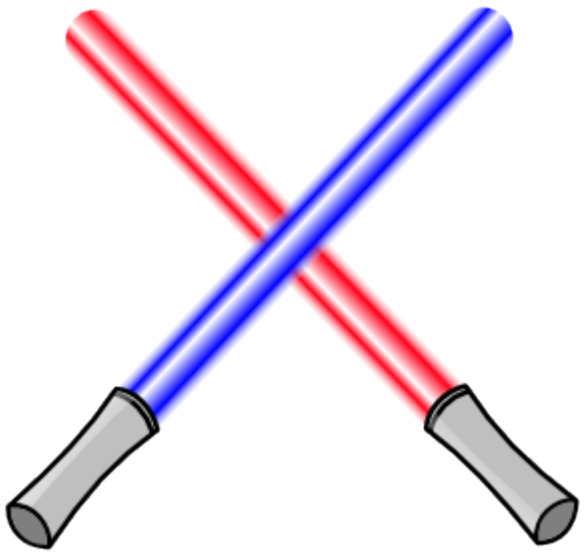


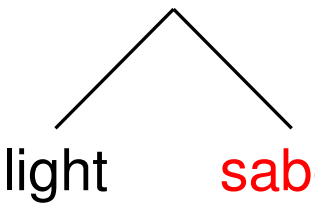
Functional modularity






```
object {sphere ...
  interior {media {emission <.4,.3,.2>}}}
object {cylinder ...}
```






```
object {cone ...
  interior {media {emission <.4,.3,.2>}}}
object {sphere ...
  interior {media {emission <.4,.3,.2>}}}
object {cylinder ...}
```

A.I. Memo 357

May 1976

FROM UNDERSTANDING COMPUTATION TO UNDERSTANDING NEURAL CIRCUITRY

by

D. Marr and T. Poggio*

Abstract: The CNS needs to be understood at four nearly independent levels of description: (1) that at which the nature of a computation is expressed; (2) that at which the algorithms that implement a computation are characterized; (3) that at which an algorithm is committed to particular mechanisms; and (4) that at which the mechanisms are realized in hardware. In general, the nature of a computation is determined by the problem to be solved, the mechanisms that are used depend upon the available hardware, and the particular algorithms chosen depend on the problem and on the available mechanisms. Examples are given of theories at each level.

On the Design and Development of Program Families

DAVID L. PARNAS

One may consider a program development to be good, if the early decisions exclude only uninteresting, undesired, or unnecessary programs. The decisions which remove desired programs would be either postponed until a later stage or confined to a well delimited subset of the code. Objective criticism of a program's structure would be based upon the fact that a decision or assumption which was likely to change has influenced too much of the code either because it was made too early in the development or because it was not confined to an information hiding module.

Clearly this is not the only criterion which one may use in evaluating program structures. Clarity (e.g., ease of understanding, ease of verification) is another quite relevant consideration. Although there is some reason to suspect that the two measures are not completely unrelated, there are no rea-

Today: Side effects

Natural languages $\xleftarrow{\text{Side effects}}$ **Programming languages**

entailment from corpora	probabilistic inference
mixed quotation	portable performance
⋮	⋮

- Studies of language
- Examples of side effects
  - State
  - Quantificationl/control
- Generalizations across side effects
  - Order matters
  - Apparent noncompositionality
- Treatments of side effects
  - Operational semantics
  - Denotational semantics

Today: Side effects

Natural languages $\xleftarrow{\text{Side effects}}$ **Programming languages**

entailment from corpora
mixed quotation
⋮

probabilistic inference
portable performance
⋮

- Studies of language
- Examples of side effects
  - State
  - Quantificationl/control
- Generalizations across side effects
  - Order matters
  - Apparent noncompositionality
- Treatments of side effects
  - Operational semantics
  - Denotational semantics

?

Describe **(is)** vs prescribe **(ought)**

How do natural languages work?

How should programming languages work?

Describe **(is)** vs prescribe **(ought)**

How do natural languages work?
How do people learn to speak?

How should programming languages work?
How should computers be designed?

Describe **(is)** vs prescribe **(ought)**

How do natural languages work?
How do people learn to speak?
How do people understand utterances?

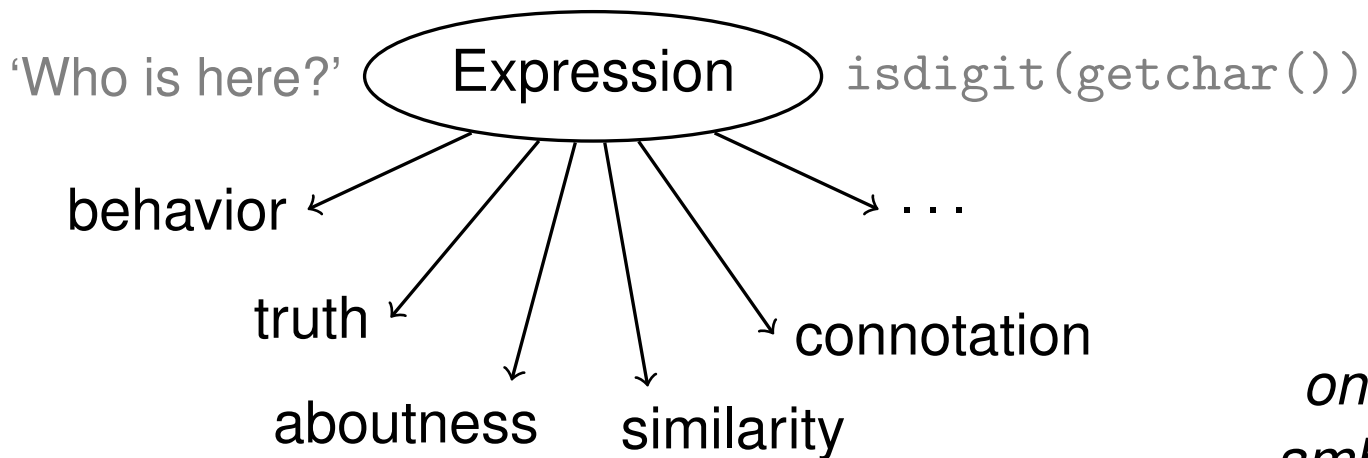
How should programming languages work?
How should computers be designed?
How should computers run programs?

Describe **(is)** vs prescribe **(ought)**

How do natural languages work?
How do people learn to speak?
How do people understand utterances?

How should programming languages work?
How should computers be designed?
How should computers run programs?

‘Semantics’



ontology,
ambiguity,
simplicity.

What is the total population of the ten largest capitals in the US?
Answering these types of complex questions compositionally involves first mapping the questions into logical forms (semantic parsing).

Liang, Jordan & Klein

What is the total population of the ten largest capitals in the US?
Answering these types of complex questions compositionally involves first mapping the questions into logical forms (semantic parsing).

The filtering function F rules out improperly-typed trees . . .
To further reduce the search space . . .

Think of DCS as a higher-level programming language tailored to natural language, which results in programs which are much simpler than the logically-equivalent lambda calculus formulae.

Liang, Jordan & Klein

Side effects

John
The man sitting there
A man
Every man
No man

} loves his mother.

Which man loves his mother?

John doesn't think anyone turned off the damn gas.

```
c = getchar();
putchar(c);
...isdigit(c)...
```

衆聲

探象之圖

Order matters

* His mother loves { a man. / every man. / no man.

* Which man does his mother love _?

  Every man loves a woman.

  Who do you think _ loves who in the story?
* Who do you think who loves _ in the story?

```
putchar(c);
c = getchar();
```

```
putchar(getchar());
```

All side effects interact within one evaluation order!

State, operationally

$$1 + 2 + 3 + \cdots + 100 = ?$$

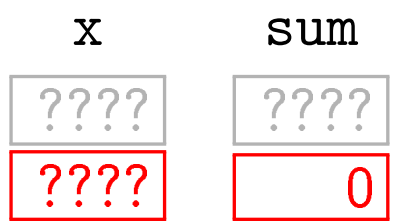
x	sum
????	????

```
sum = 0;
x = 1;
while (x ≤ 100) {
    sum = sum + x;
    x = x + 1;
}
```

State, operationally

$$1 + 2 + 3 + \cdots + 100 = ?$$

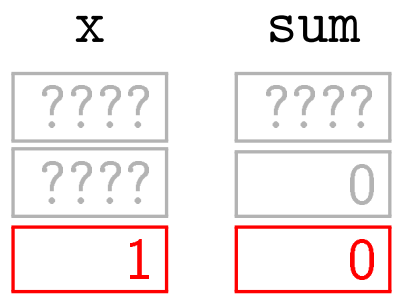
```
sum = 0;
x = 1;
while (x ≤ 100) {
    sum = sum + x;
    x = x + 1;
}
```



State, operationally

$$1 + 2 + 3 + \cdots + 100 = ?$$

```
sum = 0;
x = 1;
while (x ≤ 100) {
    sum = sum + x;
    x = x + 1;
}
```



State, operationally

$$1 + 2 + 3 + \cdots + 100 = ?$$

```
sum = 0;
x = 1;
sum = sum + x;
x = x + 1;
while (x ≤ 100) {
    sum = sum + x;
    x = x + 1;
}
```

x	sum
????	????
????	0
1	0

State, operationally

$$1 + 2 + 3 + \cdots + 100 = ?$$

```
sum = 0;
x = 1;
sum = sum + x;
x = x + 1;
while (x ≤ 100) {
    sum = sum + x;
    x = x + 1;
}
```

x	sum
????	????
????	0
1	0
1	1

State, operationally

$$1 + 2 + 3 + \cdots + 100 = ?$$

```
sum = 0;
x = 1;
sum = sum + x;
x = x + 1;
while (x ≤ 100) {
    sum = sum + x;
    x = x + 1;
}
```

x	sum
????	????
????	0
1	0
1	1
2	1

State, operationally

$$1 + 2 + 3 + \cdots + 100 = ?$$

```
sum = 0;
x = 1;
sum = sum + x;
x = x + 1;
sum = sum + x;
x = x + 1;
while (x ≤ 100) {
    sum = sum + x;
    x = x + 1;
}
```

x	sum
????	????
????	0
1	0
1	1
2	1

State, operationally

$$1 + 2 + 3 + \cdots + 100 = ?$$

```
sum = 0;
x = 1;
sum = sum + x;
x = x + 1;
sum = sum + x;
x = x + 1;
while (x ≤ 100) {
    sum = sum + x;
    x = x + 1;
}
```

x	sum
????	????
????	0
1	0
1	1
2	1
2	3

State, operationally

$$1 + 2 + 3 + \cdots + 100 = ?$$

```
sum = 0;
x = 1;
sum = sum + x;
x = x + 1;
sum = sum + x;
x = x + 1;
while (x ≤ 100) {
    sum = sum + x;
    x = x + 1;
}
```

x	sum
????	????
????	0
1	0
1	1
2	1
2	3
3	3

State, operationally

$$1 + 2 + 3 + \cdots + 100 = ?$$

```
sum = 0;
x = 1;
sum = sum + x;
x = x + 1;
sum = sum + x;
x = x + 1;
sum = sum + x;
x = x + 1;
while (x ≤ 100) {
    sum = sum + x;
    x = x + 1;
}
```

x	sum
????	????
????	0
1	0
1	1
2	1
2	3
3	3

State, operationally

$$1 + 2 + 3 + \cdots + 100 = ?$$

```
sum = 0;
x = 1;
sum = sum + x;
x = x + 1;
sum = sum + x;
x = x + 1;
sum = sum + x;
x = x + 1;
while (x ≤ 100) {
    sum = sum + x;
    x = x + 1;
}
```

x	sum
????	????
????	0
1	0
1	1
2	1
2	3
3	3
3	6

State, operationally

$$1 + 2 + 3 + \cdots + 100 = ?$$

```
sum = 0;
x = 1;
sum = sum + x;
x = x + 1;
sum = sum + x;
x = x + 1;
sum = sum + x;
x = x + 1;
while (x ≤ 100) {
    sum = sum + x;
    x = x + 1;
}
```

x	sum
????	????
????	0
1	0
1	1
2	1
2	3
3	3
3	6
4	6

State, operationally

$$1 + 2 + 3 + \cdots + 100 = ?$$

```
sum = 0;
x = 1;
sum = sum + x;
x = x + 1;
sum = sum + x;
x = x + 1;
sum = sum + x;
x = x + 1;
while (x ≤ 100) {
    sum = sum + x;
    x = x + 1;
}
```

x	sum
????	????
????	0
1	0
1	1
2	1
2	3
3	3
3	6
4	6
101	5050

(order matters)

State, operationally

Mitt loves his mother. She loves him too.

As if checking or updating a database.

State, operationally

Mitt loves his mother. She loves him too.

i loves his mother. She loves him too.

i
$\mathsf{mitt}(i)$

As if checking or updating a database.

State, operationally

Mitt loves his mother. She loves him too.
i loves his mother. She loves him too.
i loves j. She loves him too.

i j
mitt(i) mother(j, i)

As if checking or updating a database.

State, operationally

Mitt loves his mother. She loves him too.
i loves his mother. She loves him too.
i loves j. She loves him too.
She loves him too.

i j ℓ_1
mitt(i)
mother(j, i)
love(ℓ_1, i, j)

As if checking or updating a database.

State, operationally

Mitt loves his mother. She loves him too.
i loves his mother. She loves him too.
i loves j. She loves him too.
She loves him too.
j loves him too.

i j ℓ_1
mitt(i)
mother(j, i)
love(ℓ_1, i, j)

As if checking or updating a database.

State, operationally

Mitt loves his mother. She loves him too.
i loves his mother. She loves him too.
i loves j. She loves him too.
She loves him too.
j loves him too.
j loves i too.

i j ℓ_1
mitt(i)
mother(j, i)
love(ℓ_1, i, j)

As if checking or updating a database.

State, operationally

Mitt loves his mother. She loves him too.
i loves his mother. She loves him too.
i loves j. She loves him too.
She loves him too.
j loves him too.
j loves i too.

i j ℓ_1 ℓ_2
mitt(i)
mother(j, i)
love(ℓ_1, i, j)
love(ℓ_2, j, i)

As if checking or updating a database.

State, operationally

Mitt loves his mother. She loves him too.
i loves his mother. She loves him too.
i loves j. She loves him too.
She loves him too.
j loves him too.
j loves i too.

i j ℓ_1 ℓ_2
mitt(i)
mother(j, i)
love(ℓ_1, i, j)
love(ℓ_2, j, i)

As if checking or updating a database.

We like feeling dynamic and orderly.

Apparent noncompositionality

Mitt loves his mother, and Paul does too.
Mitt loves Mitt's mother, and Paul does too.

No one is better than God.
The devil is better than no one.

```
c = getchar();
while (isdigit(c)) {
    putchar(c);
    c = getchar();
}
```

```
...(random() + random())...
x = random(); ...(x + x)...
```

Unsound reasoning 'joke' based on state?

State, denotationally

$$[\![\texttt{1 <= 100}]\!] = \mathsf{true}$$

$$[\![\texttt{1 + 2}]\!] = 3$$

$$[\![\texttt{x <= 100}]\!] = ???$$

$$[\![\texttt{sum = sum + x}]\!] = ???$$

State, denotationally

'Lift' denotations from type X to type $\mathsf{store} \to (\mathsf{store} \times X)$.

$$[\![1 \ \leq \ 100]\!] = \lambda s.\, \langle s,\, \mathsf{true} \rangle$$

$$[\![1 \ + \ 2]\!] = \lambda s.\, \langle s,\, 3 \rangle$$

$$[\![\mathtt{x} \ \leq \ 100]\!] = \lambda s.\, \langle s,\, s(\mathtt{x}) \leq 100 \rangle$$

$$[\![\mathtt{sum} \ = \ \mathtt{sum} \ + \ \mathtt{x}]\!] = \lambda s.\, \langle s[\mathtt{sum} := s(\mathtt{sum}) + s(\mathtt{x})],\, s(\mathtt{sum}) + s(\mathtt{x}) \rangle$$

State, denotationally

'Lift' denotations from type X to type $\mathsf{store} \to (\mathsf{store} \times X)$.

$$[\![1 \leq 100]\!] = \lambda s.\langle s, \mathsf{true}\rangle$$
$$[\![1 + 2]\!] = \lambda s.\langle s, 3\rangle$$
$$[\![\mathtt{x} \leq 100]\!] = \lambda s.\langle s, s(\mathtt{x}) \leq 100\rangle$$
$$[\![\mathtt{sum = sum + x}]\!] = \lambda s.\langle s[\mathtt{sum} := s(\mathtt{sum}) + s(\mathtt{x})], s(\mathtt{sum}) + s(\mathtt{x})\rangle$$

Also for natural language, but often with baked-in nondeterminism (e.g., Muskens 1996 'Compositional DRT').

$$[\![\text{his mother}]\!] \approx \lambda s.\langle s[j := \text{mother}(s(i))], j\rangle$$

We like feeling meaningful and not just tree hacking.

Relating operational and denotational semantics

Programming languages have *both*

operational semantics (‘transformational syntax’)
so we feel dynamic and orderly

denotational semantics (‘model-theoretic semantics’)
so we feel meaningful and not just tree hacking

Don’t choose—relate! ‘adequacy’, ‘full abstraction’

Another example of relating the two semantics:
quantification/control . . .

Beyond state: Quantification

A man / Every man / No man } loves his mother.

* His mother loves { a man. / every man. / no man.

Which man loves his mother?

* Which man does his mother love _?

Every man loves a woman.

Who do you think _ loves who in the story?

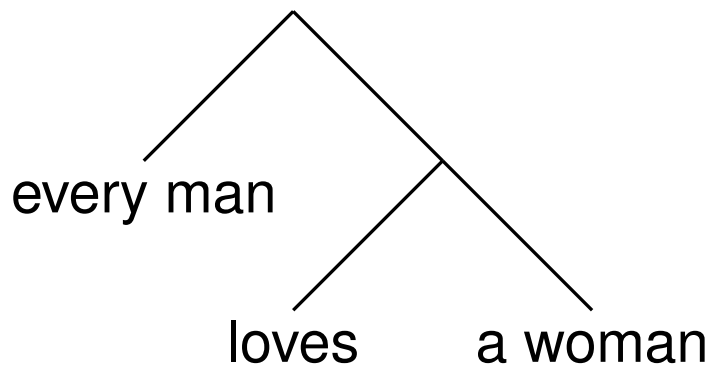
* Who do you think who loves _ in the story?

‘In-situ’ ‘nondeterminism’

Quantification, operationally

Every man loves a woman.

$$\forall i.\, (\mathsf{man}(i) \Rightarrow \exists j.\, (\mathsf{woman}(j) \wedge \mathsf{love}(i, j)))$$

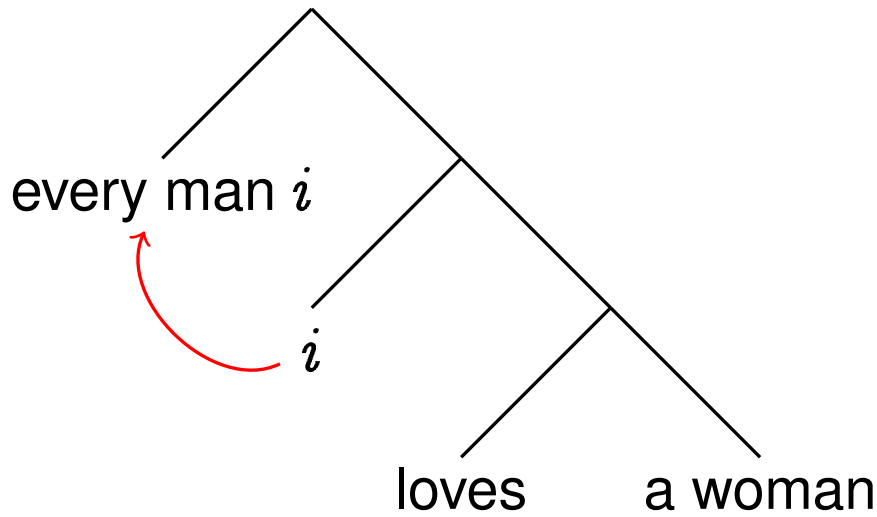


As if checking or updating a database.

Quantification, operationally

Every man loves a woman.

$$\forall i.\,(\mathsf{man}(i) \Rightarrow \exists j.\,(\mathsf{woman}(j) \wedge \mathsf{love}(i,j)))$$

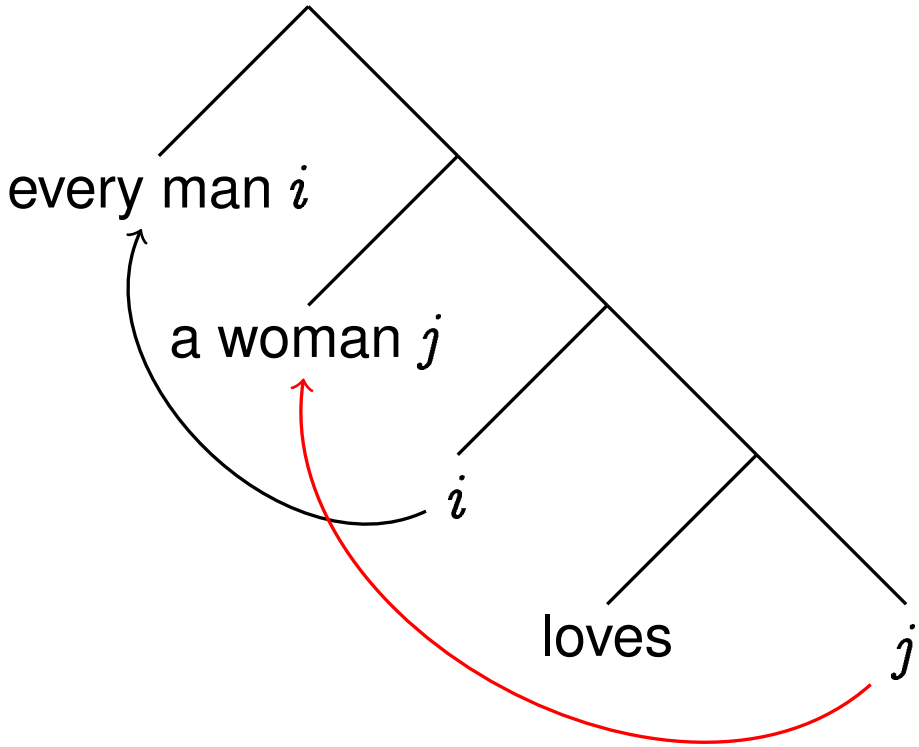


As if checking or updating a database.

Quantification, operationally

Every man loves a woman.

$$\forall i.\,(\mathsf{man}(i) \Rightarrow \exists j.\,(\mathsf{woman}(j) \wedge \mathsf{love}(i,j)))$$

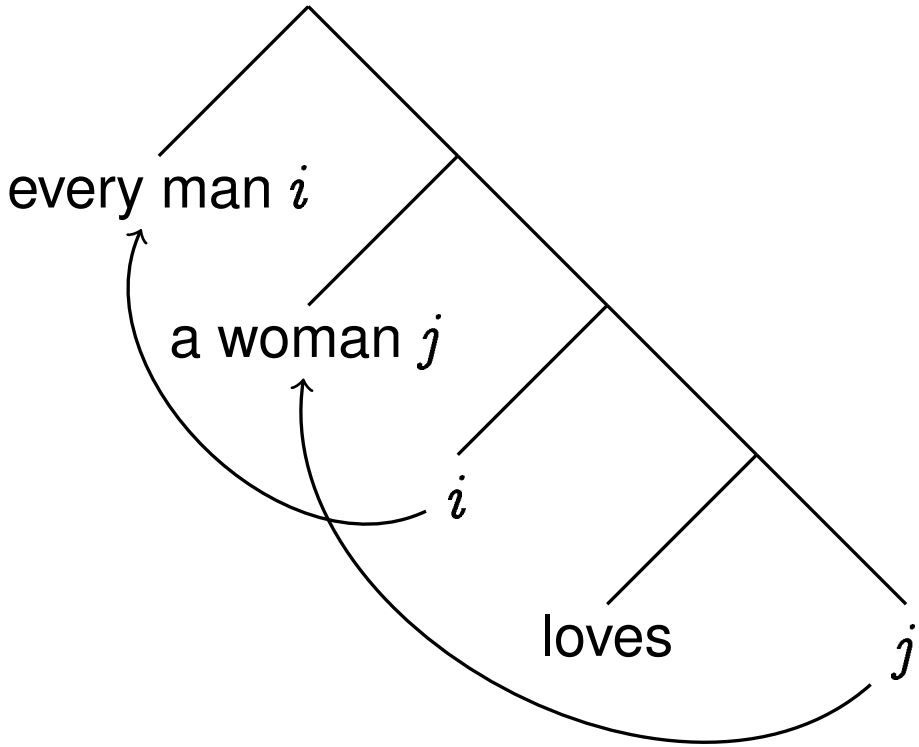


As if checking or updating a database.

Quantification, operationally

Every man loves a woman.

$$\forall i.\,(\mathsf{man}(i) \Rightarrow \exists j.\,(\mathsf{woman}(j) \wedge \mathsf{love}(i,j)))$$



As if checking or updating a database.

Every man loves his mother.
Someone from every city likes it.

Control, operationally

‘Both 1 and 3 are less than either 2 or 4.’

```
reset {(shift a. a(1) ∧ a(3)) < (shift b. b(2) ∨ b(4))}
```

Shuffle/reinstate/duplicate ‘the rest of the computation’

Control, operationally

‘Both 1 and 3 are less than either 2 or 4.’

```
reset {(shift a. a(1) ∧ a(3)) < (shift b. b(2) ∨ b(4))}
```

```
reset {1 < (shift b. b(2) ∨ b(4))} ∧
reset {3 < (shift b. b(2) ∨ b(4))}
```

Shuffle/reinstate/duplicate ‘the rest of the computation’

Control, operationally

‘Both 1 and 3 are less than either 2 or 4.’

```
reset {(shift a. a(1) ∧ a(3)) < (shift b. b(2) ∨ b(4))}
reset {1 < (shift b. b(2) ∨ b(4))} ∧
reset {3 < (shift b. b(2) ∨ b(4))}
(reset {1 < 2} ∨ reset {1 < 4}) ∧
reset {3 < (shift b. b(2) ∨ b(4))}
```

Shuffle/reinstate/duplicate ‘the rest of the computation’

Control, operationally

‘Both 1 and 3 are less than either 2 or 4.’

```
reset {(shift a. a(1) ∧ a(3)) < (shift b. b(2) ∨ b(4))}
```

```
reset {1 < (shift b. b(2) ∨ b(4))} ∧
reset {3 < (shift b. b(2) ∨ b(4))}
```

```
(reset {1 < 2} ∨ reset {1 < 4}) ∧
reset {3 < (shift b. b(2) ∨ b(4))}
```

Shuffle/reinstate/duplicate ‘the rest of the computation’

Control, operationally

‘Both 1 and 3 are less than either 2 or 4.’

```
reset {(shift a. a(1) ∧ a(3)) < (shift b. b(2) ∨ b(4))}
reset {1 < (shift b. b(2) ∨ b(4))} ∧
reset {3 < (shift b. b(2) ∨ b(4))}
(reset {1 < 2} ∨ reset {1 < 4}) ∧
reset {3 < (shift b. b(2) ∨ b(4))}
```

Shuffle/reinstate/duplicate ‘the rest of the computation’

Useful: input/output, suspend/resume, backtracking search, probabilistic programming, code generation, classical logic, . . .

Common: `fork`/`exit`, generators, (restartable) exceptions, . . .

Obscure? but natural language offers intuitions, applications!

Quantification and control, denotationally

$$[\![\texttt{1 } \leq \texttt{ 100}]\!] = \mathsf{true}$$

$$[\![\texttt{1 + 2}]\!] = 3$$

$$[\![\texttt{shift c. c(1)} \wedge \texttt{c(3)}]\!] = ???$$

$$[\![\text{a woman}]\!] = ???$$

Quantification and control, denotationally

'Lift' denotations from type X to type $\overbrace{(X \to R)}^{\text{continuation}} \to R$.

$$[\![\texttt{1 <= 100}]\!] = \lambda c.\, c(\mathsf{true})$$
$$[\![\texttt{1 + 2}]\!] = \lambda c.\, c(3)$$
$$[\![\texttt{shift c. c(1) /\textbackslash\ c(3)}]\!] = \lambda c.\, c(1) \wedge c(3)$$
$$[\![\text{a woman}]\!] = \lambda c.\, \exists j.\, (\mathsf{woman}(j) \wedge c(j))$$

Quantification and control, denotationally

'Lift' denotations from type X to type $\overbrace{(X \to R)}^{\text{continuation}} \to R$.

$$\begin{aligned}
[\![\texttt{1 <= 100}]\!] &= \lambda c.\, c(\mathsf{true}) \\
[\![\texttt{1 + 2}]\!] &= \lambda c.\, c(3) \\
[\![\texttt{shift c. c(1) /\textbackslash\ c(3)}]\!] &= \lambda c.\, c(1) \wedge c(3) \\
[\![\text{a woman}]\!] &= \lambda c.\, \exists j.\, (\mathsf{woman}(j) \wedge c(j))
\end{aligned}$$

This connection lets us uniformly explain linguistic side effects and their interaction.

Every man loves his mother.
Someone from every city likes it.
Every farmer who owns a donkey beats it.

Ongoing work: Side effects and quotation

In New York, a man is mugged every 11 seconds.
I would now like you to meet that man.

Nobody had the frigging mind to turn off the damn gas.

Safely generate code with optimizations that reach across binders, e.g., ‘loop invariant code motion’

Today: Side effects

Natural languages $\longleftarrow$ Side effects $\longrightarrow$ **Programming languages**

entailment from corpora	probabilistic inference
mixed quotation	portable performance
⋮	⋮

- Studies of language
- Examples of side effects
  - State
  - Quantificationl/control
- Generalizations across side effects
  - Order matters
  - Apparent noncompositionality
- Treatments of side effects
  - Operational semantics
  - Denotational semantics

Describe **(is)** vs prescribe **(ought)** revisited

> *A computer language is not just a way of getting a computer to perform operations but rather . . . a novel formal medium for expressing ideas about methodology. Thus, programs must be written for people to read, and only incidentally for machines to execute.*
>
> —*Abelson & Sussman*

So much to communicate, so little time and common ground.
Human communication has adapted to only ever work when cooperative (and interactive), ambiguous (and fallible).

How do we cooperate?
How should we cooperate?

Describe **(is)** vs prescribe **(ought)** revisited

> *A computer language is not just a way of getting a computer to perform operations but rather . . . a novel formal medium for expressing ideas about methodology. Thus, programs must be written for people to read, and only incidentally for machines to execute.*
>
> —*Abelson & Sussman*

So much to communicate, so little time and common ground. Human communication has adapted to only ever work when cooperative (and interactive), ambiguous (and fallible).

How do we cooperate?
How should we cooperate?

Describe **(is)** vs prescribe **(ought)** revisited

> *A computer language is not just a way of getting a computer to perform operations but rather . . . a novel formal medium for expressing ideas about methodology. Thus, programs must be written for people to read, and only incidentally for machines to execute.*
>
> —*Abelson & Sussman*

So much to communicate, so little time and common ground.
Human communication has adapted to only ever work when cooperative (and interactive), ambiguous (and fallible).

How do we cooperate?
How should we cooperate?

Describe **(is)** vs prescribe **(ought)** revisited

> *A computer language is not just a way of getting a computer to perform operations but rather . . . a novel formal medium for expressing ideas about methodology. Thus, programs must be written for people to read, and only incidentally for machines to execute.*
>
> —*Abelson & Sussman*

So much to communicate, so little time and common ground.
Human communication has adapted to only ever work when cooperative (and interactive), ambiguous (and fallible).

How do we cooperate?
How should we cooperate?

Describe **(is)** vs prescribe **(ought)** revisited

> *A computer language is not just a way of getting a computer to perform operations but rather . . . a novel formal medium for expressing ideas about methodology. Thus, programs must be written for people to read, and only incidentally for machines to execute.*
>
> —*Abelson & Sussman*

So much to communicate, so little time and common ground.
Human communication has adapted to only ever work when cooperative (and interactive), ambiguous (and fallible).

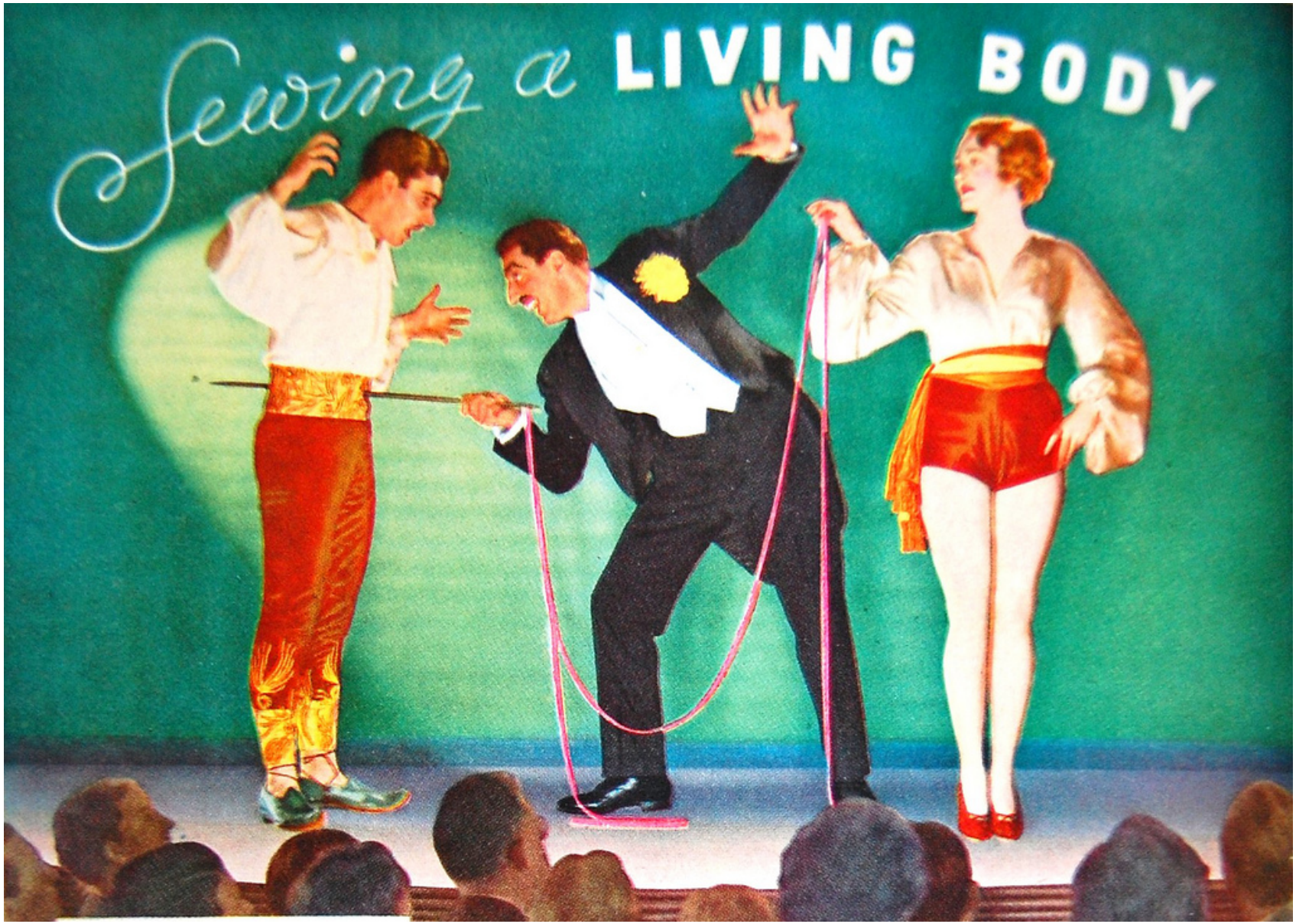
How do we cooperate?
How should we cooperate?

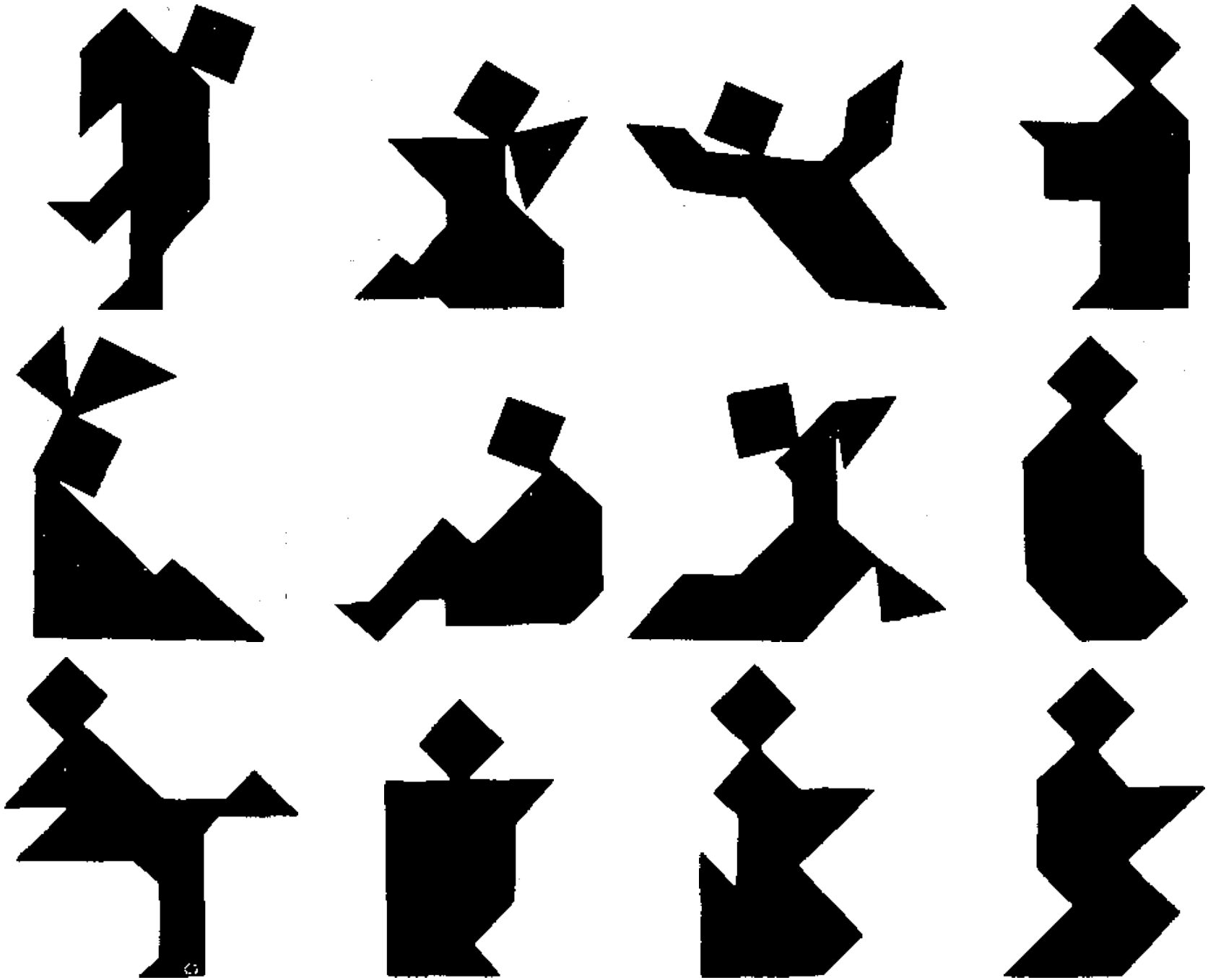
Towards a ‘just semantics’

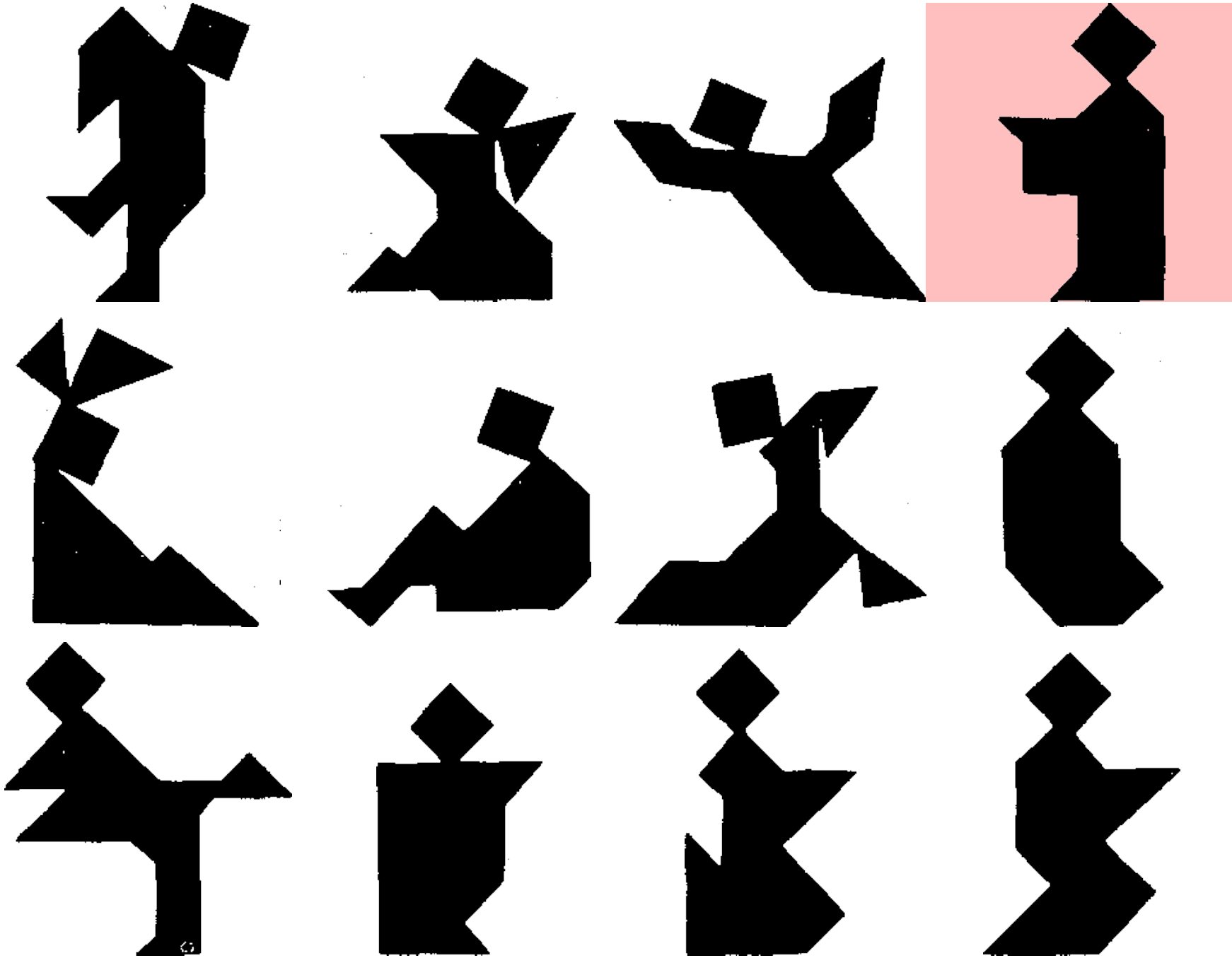
How should we cooperate?

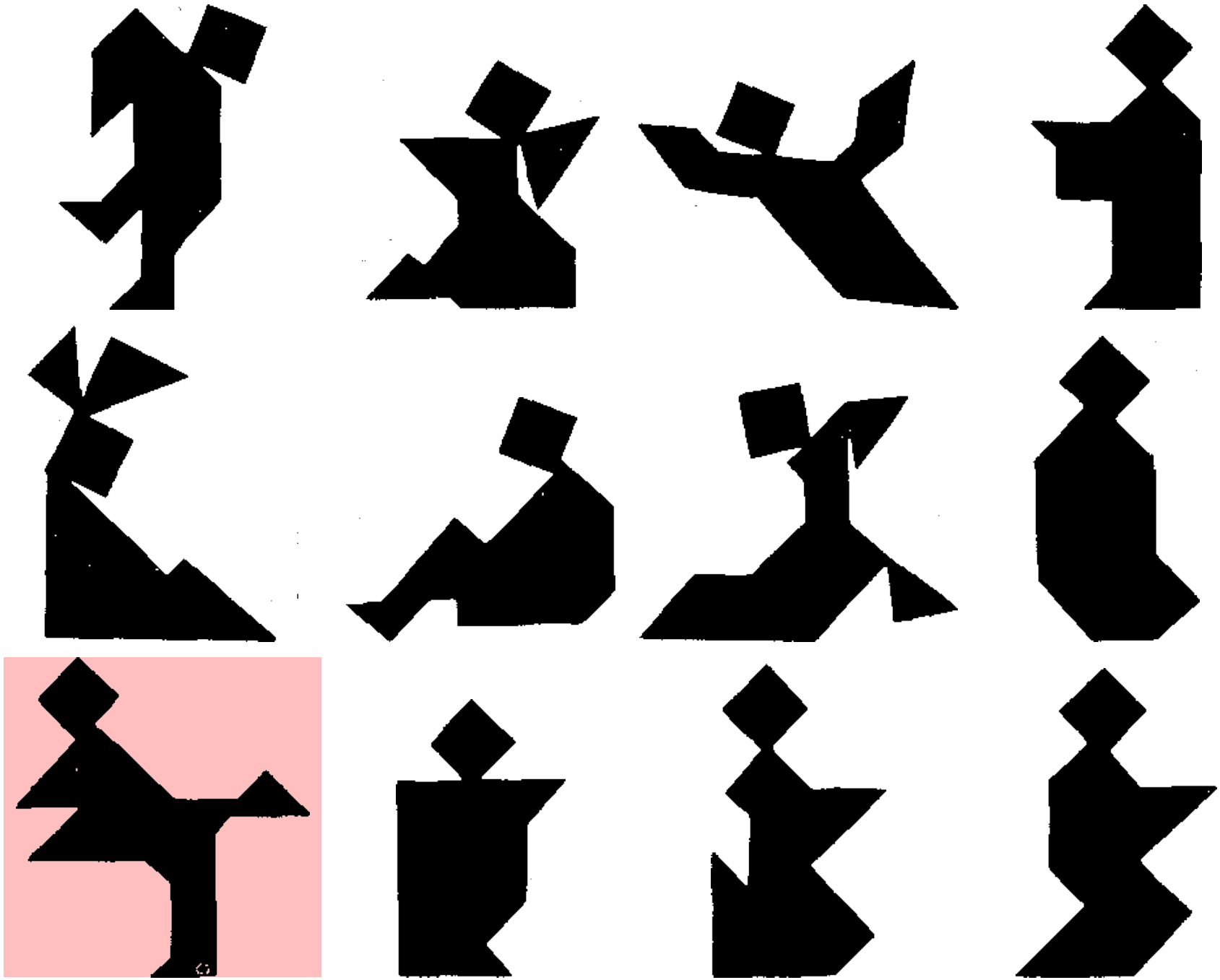
- What do we tend to want to signify?
- How hard do we have to work?
  - Motor/sensory work (hence use shorthand)
  - Cognitive work (hence be systematic)
  - Emotional work
  - Standby work
- Who adapts to whom?

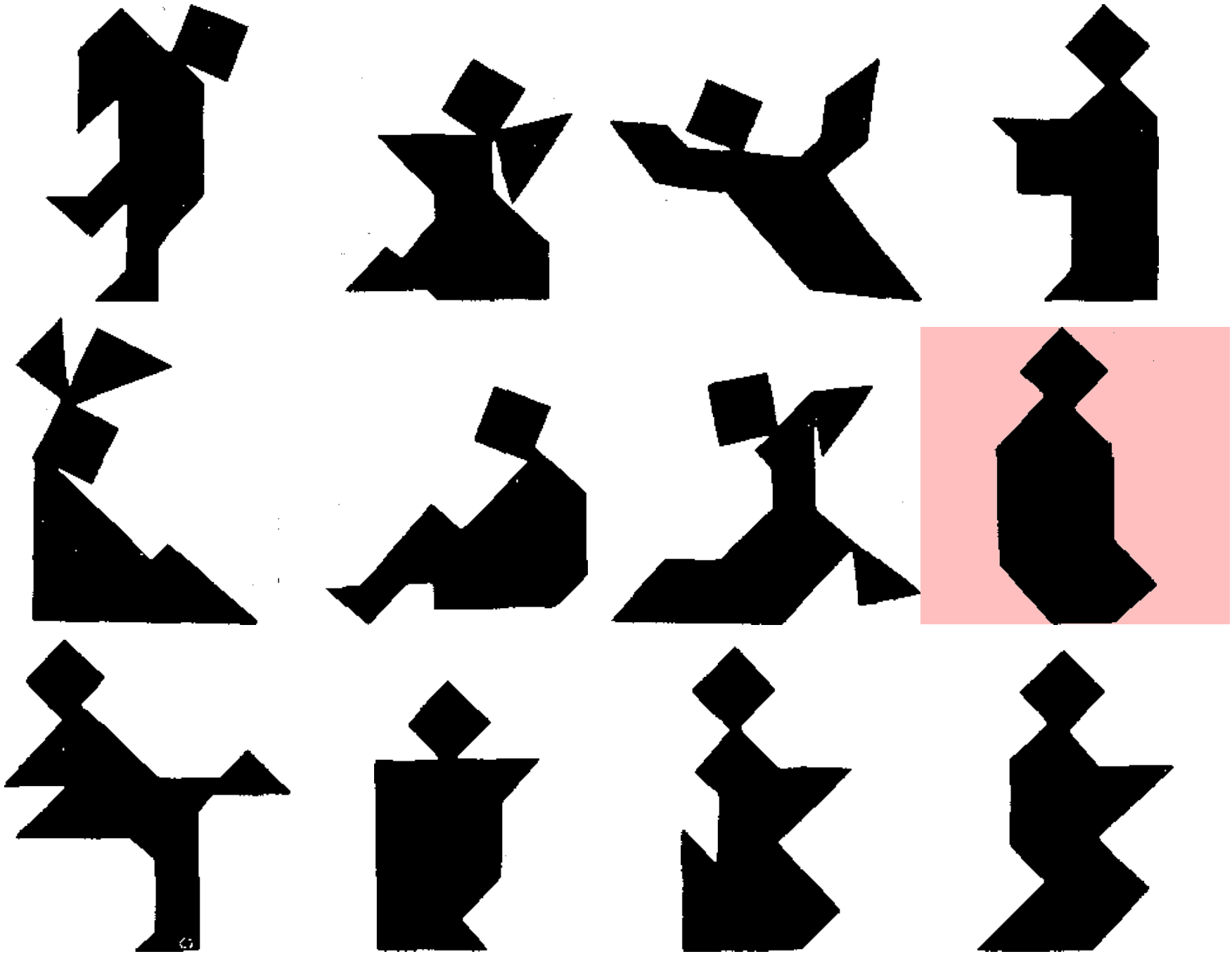
Collaborative reference

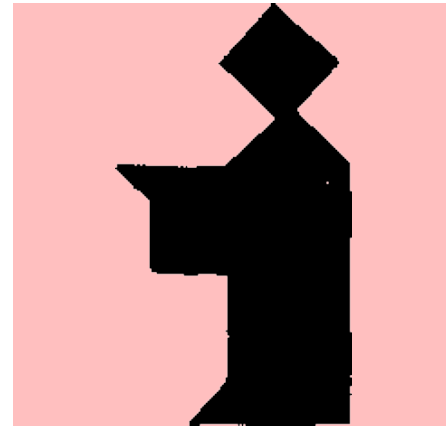








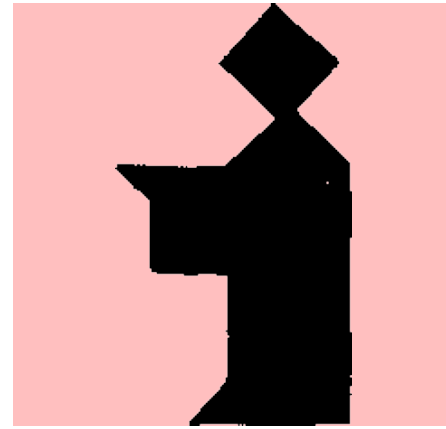




Interactive, not literary.

A: *the guy reading with, holding his book to the left.*
B: *Okay, kind of standing up?*
A: *Yeah.*
B: *Okay.*

Context and feedback!



Interactive, not literary.

A: *the guy reading with, holding his book to the left.*
B: *Okay, kind of standing up?*
A: *Yeah.*
B: *Okay.*

Context and feedback!