# Game Semantics and Infinite Games

Samson Abramsky

Oxford University

http://web.comlab.ox.ac.uk/oucl/work/samson.abramsky/

## The Game Plan

- Overview of work in Game Semantics, locating it the games landscape.
- How infinite games arise naturally in game semantics, and the distinctive game semantics perspective on them.

# Compositionality

A methodological principle from Computer Science (and Logic) of **major** potential importance for mathematical modelling throughout the sciences.

**Traditional aproach**: whole-system (monolithic) analysis of given systems. Key structuring templates, e.g. 'Find the Hamiltonian', 'Find the Nash Equilibrium', etc.

**Compositional approach**: start with a fixed set of basic (simple) building blocks, and **constructions** for building new systems out of given sub-systems, and build up the required complex system with these.

The algebraic view:

$$S \;=\; \omega(S_1, \ldots, S_n)$$

The logical view:

$$\frac{S_1 \models \phi_1, \ldots, S_n \models \phi_n}{\omega(S_1, \ldots, S_n) \models \phi}$$

## Generalities

Game Semantics is centrally concerned with the **compositional modelling of interactive systems**. This paradigm turns out to be fruitful for modelling:

- Logics (Linear, Intuitionistic, Classical, ...) and the coresponding type theories.
- Programming languages with a wide range of features (procedures, block structure, references, non-local control, concurrency, ...).

## Two faces of game semantics

- Games as a highly structured mathematical universe, an intensional setting for denotational semantics.
- Games as **concrete objects** (representable as graphs, automata etc.), suitable for algorithmic manipulation, hence providing a basis for compositional verification and program analysis.

# Basic Ideas

• **Types** of a **programming language**, or **formulas** of a **logic**, are interpreted as 2-person games: the Player is the System (program fragment, proof from assumptions) currently under consideration, while the Opponent is the Environment or context.

• **Programs**, or **proofs** in a logic, are **strategies** for these games.

So game semantics is inherently a semantics of **open systems**; the meaning of a program is given by its potential interactions with its environment.

• **Compositionality**. The key operation is plugging two strategies together, so that each **actualizes** part of the environment of the other. (Usual game idea corresponds to a **closed** system, with no residual environment). This exploits the game-theoretic P/O duality.

# Types as Games

- A simple example of a basic datatype of natural numbers:

$$\mathbf{nat} = \{q \cdot n \mid n \in \mathbb{N}\}$$

  Note a further classification of moves, orthgonal to the P/O duality; $q$ is a **question**, $n$ are **answers**. This turns out to be important for capturing **control features** of programming languages.

- Forming function or procedure types $A \Rightarrow B$. We form a new game from disjoint copies of $A$ and $B$, **with P/O roles in $A$ reversed**. Thus we think of $A \Rightarrow B$ as a **structured interface** to the Environment; in $B$, we interact with the caller of the procedure, **covariantly**, while in $A$, we interact with the argument supplied to the procedure call, **contravariantly**.

# Example

Strategy for $\lambda f : \mathbf{nat} \Rightarrow \mathbf{nat}.\, \lambda x : \mathbf{nat}.\, f(x) + 2$.

$$(\quad \mathbf{nat} \quad \Rightarrow \quad \mathbf{nat} \quad) \quad \Rightarrow \quad \mathbf{nat} \quad \Rightarrow \quad \mathbf{nat}$$

# Example

Strategy for $\lambda f : \mathbf{nat} \Rightarrow \mathbf{nat}.\, \lambda x : \mathbf{nat}.\, f(x) + 2$.

$$(\quad \mathbf{nat} \quad \Rightarrow \quad \mathbf{nat} \quad) \quad \Rightarrow \quad \mathbf{nat} \quad \Rightarrow \quad \mathbf{nat}$$

$O$ q

# Example

Strategy for $\lambda f : \mathbf{nat} \Rightarrow \mathbf{nat}.\, \lambda x : \mathbf{nat}.\, f(x) + 2$.

$$(\quad \mathbf{nat} \quad \Rightarrow \quad \mathbf{nat} \quad) \quad \Rightarrow \quad \mathbf{nat} \quad \Rightarrow \quad \mathbf{nat}$$

$O$ q

$P$ q

# Example

Strategy for $\lambda f : \mathbf{nat} \Rightarrow \mathbf{nat}.\, \lambda x : \mathbf{nat}.\, f(x) + 2$.

$$(\quad \mathbf{nat} \quad \Rightarrow \quad \mathbf{nat} \quad) \quad \Rightarrow \quad \mathbf{nat} \quad \Rightarrow \quad \mathbf{nat}$$

$O$ q

$P$ q

$O$ q

# Example

Strategy for $\lambda f : \mathbf{nat} \Rightarrow \mathbf{nat}.\, \lambda x : \mathbf{nat}.\, f(x) + 2$.

$$(\quad \mathbf{nat} \quad \Rightarrow \quad \mathbf{nat} \quad) \quad \Rightarrow \quad \mathbf{nat} \quad \Rightarrow \quad \mathbf{nat}$$

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $O$ | | | | | | | q |
| $P$ | | | q | | | | |
| $O$ | | q | | | | | |
| $P$ | | | | | q | | |

# Example

Strategy for $\lambda f : \mathbf{nat} \Rightarrow \mathbf{nat}.\, \lambda x : \mathbf{nat}.\, f(x) + 2$.

$$(\ \mathbf{nat} \Rightarrow \mathbf{nat}\ ) \Rightarrow \mathbf{nat} \Rightarrow \mathbf{nat}$$

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| *O* | | | | | | | | q |
| *P* | | | q | | | | | |
| *O* | | q | | | | | | |
| *P* | | | | | | q | | |
| *O* | | | | | | n | | |

# Example

Strategy for $\lambda f : \mathbf{nat} \Rightarrow \mathbf{nat}.\, \lambda x : \mathbf{nat}.\, f(x) + 2$.

| | $($ **nat** | $\Rightarrow$ | **nat** $)$ | $\Rightarrow$ | **nat** | $\Rightarrow$ | **nat** |
|---|---|---|---|---|---|---|---|
| $O$ | | | | | | | q |
| $P$ | | | q | | | | |
| $O$ | q | | | | | | |
| $P$ | | | | | q | | |
| $O$ | | | | | n | | |
| $P$ | n | | | | | | |

# Example

Strategy for $\lambda f : \mathbf{nat} \Rightarrow \mathbf{nat}.\, \lambda x : \mathbf{nat}.\, f(x) + 2$.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | ( | **nat** | ⇒ | **nat** | ) | ⇒ | **nat** | ⇒ | **nat** |
| *O* | | | | | | | | q |
| *P* | | | | q | | | | |
| *O* | | q | | | | | | |
| *P* | | | | | | | q | |
| *O* | | | | | | | n | |
| *P* | | n | | | | | | |
| *O* | | | | m | | | | |

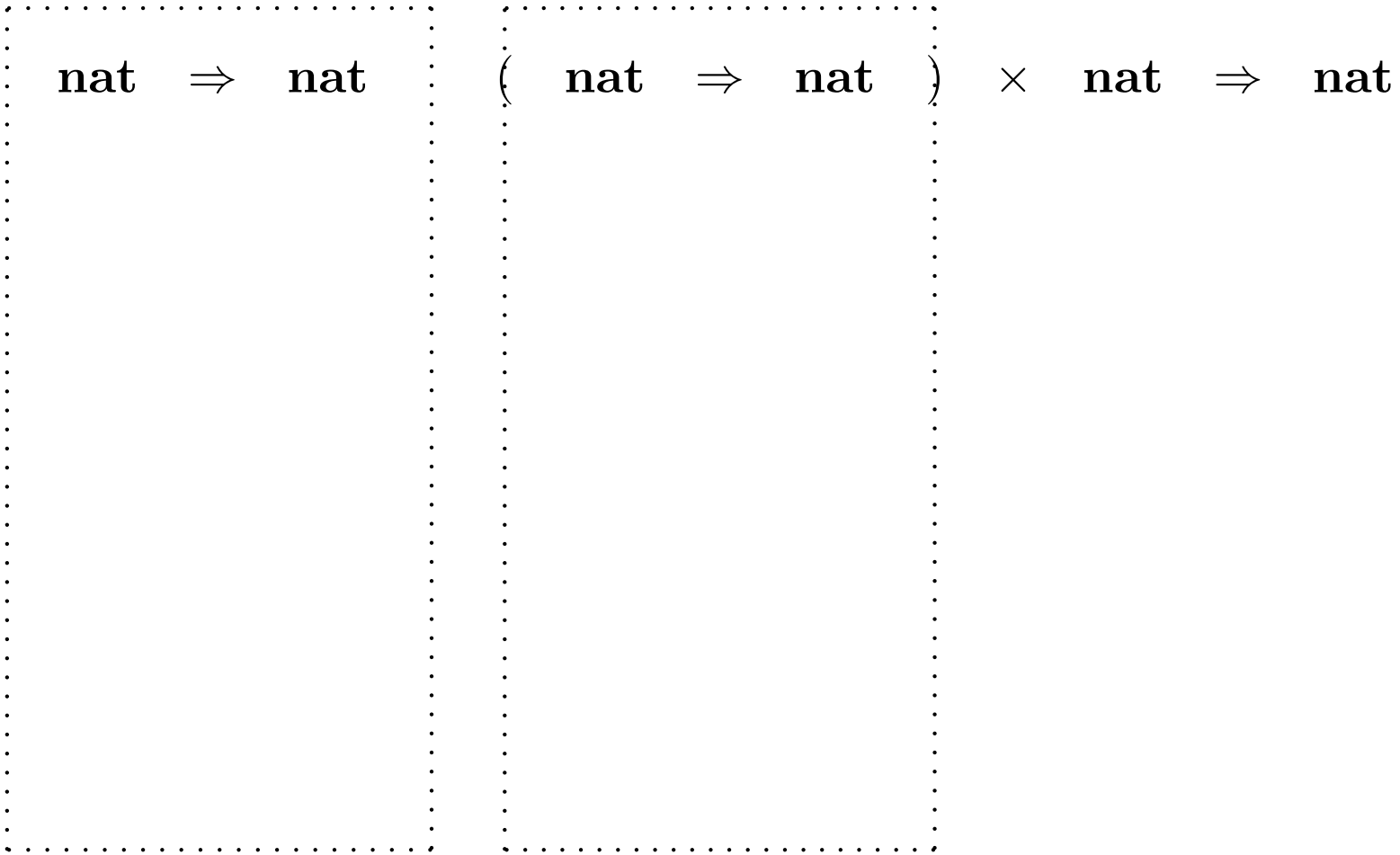
# Example

Strategy for $\lambda f : \mathbf{nat} \Rightarrow \mathbf{nat}.\, \lambda x : \mathbf{nat}.\, f(x) + 2$.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | ( | **nat** | ⇒ | **nat** | ) | ⇒ | **nat** | ⇒ | **nat** |
| $O$ | | | | | | | | | q |
| $P$ | | | | q | | | | | |
| $O$ | | q | | | | | | | |
| $P$ | | | | | | q | | | |
| $O$ | | | | | | n | | | |
| $P$ | | n | | | | | | | |
| $O$ | | | | m | | | | | |
| $P$ | | | | | | | | | m + 2 |

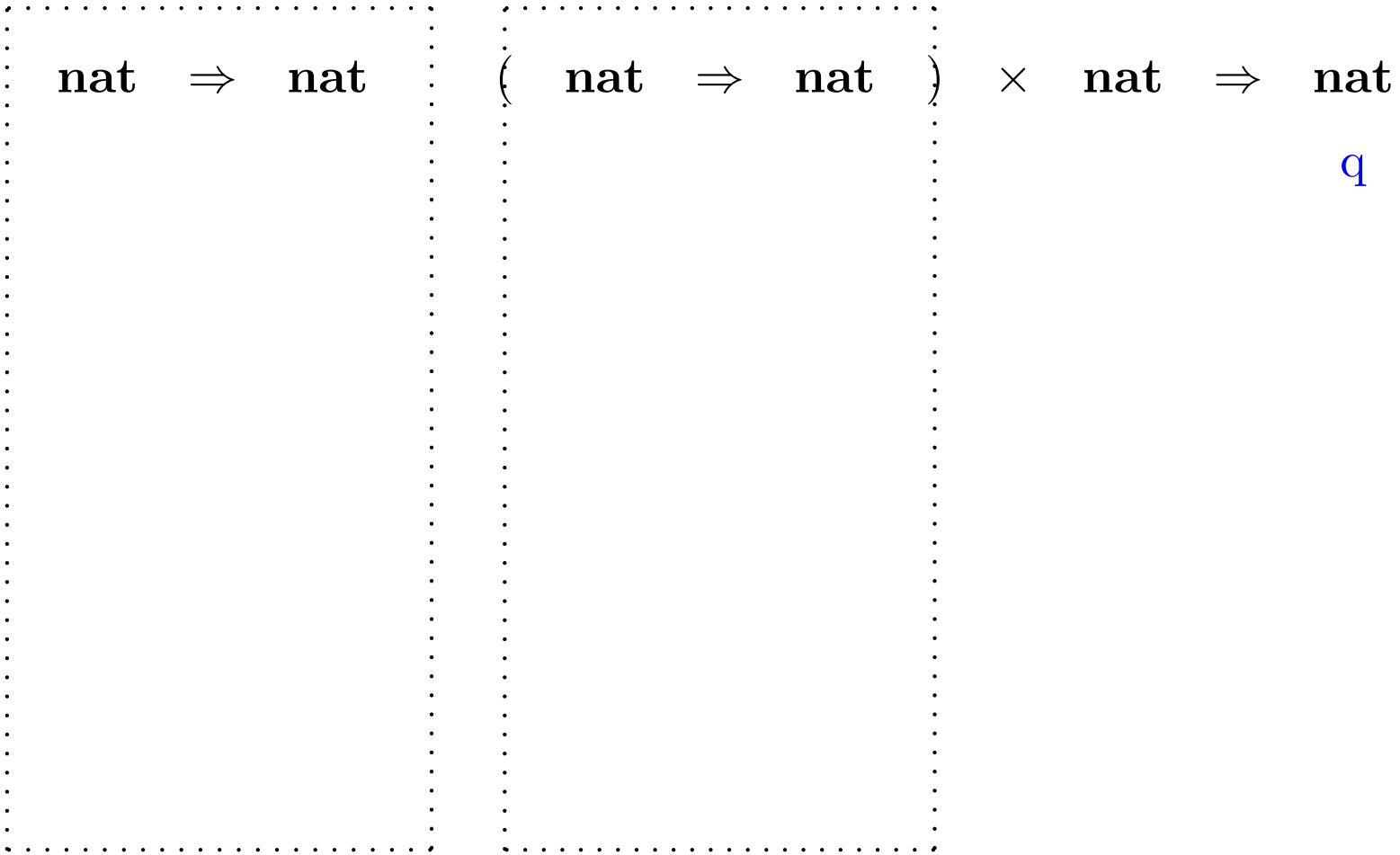
# Composition

Apply $\lambda f : \mathbf{nat} \Rightarrow \mathbf{nat}.\, \lambda x : \mathbf{nat}.\, f(x) + 2$ to $\lambda x : \mathbf{nat}.\, x^2$.



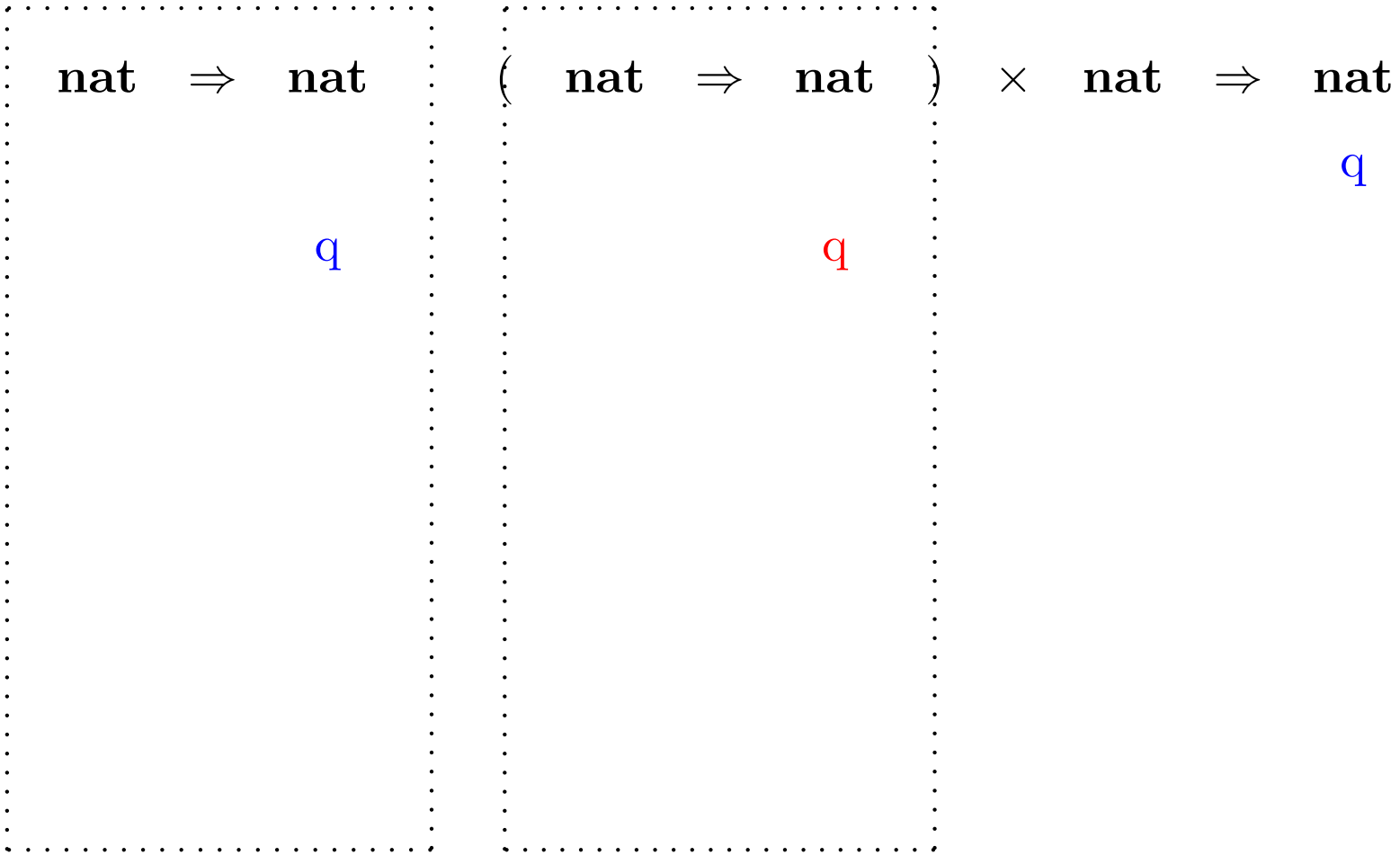
# Composition

Apply $\lambda f : \mathbf{nat} \Rightarrow \mathbf{nat}.\, \lambda x : \mathbf{nat}.\, f(x) + 2$ to $\lambda x : \mathbf{nat}.\, x^2$.



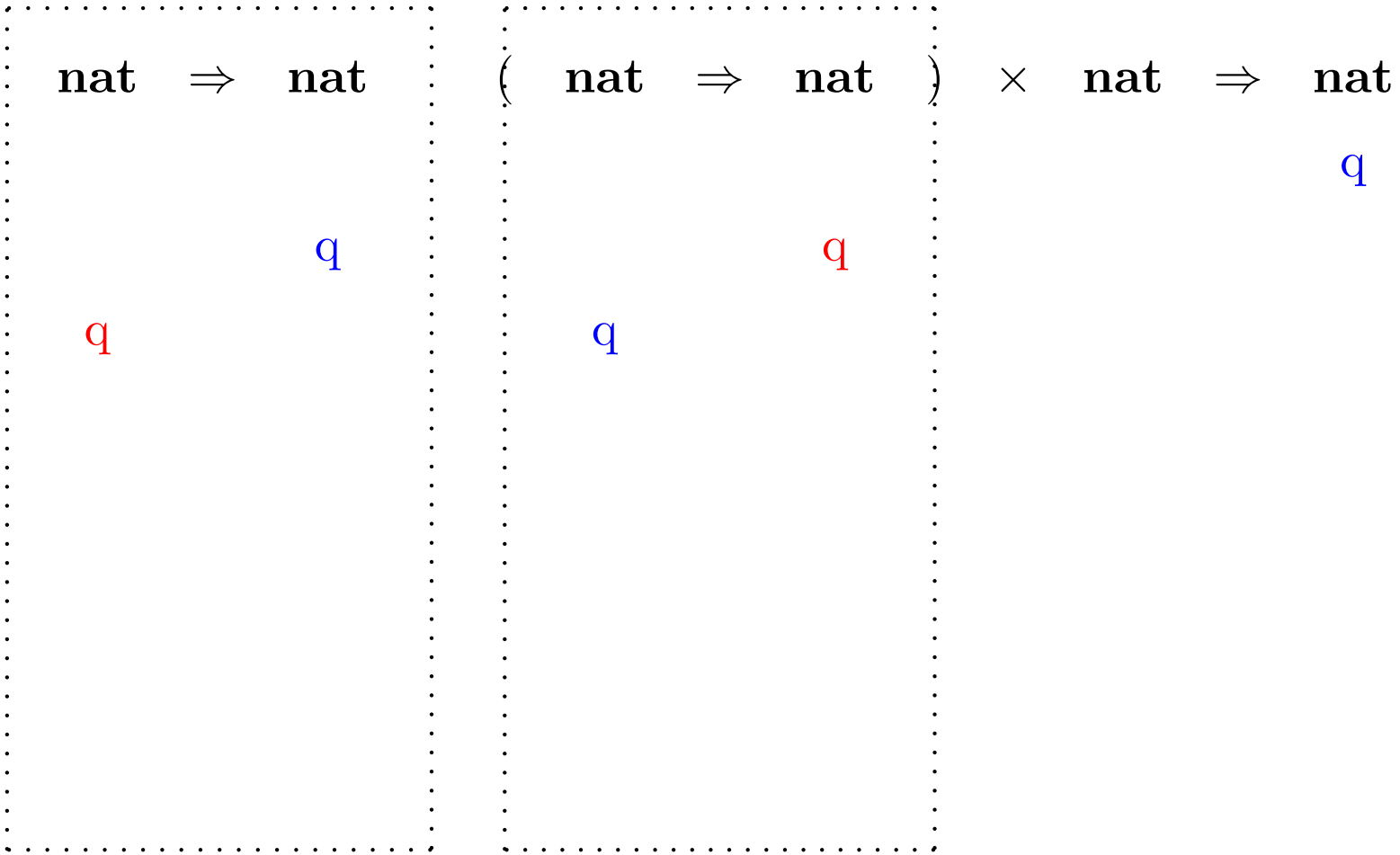
# Composition

Apply $\lambda f : \mathbf{nat} \Rightarrow \mathbf{nat}.\, \lambda x : \mathbf{nat}.\, f(x) + 2$ to $\lambda x : \mathbf{nat}.\, x^2$.



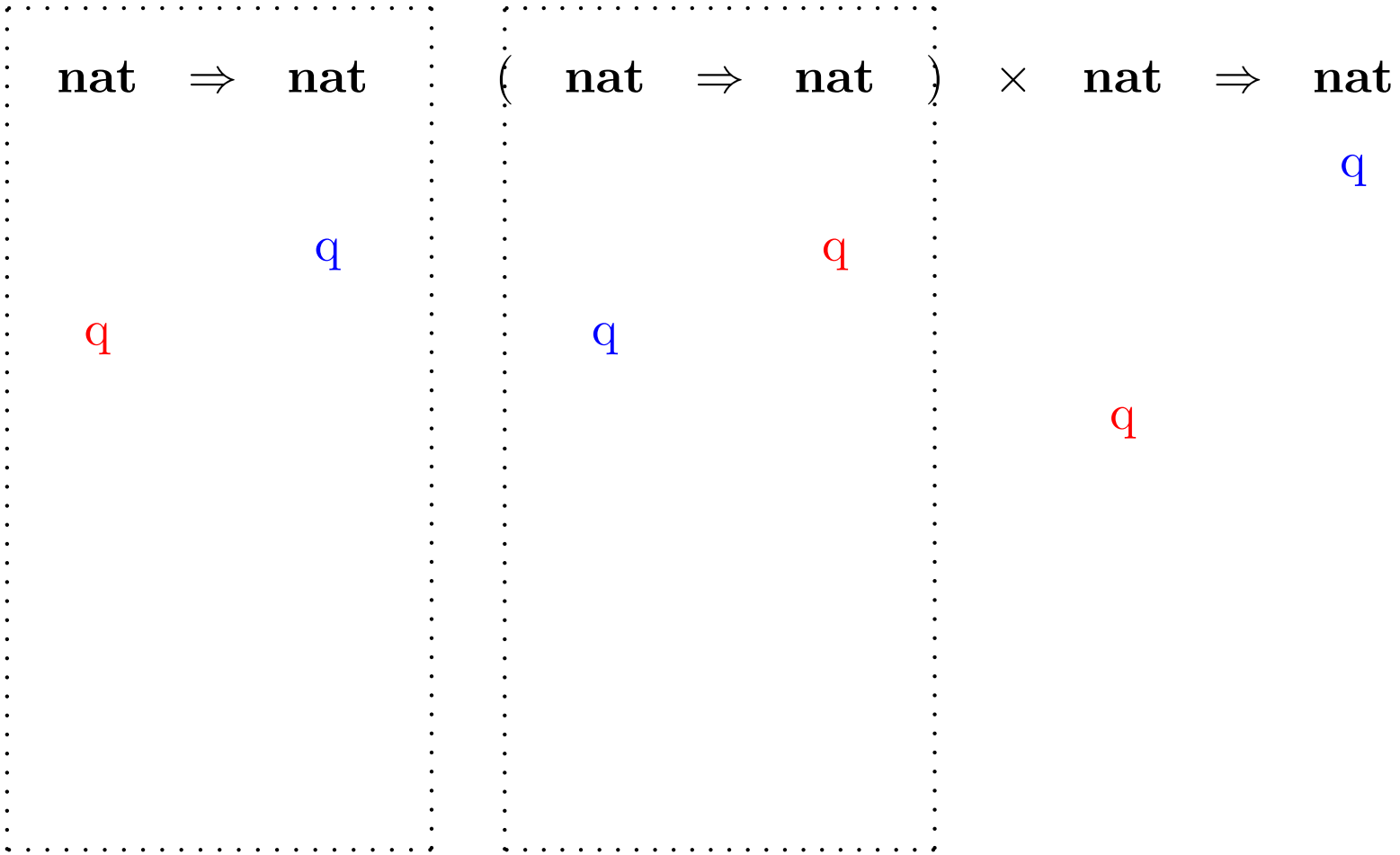
# Composition

Apply $\lambda f : \mathbf{nat} \Rightarrow \mathbf{nat}.\, \lambda x : \mathbf{nat}.\, f(x) + 2$ to $\lambda x : \mathbf{nat}.\, x^2$.



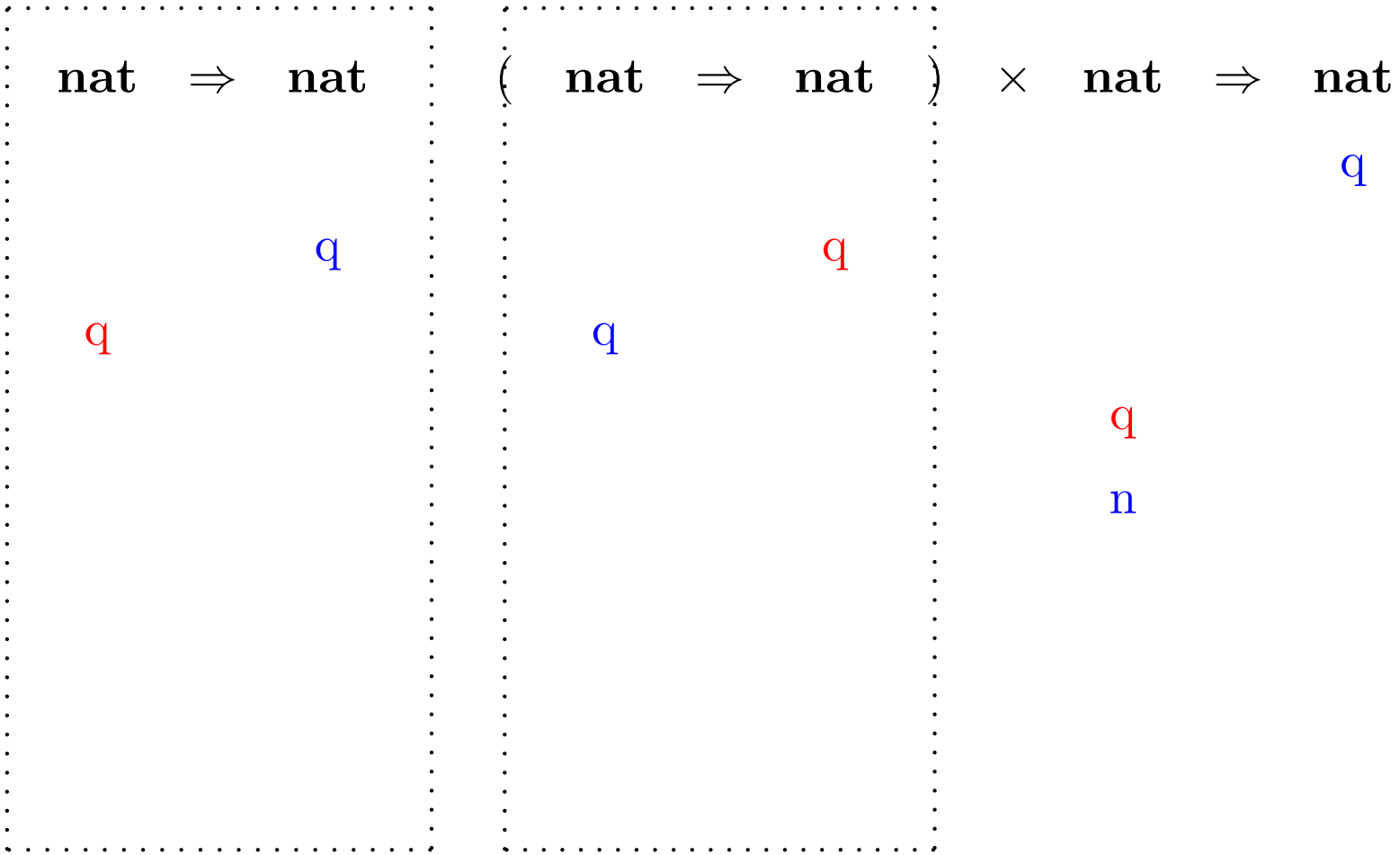
# Composition

Apply $\lambda f : \mathbf{nat} \Rightarrow \mathbf{nat}.\, \lambda x : \mathbf{nat}.\, f(x) + 2$ to $\lambda x : \mathbf{nat}.\, x^2$.



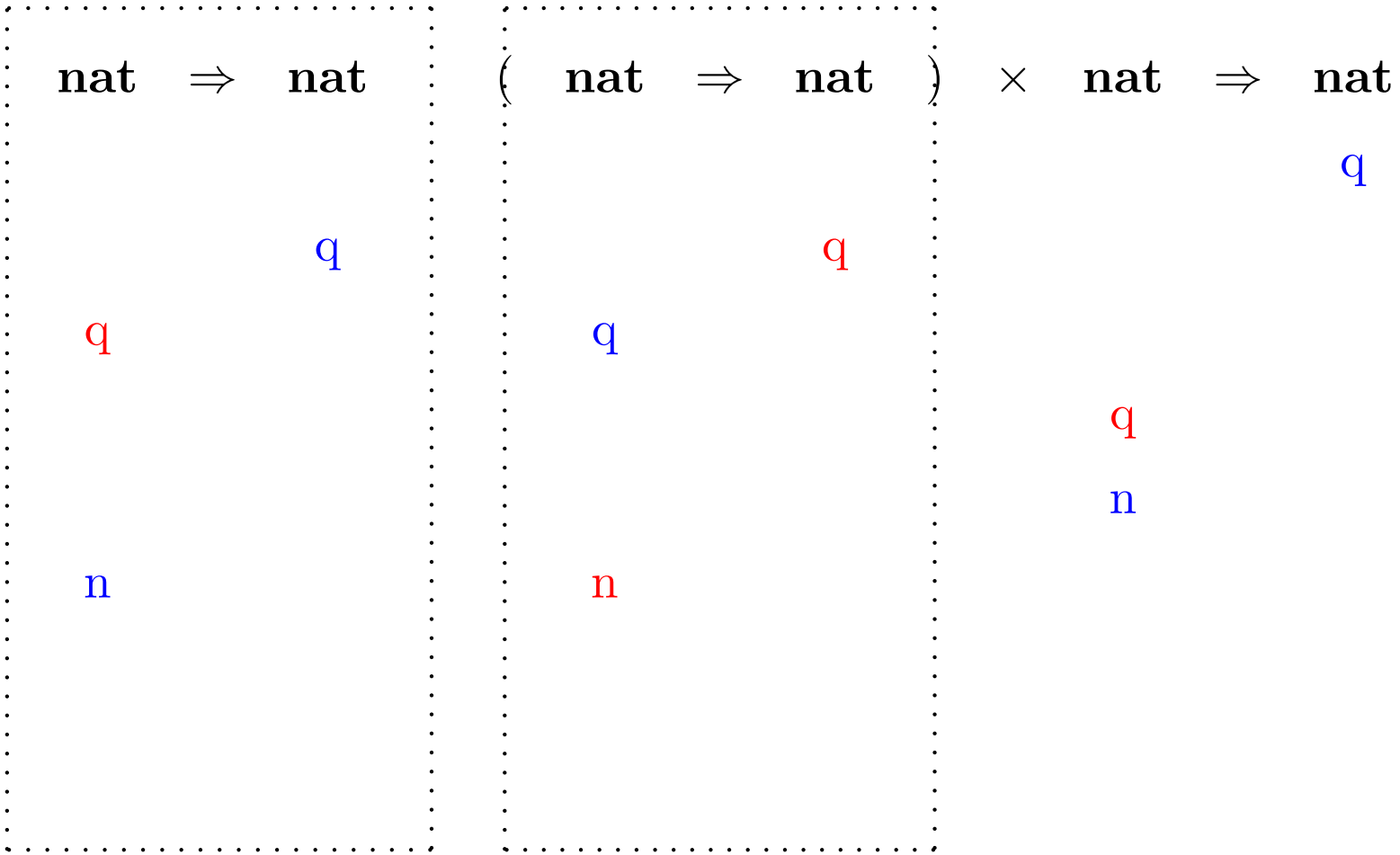
# Composition

Apply $\lambda f : \mathbf{nat} \Rightarrow \mathbf{nat}.\, \lambda x : \mathbf{nat}.\, f(x) + 2$ to $\lambda x : \mathbf{nat}.\, x^2$.



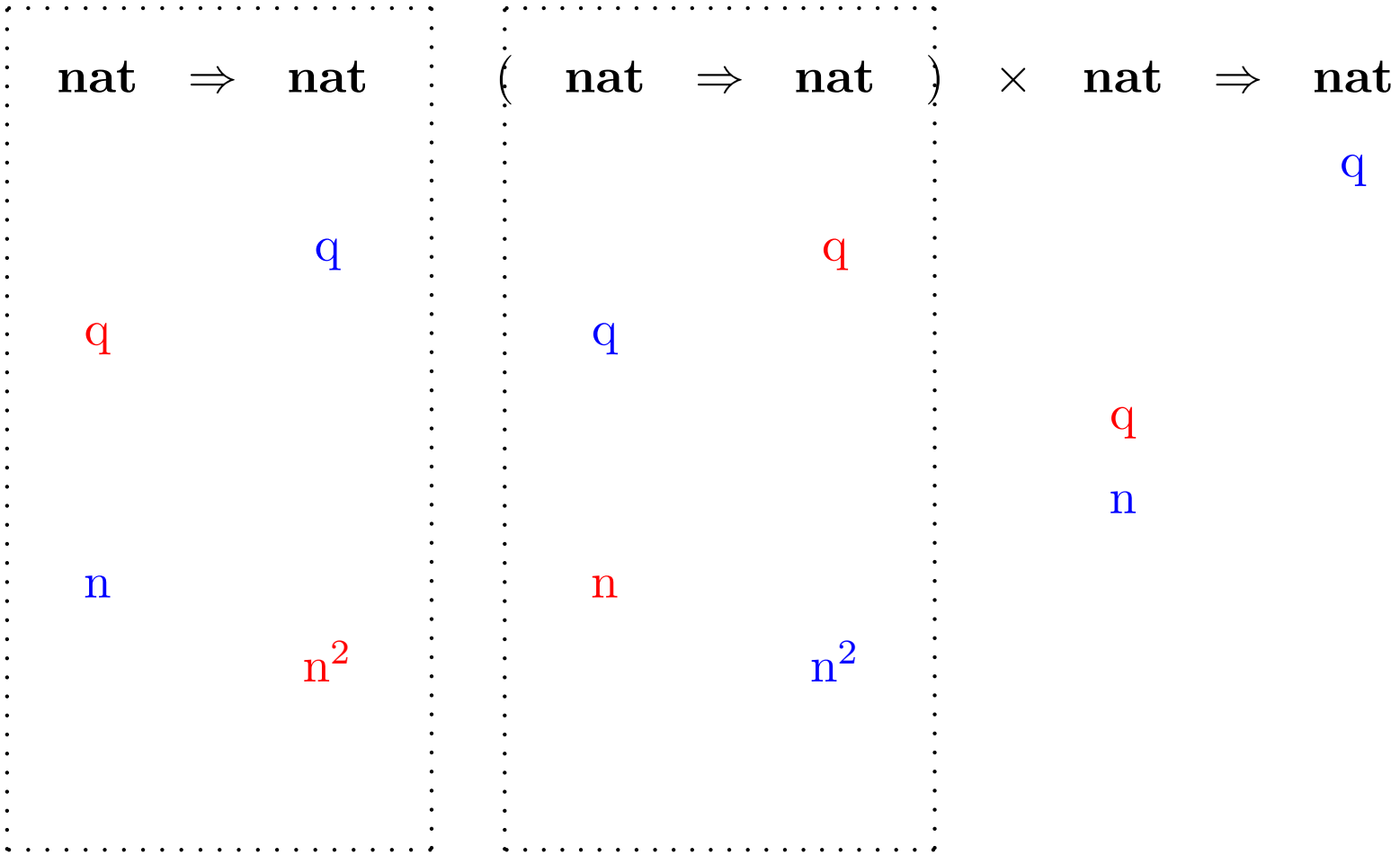
# Composition

Apply $\lambda f : \mathbf{nat} \Rightarrow \mathbf{nat}.\, \lambda x : \mathbf{nat}.\, f(x) + 2$ to $\lambda x : \mathbf{nat}.\, x^2$.



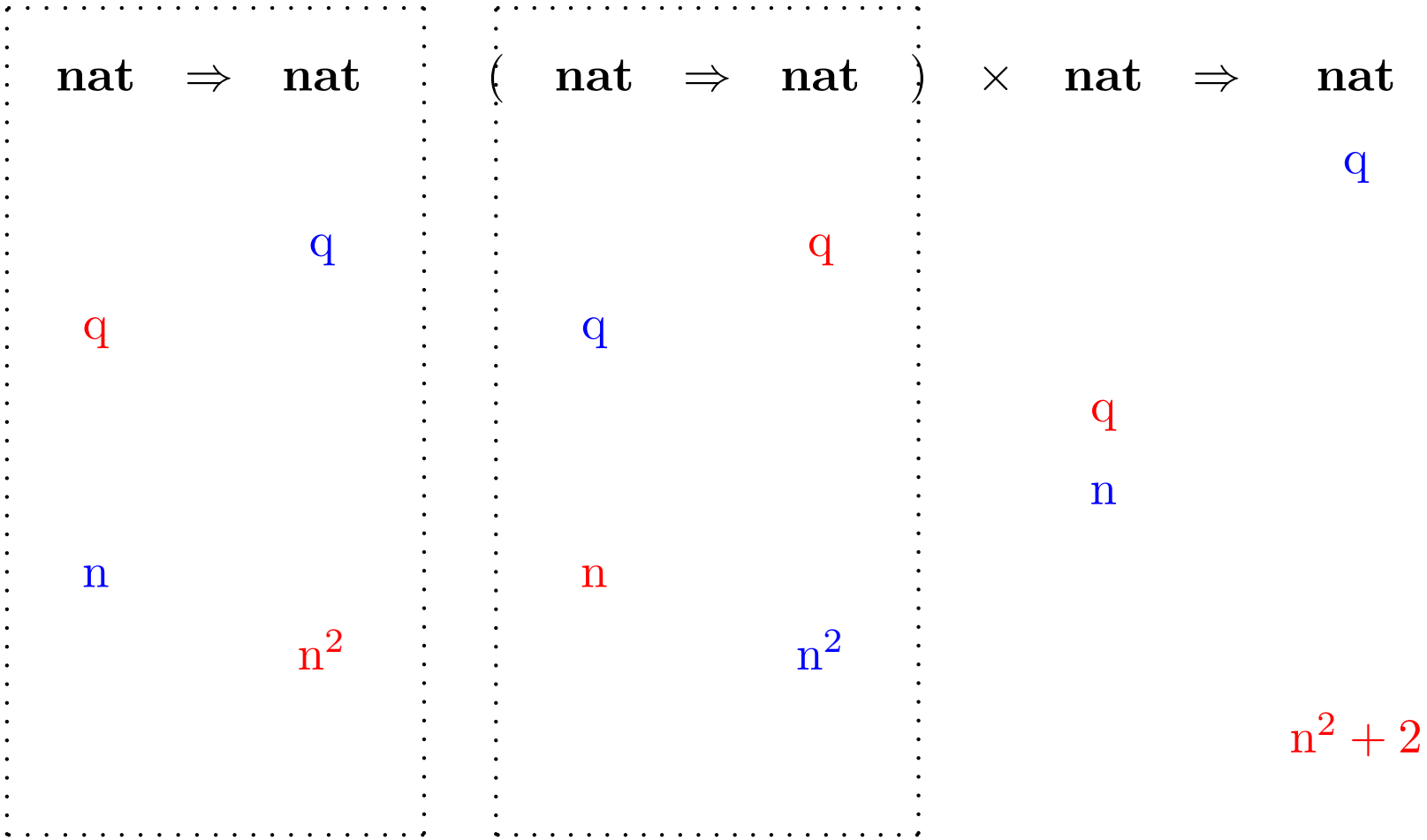
# Composition

Apply $\lambda f : \mathbf{nat} \Rightarrow \mathbf{nat}.\, \lambda x : \mathbf{nat}.\, f(x) + 2$ to $\lambda x : \mathbf{nat}.\, x^2$.

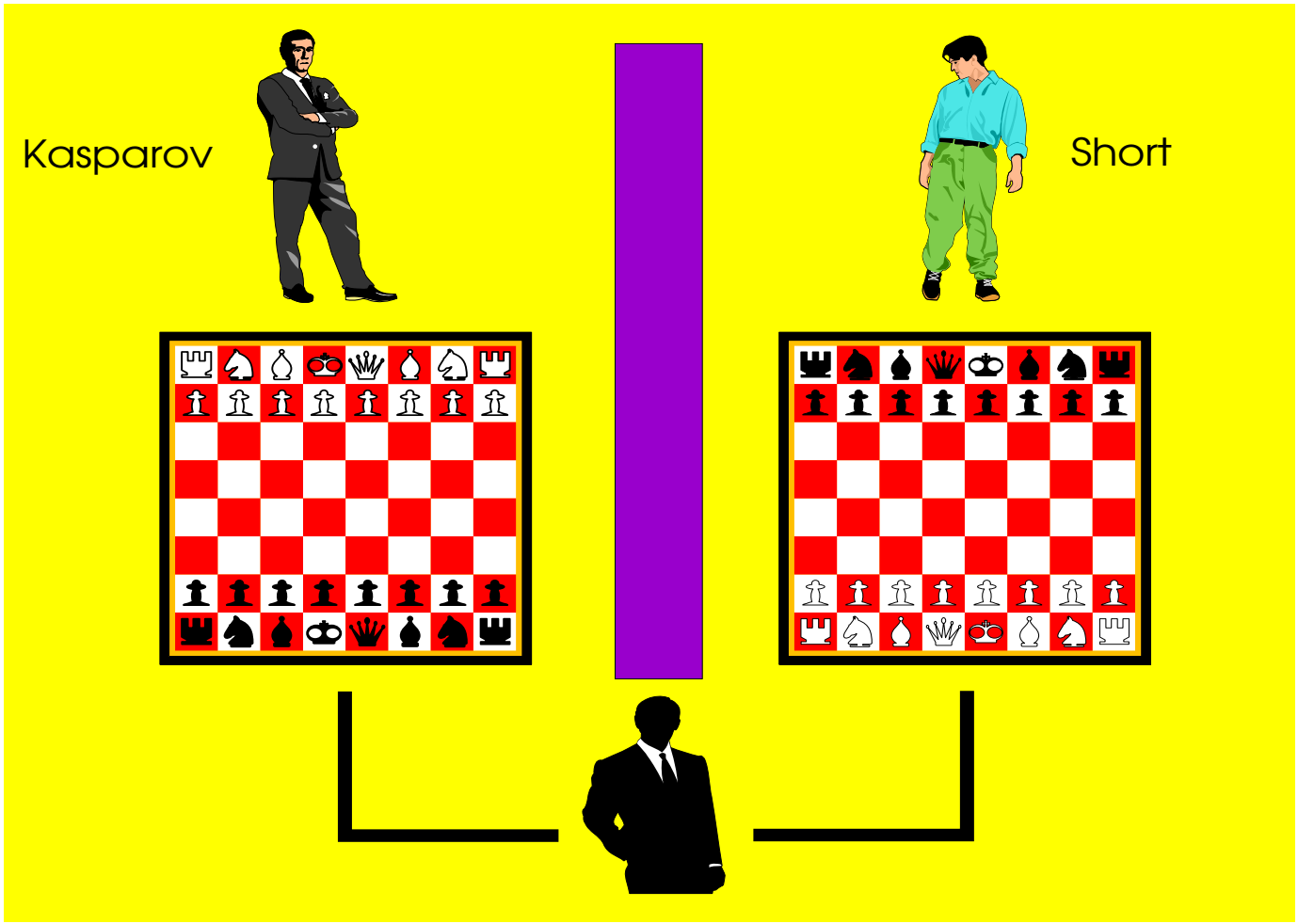


# Composition

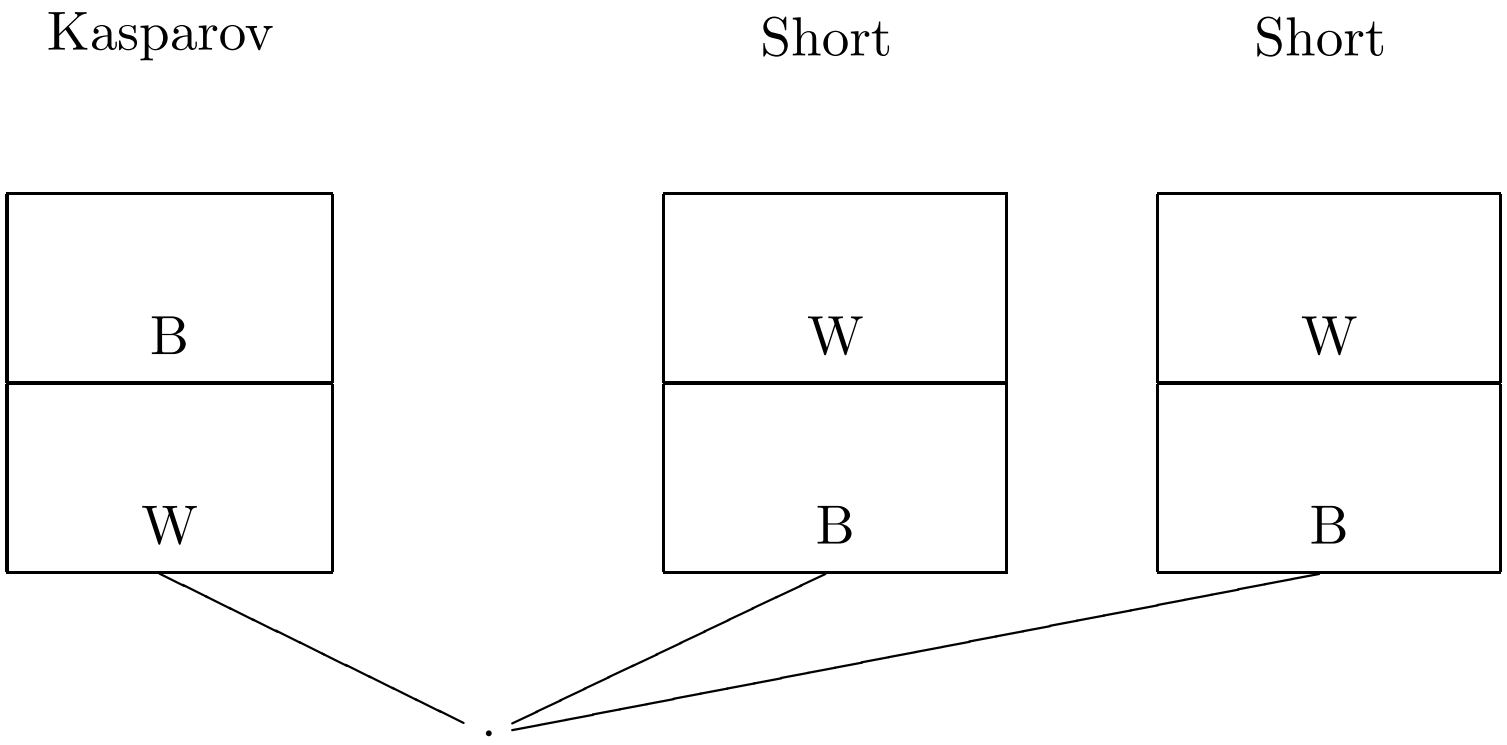
Apply $\lambda f : \mathbf{nat} \Rightarrow \mathbf{nat}.\, \lambda x : \mathbf{nat}.\, f(x) + 2$ to $\lambda x : \mathbf{nat}.\, x^2$.



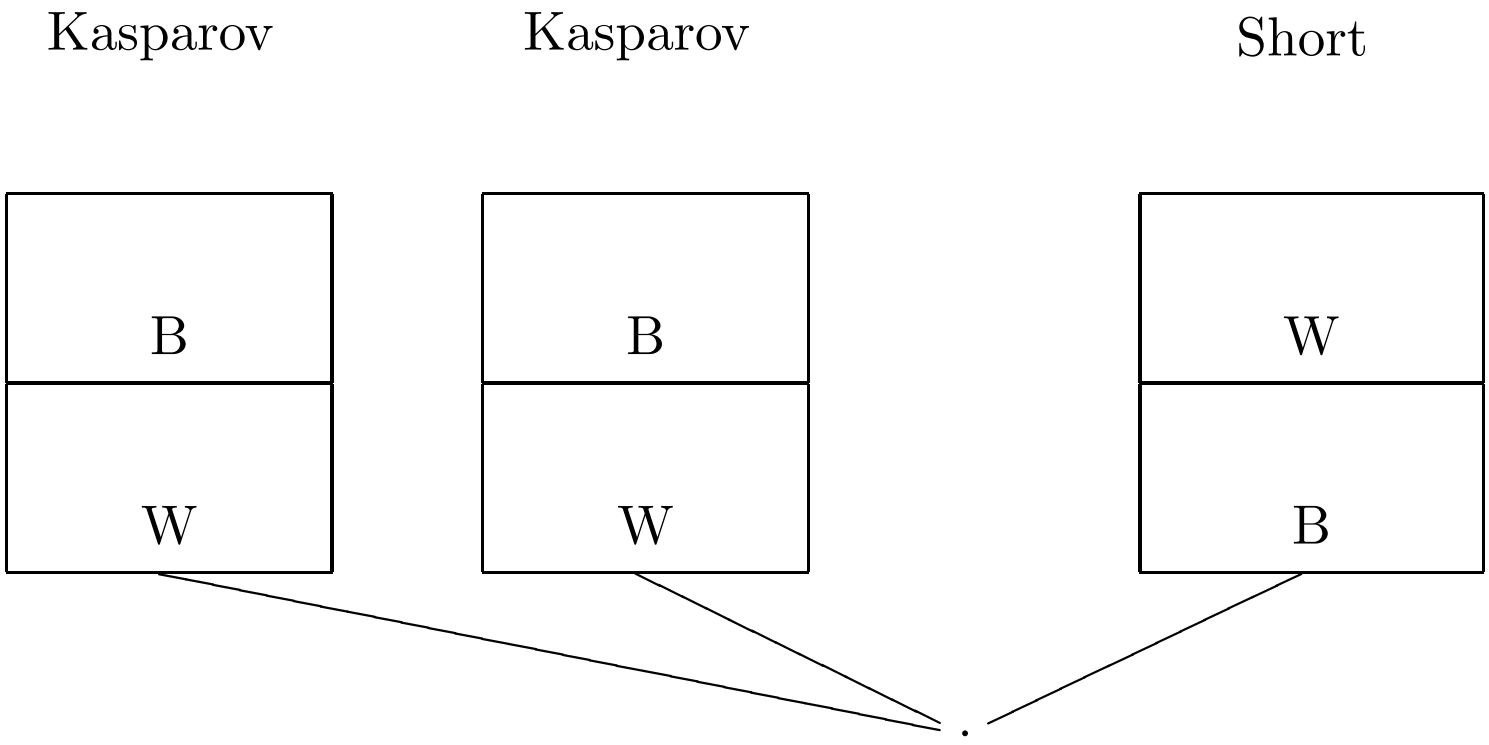
# The Copy-Cat Strategy



# Does Copy-Cat still work here?



# And here?



# The General Picture

Games and strategies organize themselves into mathematical structures (categories of various kinds) suitable for modelling programming languages.

By imposing various **structural constraints** on strategies, exact matches can be found with various **logical disciplines**, leading to **full completeness** results, which characterize the 'space of proofs' of various logics.

Similarly, exact matches can be found with a wide range of **computational features** as embodied in key programming language constructs, leading to **full abstraction** results.

# Full Completeness

Ordinary completeness speaks of **provability**; full (and faithful) completeness speaks of **proofs**. A proof of $\Gamma \vdash A$ will denote a strategy

$$\sigma : [\![\Gamma]\!] \longrightarrow [\![A]\!].$$

This is (part of) soundness.

Completeness asks for a converse; that for every such $\sigma$, there exists a proof $\Pi$ of $\Gamma \vdash A$. Full completeness asks that moreover $\Pi$ **denotes** the $\sigma$ we started with, *i.e.* that the mapping of proofs to strategies is surjective. Faithfulness is the additional requirement that different normal forms map onto distinct strategies.

These results give intrinsic semantic characterizations of the 'space of proofs' of a logic.

# The Game Semantics Landscape

Game semantics has proved to be a flexible and powerful paradigm for constructing highly structured fully abstract semantics for languages with a wide range of computational features:

- (higher-order) functions and procedures
- call by name and call by value
- locally scoped state
- general reference types
- control features (continuations, exceptions)
- non-determinism, probabilities
- concurrency (GM, FOSSACS 04)
- names and freshness (AGMOS, LiCS 04)

# Algorithmic Game Semantics

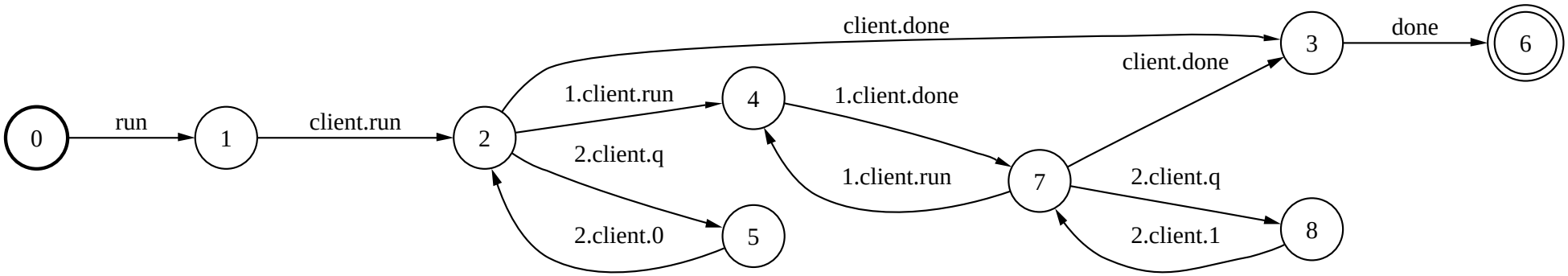
We can take advantage of the **concrete nature** of game semantics. A play is a sequence of moves, so a strategy can be represented by the set of its plays, i.e. by a **language** over the alphabet of moves, and hence by an automaton.

There are significant finite-state fragments of the semantics for various interesting languages, as first observed by Ghica and McCusker (ICALP 00).

This means we can **compositionally construct** automata as (representations of) the meanings of open (incomplete) programs, giving a powerful basis for compositional software model-checking.

# Warm-up example

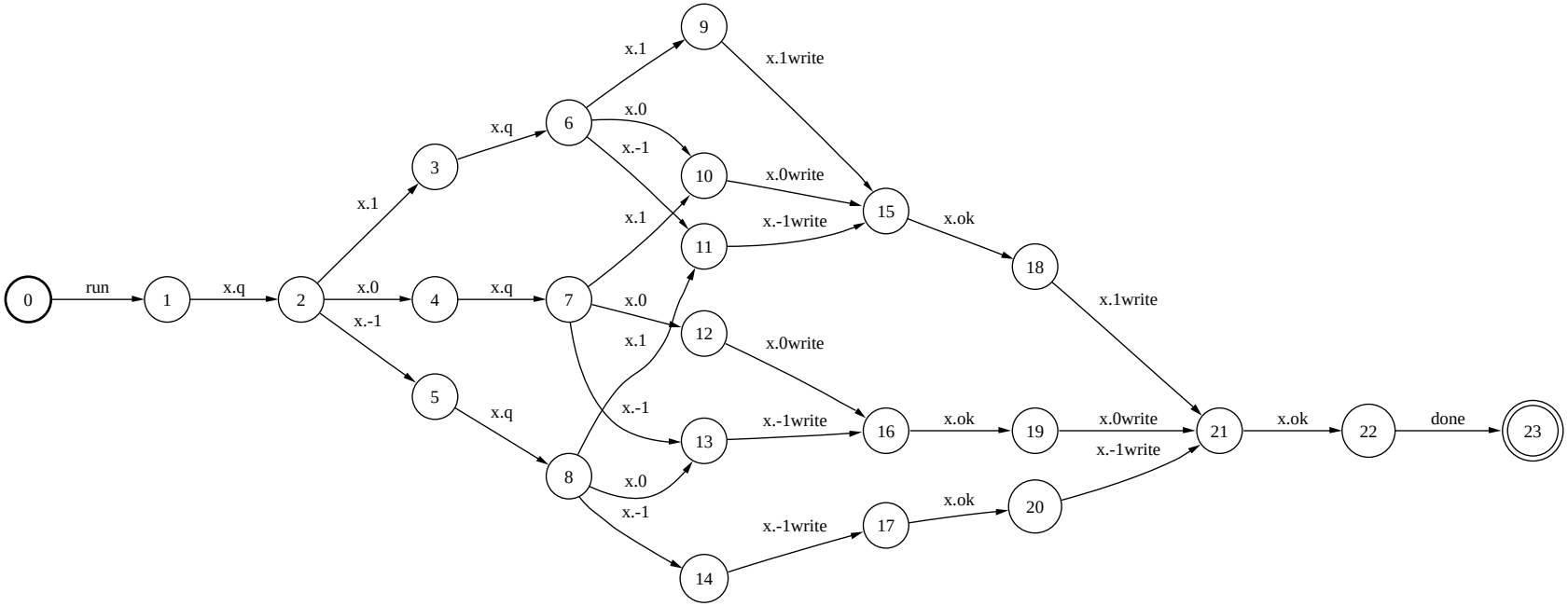
```
client : com -> exp -> com |-
        new var v:= 0 in
        let set be v:=1 in
        let get be !v in
        client (set, get): com.
```



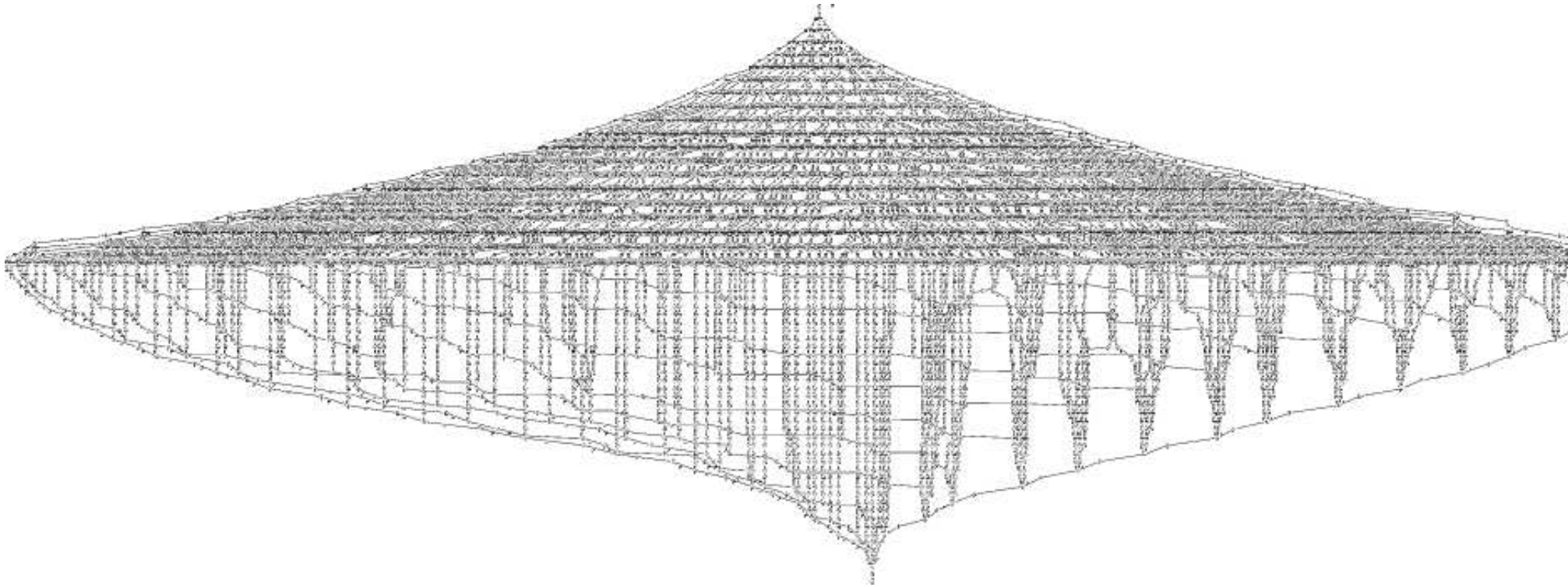
# Sorting: bubblesort

```
x:var |-
  array a[\(n\)] in
  new var i:=0 in
  while !i < \(n\) do a[!i]:=!x; i:=!i+1 od;
  new var flag:=1 in
  while !flag do
    new var i:=0 in
    flag:=0;
    while !i < \(n\) - 1 do
      if !a[!i] > !a[!i+1] then
        flag:=1;
        new var temp:=!a[!i] in
          a[!i]:=!a[!i+1]; a[!i+1]:=!temp
        else skip fi;
      i:=!i+1 od od;
  new var i:=0 in
  while !i < \(n\) do x:=!a[!i]; i:=!i+1 od : com.
```

# 2-element array of integers %2



# 20-element array of integers %2



# On modeling sorting programs

"[...] it seems impossible to use model-checking to verify that a sorting algorithm is correct since sorting correctness is a data-oriented property involving several quantifications and data structures." [Bandera user manual]

Why does it work?

- program state-space: $5.5 \times 10^{12}$ states
- model: $6,393$ states
- max space: $1,153,240$ states

**Hiding local state!**

## Game Semantics in the Games landscape

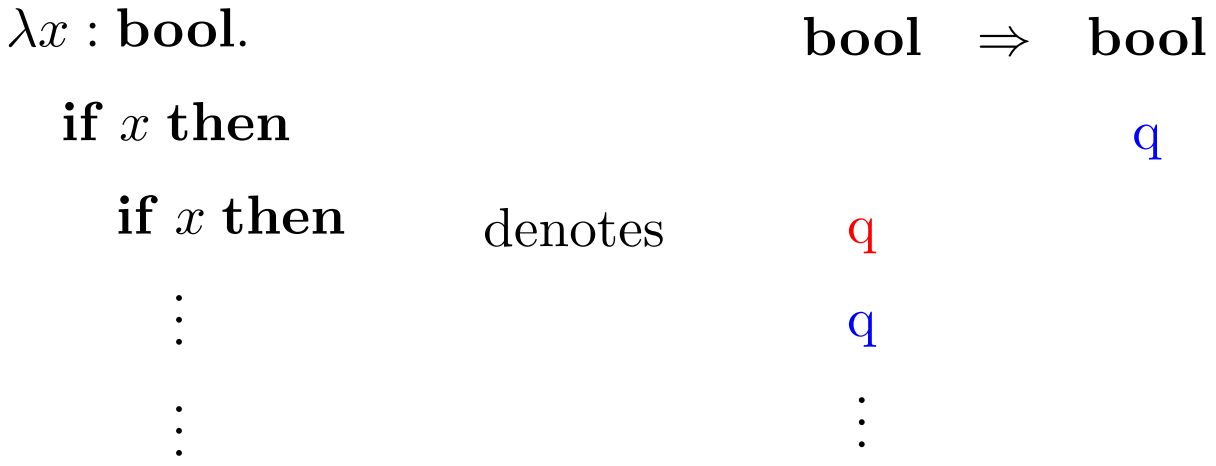
Some comparisons:

- Hintikaa GTS and IF logic. GS is more compositional; a proper analysis of implication!
- Lorenzen school of dialogue games. An ancestor; more compositional, 'syntax-free', much wider scope.
- Blass games. Another ancestor. Overcomes problems with compositionality.

Our main focus (to date) has been on **structural** aspects, (categories of) games in extensive form, rather than fine-grained analysis of winning strategies, or solution concepts and equilibria. Our key equilibria are 'logical', e.g. the copy-cat strategy.

# Infinite Games in Game Semantics

Infinite games arise in modelling even the simplest function (or procedure) types, because of the possibility of repeated evaluation of arguments, and the fact that each **interaction** with a (free) argument is made **observable**:



For untyped calculi:

$$D \cong [D \Rightarrow D]$$

$$\cdots \quad \Rightarrow \quad D \quad ) \quad \Rightarrow \quad D \quad ) \quad \Rightarrow \quad D$$

q

q

q

$\cdots$

## Programs vs. Proofs

Proofs can be seen as programs (the Curry-Howard isomorphism), but not all programs are proofs. Proofs in logical systems, as opposed to programs in general, should **terminate/normalize**, they should define **total** functions. Programs in general can diverge.

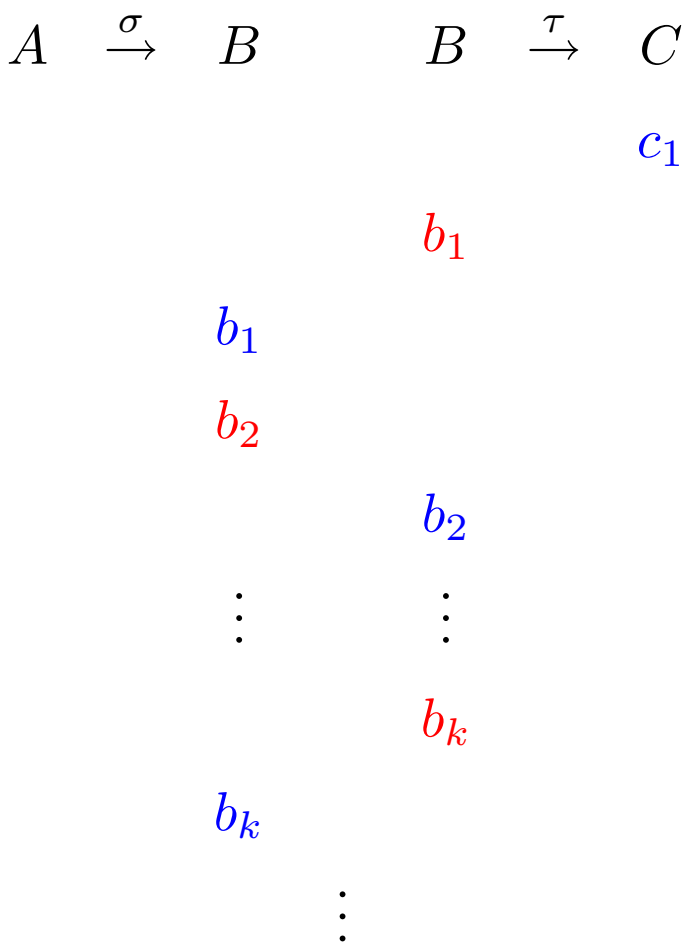
How can we make this distinction in terms of Game Semantics?

## First attempt: Totality

We would like to find a condition on strategies generalizing totality of functions. The obvious candidate is to require that at each stage of play, a strategy $\sigma$ on A has some response to every possible move by opponent. Call a strategy **total** if it satisfies this condition.

This is ok if games are **finite**; but as we have seen, in general they are not. When games are infinite, total strategies **are not closed under composition**.

# Infinite Chattering



## Winning strategies

We need to expand our concept of game by specifying which infinite plays of the game are wins for Player. Then we say that a strategy is **winning** if at each finite stage when it is Player's turn to move it has a well defined response, and moreover every infinite play following $\sigma$ is a win for Player.

We introduce an expanded of refined notion of game as a pair $(A, W_A)$, where $A$ is a game as before, and $W_A \subseteq P_A^\infty$ is the designated set of winning infinite plays for Player. A winning strategy for $(A, W_A)$ is a strategy for $A$ which is winning with respect to $W_A$.

We now extend the definitions of connectives such as $\Rightarrow$ to act on the winning set specifications:

$$(A, W_A) \Rightarrow (B, W_B) \quad = \quad (A \Rightarrow B, W_{A \Rightarrow B})$$

where

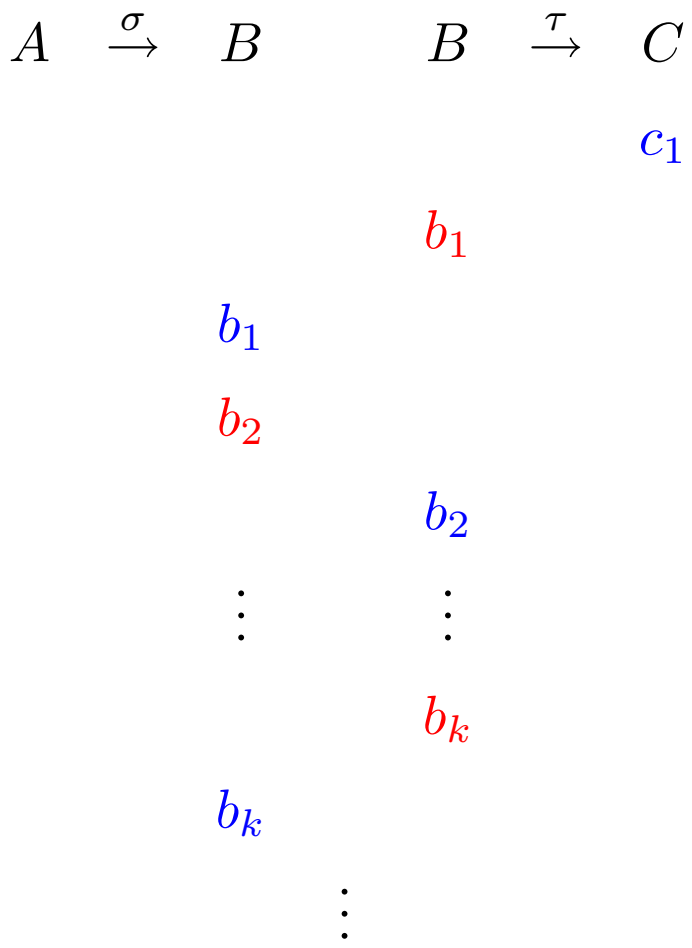
$$W_{A \Rightarrow B} \quad = \quad \{s \in P^\infty_{A \Rightarrow B} \mid s \upharpoonright A \in P_A \cup W_A \Rightarrow s \upharpoonright B \in W_B\}$$

Again, this definition can be seen as expressing a suitable notion of **logical equilibrium**. The important point to be proved is that **winning strategies are closed under all the operations on strategies corresponding to constructions of proofs**.

# Example: the Copy-Cat Strategy

$$
\begin{array}{ccc}
A & \Rightarrow & A \\
 & & a_1 \\
a_1 & & \\
a_2 & & \\
 & & a_2 \\
 & \vdots &
\end{array}
$$

# Example: Composition



## Proof that winning strategies compose

Suppose then that

$$\sigma : (A, W_A) \to (B, W_B), \qquad \tau : (B, W_B) \to (C, W_C).$$

We want to prove that $\sigma; \tau$ is total, i.e. that there can be no infinite chattering in B.
Suppose for a contradiction that there is an infinite play

$$t = sb_0b_1 \cdots \in \sigma \| \tau$$

with all moves after the finite prefix $s$ in $B$. Then $t \upharpoonright A, B$ is an infinite play in $A \Rightarrow B$ following $\sigma$, while $t \upharpoonright B, C$ is an infinite play in $B \multimap C$ following $\tau$. Since $\sigma$ is winning and $t \upharpoonright A$ is finite, we must have $t \upharpoonright B \in W_B$. But then since $\tau$ is winning we must have $t \upharpoonright C \in W_C$, which is impossible since $t \upharpoonright C$ is finite.

These ideas lead to **game-semantical proofs of normalization** for various logical calculi.

Some references (available from my web page):

- *Semantics of Interaction*.
- *Full Completeness for Multiplicative Linear Logic* (with Radha Jagadeesan).
- *Concurrent Games and Full Completeness* (with Paul-André Melliès).
- *A Game Semantics for Generic Polymorphism* (with Radha Jagadeesan).