Tutorial on Game Semantics

Samson Abramsky
Oxford University Computing Laboratory
`http://web.comlab.ox.ac.uk/oucl/work/samson.abramsky/`

December 4, 2008

Introduction

What is Logic about, anyway?

What is Logic about, anyway?

Two views: the ‘model-theoretic’ and the ‘proof-theoretic’ perspectives.

Two views: the ‘model-theoretic’ and the ‘proof-theoretic’ perspectives.

1. **The Descriptive View.** Logic is used to **talk about** structure. This is the view taken in Model Theory, and in most of the uses of Logic (Temporal logics, MSO etc.) in Verification. It is by far the more prevalent and widely-understood view.

Two views: the 'model-theoretic' and the 'proof-theoretic' perspectives.

1. **The Descriptive View.** Logic is used to **talk about** structure. This is the view taken in Model Theory, and in most of the uses of Logic (Temporal logics, MSO etc.) in Verification. It is by far the more prevalent and widely-understood view.

2. **The Intrinsic View.** Logic is taken to **embody** structure. This is, implicitly or explicitly, the view taken in the Curry-Howard isomorphism, and more generally in Structural Proof Theory, and in (much of) Categorical Logic. In the Curry-Howard isomorphism, one is not using logic to **talk about** functional programming; rather, logic (in this aspect) **is** functional programming.

What is Logic about, anyway?

Two views: the 'model-theoretic' and the 'proof-theoretic' perspectives.

1. **The Descriptive View.** Logic is used to **talk about** structure. This is the view taken in Model Theory, and in most of the uses of Logic (Temporal logics, MSO etc.) in Verification. It is by far the more prevalent and widely-understood view.

2. **The Intrinsic View.** Logic is taken to **embody** structure. This is, implicitly or explicitly, the view taken in the Curry-Howard isomorphism, and more generally in Structural Proof Theory, and in (much of) Categorical Logic. In the Curry-Howard isomorphism, one is not using logic to **talk about** functional programming; rather, logic (in this aspect) **is** functional programming.

Amazingly, the relationship between these two points of view has hardly been identified as an issue, let alone discussed.

What is Logic about, anyway?

Two views: the ‘model-theoretic’ and the ‘proof-theoretic’ perspectives.

1. **The Descriptive View.** Logic is used to **talk about** structure. This is the view taken in Model Theory, and in most of the uses of Logic (Temporal logics, MSO etc.) in Verification. It is by far the more prevalent and widely-understood view.

2. **The Intrinsic View.** Logic is taken to **embody** structure. This is, implicitly or explicitly, the view taken in the Curry-Howard isomorphism, and more generally in Structural Proof Theory, and in (much of) Categorical Logic. In the Curry-Howard isomorphism, one is not using logic to **talk about** functional programming; rather, logic (in this aspect) **is** functional programming.

Amazingly, the relationship between these two points of view has hardly been identified as an issue, let alone discussed.
I hope we will do this in LINT!

And Game Semantics?

And Game Semantics?

Games have many faces in logic and computation.

And Game Semantics?

Games have many faces in logic and computation.

‘Game Semantics’ can cover a wide range of material.

And Game Semantics?

Games have many faces in logic and computation.

'Game Semantics' can cover a wide range of material.

Since around 1992, a community has developed in the Logic and Semantics side of CS working in Game Semantics with the following key features, making it rather distinct from previous work under this heading.

And Game Semantics?

Games have many faces in logic and computation.

‘Game Semantics’ can cover a wide range of material.

Since around 1992, a community has developed in the Logic and Semantics side of CS working in Game Semantics with the following key features, making it rather distinct from previous work under this heading.

- Compositionality

Games have many faces in logic and computation.

'Game Semantics' can cover a wide range of material.

Since around 1992, a community has developed in the Logic and Semantics side of CS working in Game Semantics with the following key features, making it rather distinct from previous work under this heading.

- Compositionality
- Syntax-independence

And Game Semantics?

Games have many faces in logic and computation.

'Game Semantics' can cover a wide range of material.

Since around 1992, a community has developed in the Logic and Semantics side of CS working in Game Semantics with the following key features, making it rather distinct from previous work under this heading.

- Compositionality
- Syntax-independence
- Powerful results on full abstraction and full completeness for a wide range of programming languages and logical type theories

Games have many faces in logic and computation.

'Game Semantics' can cover a wide range of material.

Since around 1992, a community has developed in the Logic and Semantics side of CS working in Game Semantics with the following key features, making it rather distinct from previous work under this heading.

- Compositionality
- Syntax-independence
- Powerful results on full abstraction and full completeness for a wide range of programming languages and logical type theories
- More recently an algorithmic turn, and many striking applications to verification.

Game Semantics for Programs

Basic Ideas

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

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

- Programs are **strategies** for these games.

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

- Programs 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.

Basic Ideas

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

- Programs 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:

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

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

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

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.

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

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

  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**.

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

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

(This names the procedure $P(f, x)$ such that $P(f, x)$ returns $f(x) + 2$.)

Example

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

(This names the procedure $P(f, x)$ such that $P(f, x)$ returns $f(x) + 2$.)

$$(\quad \mathbb{N} \quad \Rightarrow \quad \mathbb{N} \quad) \quad \Rightarrow \quad \mathbb{N} \quad \Rightarrow \quad \mathbb{N}$$

Example

Strategy for $\lambda f : \mathbb{N} \Rightarrow \mathbb{N}. \lambda x : \mathbb{N}. f(x) + 2$.

$$(\quad \mathbb{N} \quad \Rightarrow \quad \mathbb{N} \quad) \quad \Rightarrow \quad \mathbb{N} \quad \Rightarrow \quad \mathbb{N}$$

O

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

$$(\quad \mathbb{N} \quad \Rightarrow \quad \mathbb{N} \quad) \quad \Rightarrow \quad \mathbb{N} \quad \Rightarrow \quad \mathbb{N}$$

O						q

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

	(	$\mathbb{N}$	$\Rightarrow$	$\mathbb{N}$	)	$\Rightarrow$	$\mathbb{N}$	$\Rightarrow$	$\mathbb{N}$
O									q
P				q					

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

	$($	$\mathbb{N}$	$\Rightarrow$	$\mathbb{N}$	$)$	$\Rightarrow$	$\mathbb{N}$	$\Rightarrow$	$\mathbb{N}$
O									q
P				q					
O		q							

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

	(	$\mathbb{N}$	$\Rightarrow$	$\mathbb{N}$	)	$\Rightarrow$	$\mathbb{N}$	$\Rightarrow$	$\mathbb{N}$
O									q
P				q					
O		q							
P							q		

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

	(	$\mathbb{N}$	$\Rightarrow$	$\mathbb{N}$	)	$\Rightarrow$	$\mathbb{N}$	$\Rightarrow$	$\mathbb{N}$
O									q
P				q					
O		q							
P							q		
O							n		

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

	(	$\mathbb{N}$	$\Rightarrow$	$\mathbb{N}$	)	$\Rightarrow$	$\mathbb{N}$	$\Rightarrow$	$\mathbb{N}$
O									q
P				q					
O		q							
P							q		
O							n		
P		n							

Example

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

	($\mathbb{N}$	$\Rightarrow$	$\mathbb{N}$)	$\Rightarrow$	$\mathbb{N}$	$\Rightarrow$	$\mathbb{N}$
O							q
P			q				
O	q						
P					q		
O					n		
P	n						
O			m				

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

	(	$\mathbb{N}$	$\Rightarrow$	$\mathbb{N}$	)	$\Rightarrow$	$\mathbb{N}$	$\Rightarrow$	$\mathbb{N}$
O									q
P				q					
O		q							
P							q		
O							n		
P		n							
O				m					
P									$\mathrm{m} + 2$

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

$$\mathbb{N} \Rightarrow \mathbb{N} \qquad (\mathbb{N} \Rightarrow \mathbb{N}) \times \mathbb{N} \Rightarrow \mathbb{N}$$

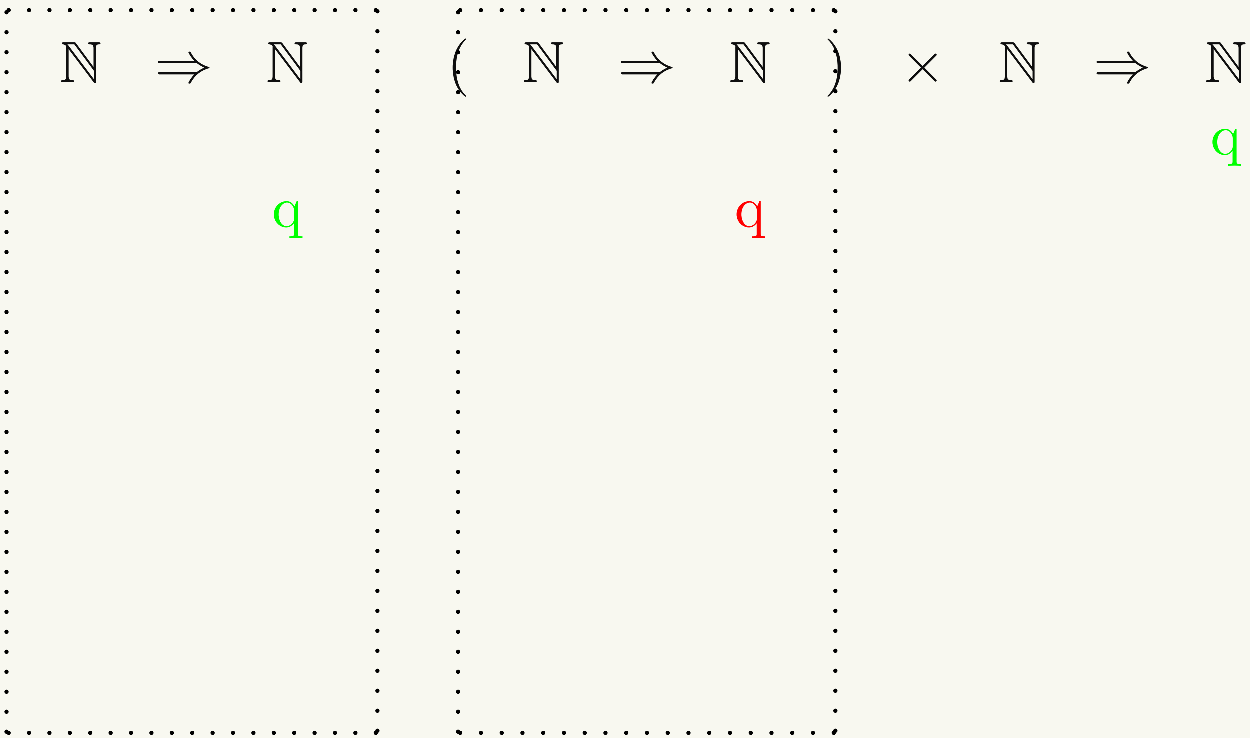
Composition

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

$$\mathbb{N} \Rightarrow \mathbb{N} \qquad (\mathbb{N} \Rightarrow \mathbb{N}) \times \mathbb{N} \Rightarrow \mathbb{N}$$

$$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad q$$

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



Composition

Introduction

Game Semantics for Programs

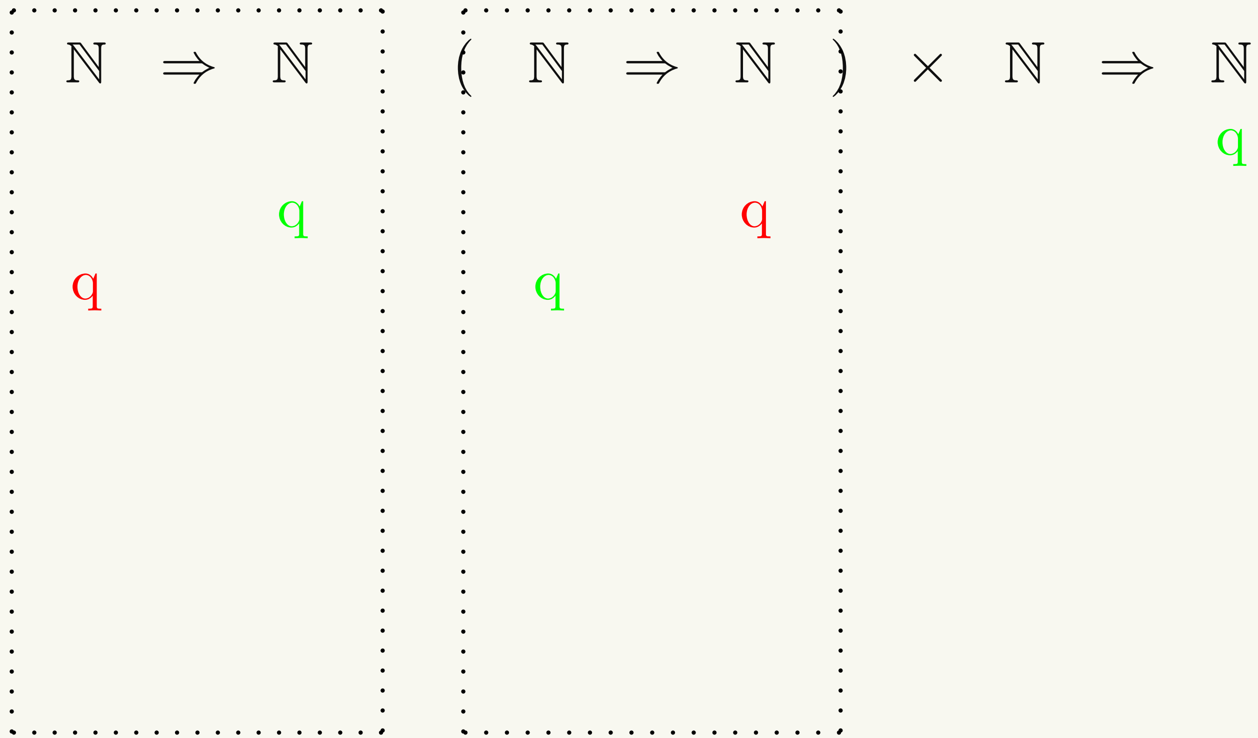
- Basic Ideas
- Types as Games
- Example

Overview

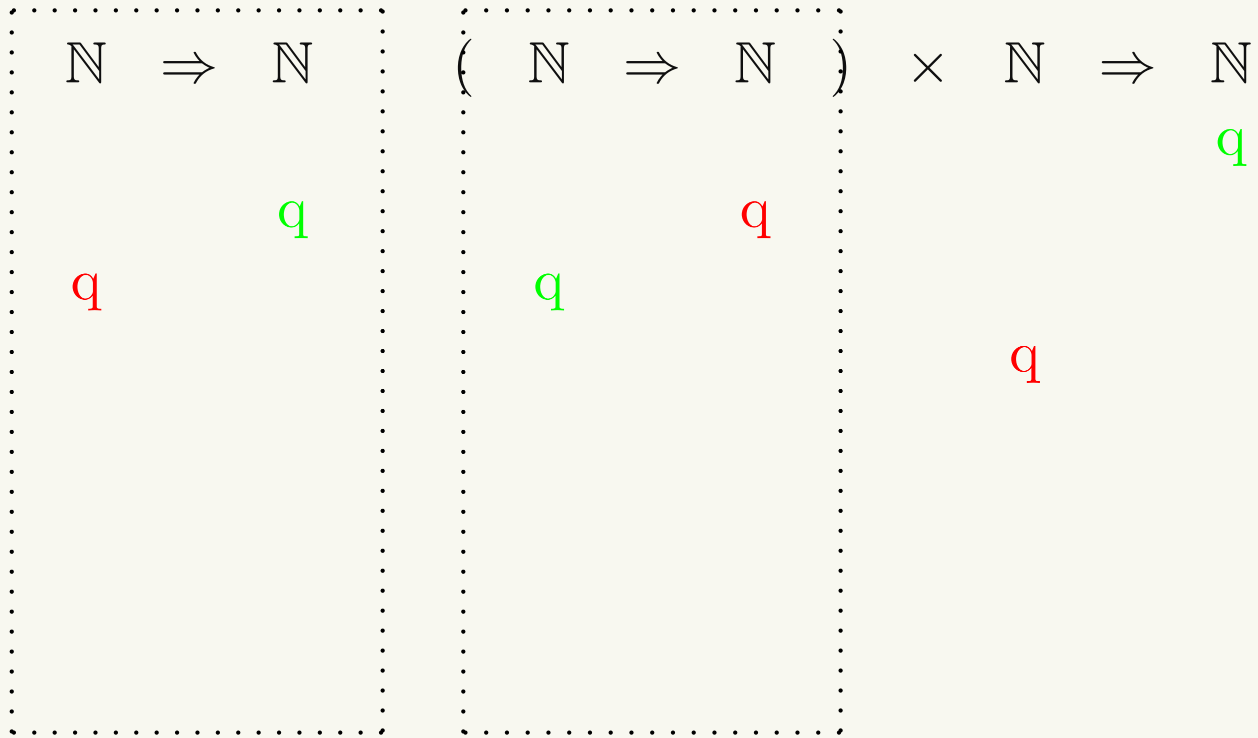
The Structure of the Games Universe: a glimpse under the hood

Copying in Game Semantics

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



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



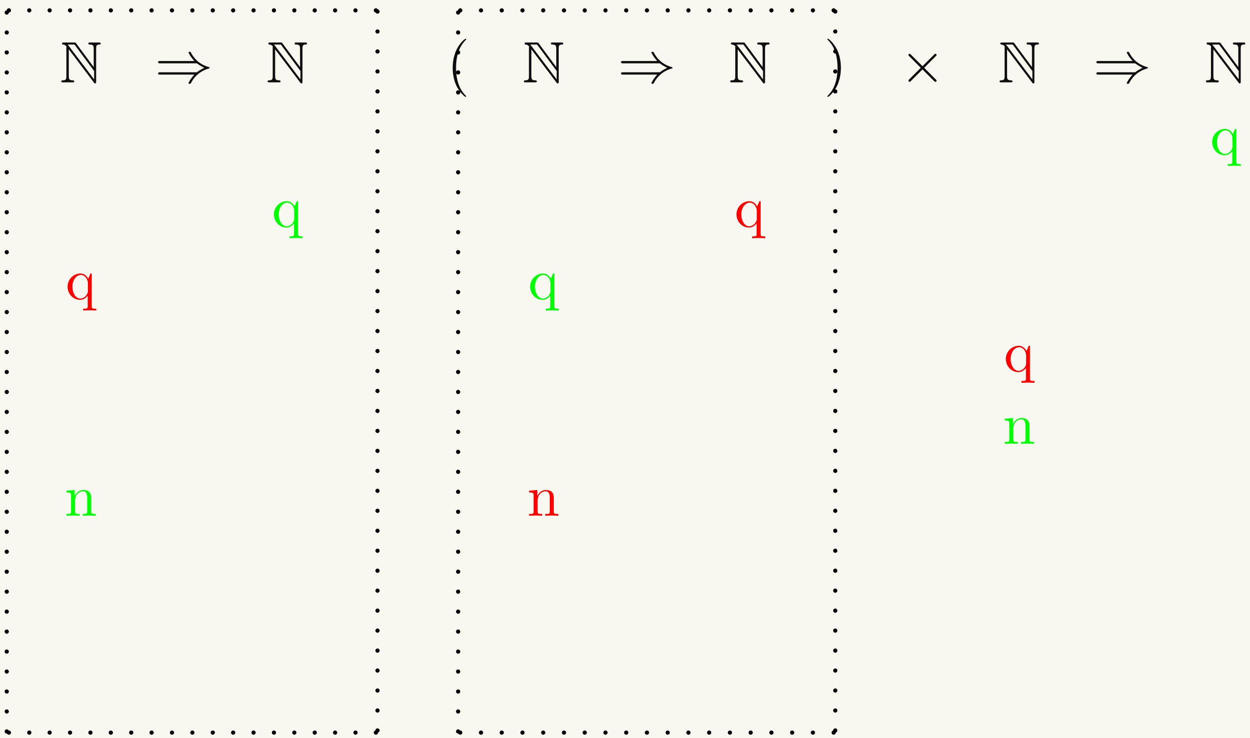
Composition

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

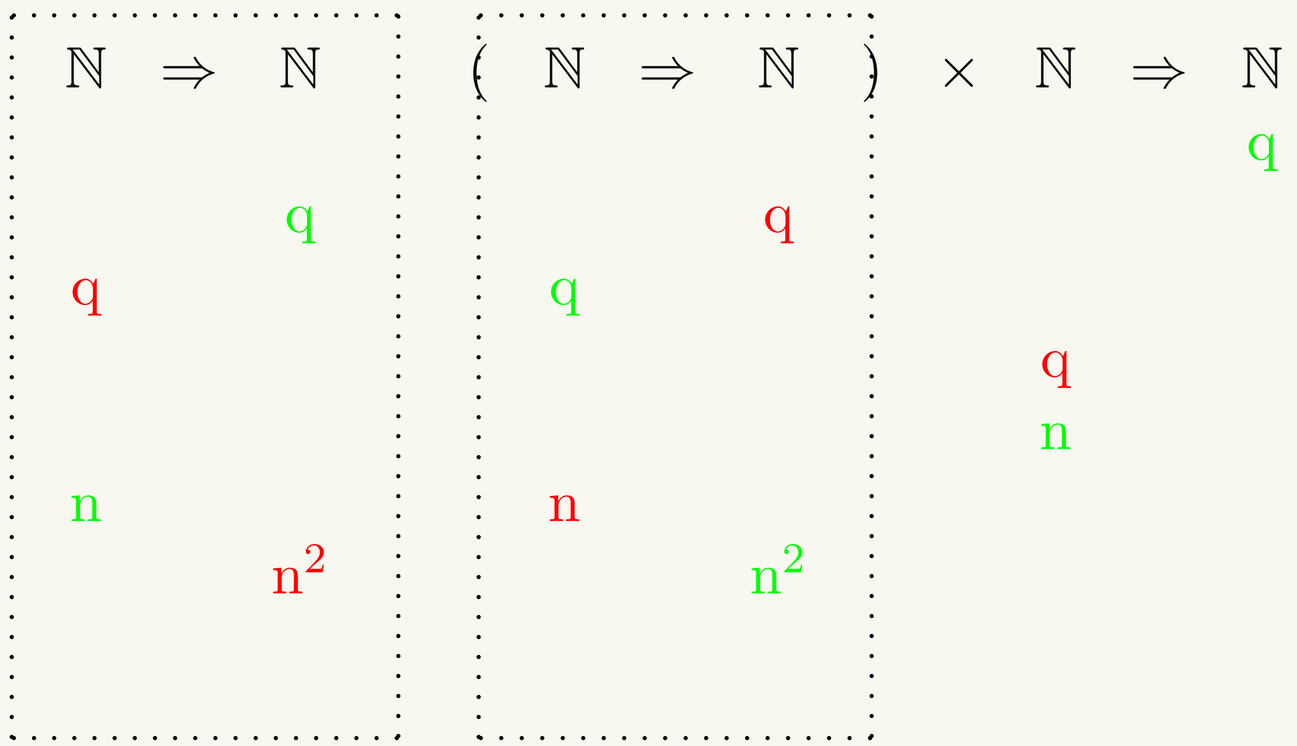
$\mathbb{N}$	$\Rightarrow$	$\mathbb{N}$	(	$\mathbb{N}$	$\Rightarrow$	$\mathbb{N}$	)	$\times$	$\mathbb{N}$	$\Rightarrow$	$\mathbb{N}$
											q
		q				q					
q				q							
									q		
									n		

Composition

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



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



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

$\mathbb{N}$	$\Rightarrow$	$\mathbb{N}$	($\mathbb{N}$	$\Rightarrow$	$\mathbb{N}$)	$\times$	$\mathbb{N}$	$\Rightarrow$	$\mathbb{N}$
									q
		q			q				
q			q						
							q		
							n		
n			n						
		n^2			n^2				
									$n^2 + 2$

Overview

Some Key Features

- Compositionality I: The importance of composition as an operation on strategies, for gluing small pieces together. This gives direct meaning to open programs, proofs from assumptions, etc.

- Compositionality I: The importance of composition as an operation on strategies, for gluing small pieces together. This gives direct meaning to open programs, proofs from assumptions, etc.

- Compositionality II: The attribution of games as meanings of types or formulas, and of strategies as meanings of terms or proofs, is done in a systematic, compositional fashion, following more generally from:

- Compositionality I: The importance of composition as an operation on strategies, for gluing small pieces together. This gives direct meaning to open programs, proofs from assumptions, etc.

- Compositionality II: The attribution of games as meanings of types or formulas, and of strategies as meanings of terms or proofs, is done in a systematic, compositional fashion, following more generally from:

- Games as a mathematical universe with its own structure, independently of any preconceived syntax. The right mathematical language for expressing this is (of course) category theory.

Some Beautiful Results of Game Semantics

Some Beautiful Results of Game Semantics

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

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

Key Examples Cartesian closed categories, Linear categories (symmmetric monoidal closed categories with monoidal adjunctions to cartesian closed categories).

Some Beautiful Results of Game Semantics

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

Key Examples Cartesian closed categories, Linear categories (symmmetric monoidal closed categories with monoidal adjunctions to cartesian closed categories).

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.

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

Key Examples Cartesian closed categories, Linear categories (symmmetric monoidal closed categories with monoidal adjunctions to cartesian closed categories).

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

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.

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 σ, there exists a proof Π of $\Gamma \vdash A$. Full completeness asks that moreover Π **denotes** the σ 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.

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 σ, there exists a proof Π of $\Gamma \vdash A$. Full completeness asks that moreover Π **denotes** the σ 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.

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 σ, there exists a proof Π of $\Gamma \vdash A$. Full completeness asks that moreover Π **denotes** the σ 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.

There are results of this kind now for a range of logics and logical type theories, including:

- Simply typed and polymorphic lambda calculus
- The lambda-mu calculus
- Various fragments of Linear 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:

The Game Semantics Landscape

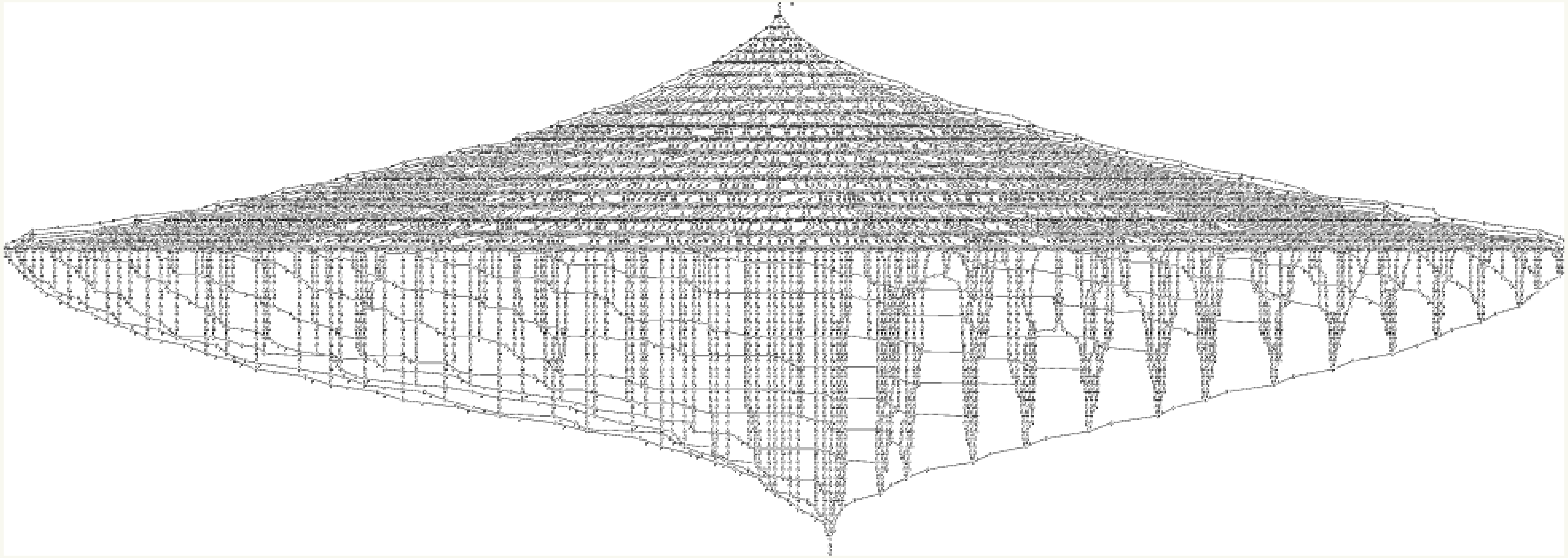
Game semantics has proved to be a flexible and powerful paradigm for constructing highly structured fully abstract semantics for languages with a wide range of computational features:

- (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
- names and freshness

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.

The key construct is **composition**; the corresponding construction on automata is 'product automaton plus hiding'.

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: 5.5×10^{12} states
- model: $6,393$ states
- max space: $1,153,240$ states

Hiding local state!

Current Work

Current Work

- Model-checking: Ghica (Birmingham):
  - State-of-the-art tool MAGE.
  - Earlier tool: GameChecker (FDR based)
  - Related work by Lazic and Dimovski (Warwick).

Current Work

- Model-checking: Ghica (Birmingham):
  - State-of-the-art tool MAGE.
  - Earlier tool: GameChecker (FDR based)
  - Related work by Lazic and Dimovski (Warwick).
- Andrzej Murawski, Joel Ouaknine and Ben Worrell: automated verification of probabilistic programs.

- Model-checking: Ghica (Birmingham):
  - State-of-the-art tool MAGE.
  - Earlier tool: GameChecker (FDR based)
  - Related work by Lazic and Dimovski (Warwick).
- Andrzej Murawski, Joel Ouaknine and Ben Worrell: automated verification of probabilistic programs.
- Luke Ong, Andrzej Murawski: extensive applications of Game Semantics to proving theoretical results on complexity of verification problems. A beautiful combination of game-semantic and automata-theoretic methods.

Game Semantics in the Games landscape

Some comparisons:

Some comparisons:

- Hintikka GTS and IF logic. GS is more compositional; a proper analysis of implication!

Some comparisons:

- Hintikka GTS and IF logic. GS is more compositional; a proper analysis of implication!
- Lorenzen school of dialogue games. An ancestor; GS is more compositional, ‘syntax-free’, much wider scope.

Some comparisons:

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

Some comparisons:

- Hintikka GTS and IF logic. GS is more compositional; a proper analysis of implication!
- Lorenzen school of dialogue games. An ancestor; GS is more compositional, 'syntax-free', much wider scope.
- Blass games. Another ancestor. Overcomes problems with compositionality.
- Kleene oracle semantics for higher-type recursive functionals. Fixes a number of problems. again fundamentally related to composition and substitution.

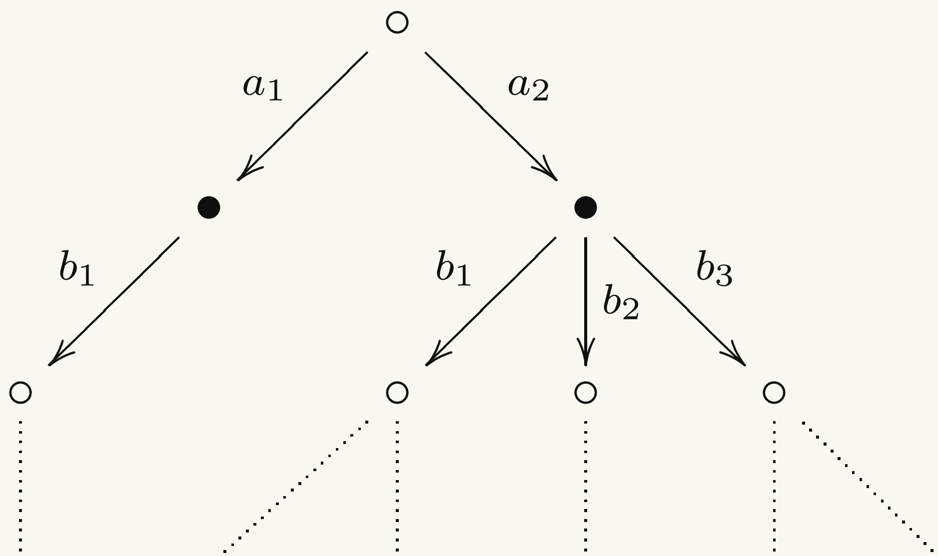
Some comparisons:

- Hintikka GTS and IF logic. GS is more compositional; a proper analysis of implication!
- Lorenzen school of dialogue games. An ancestor; GS is more compositional, 'syntax-free', much wider scope.
- Blass games. Another ancestor. Overcomes problems with compositionality.
- Kleene oracle semantics for higher-type recursive functionals. Fixes a number of problems. again fundamentally related to composition and substitution.

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.

The Structure of the Games Universe:
a glimpse under the hood

A game specifies the set of possible runs (or ‘plays’). It can be thought of as a tree



- nodes ○ are Opponent positions
- nodes ● are Player positions
- arcs are labelled with moves

Formal definition of games

Formally, we define a game G to be a structure (M_G, λ_G, P_G), where

- M_G is the set of *moves* of the game;
- $\lambda_G : M_G \longrightarrow \{P, O\}$ is a labelling function designating each move as by Player or Opponent;
- $P_G \subseteq^{\text{nepref}} M_G^{\text{alt}}$, *i.e.* P_G is a non-empty, prefix-closed subset of M_G^{alt}, the set of alternating sequences of moves in M_G.

Alternating Sequences

More formally, M_G^{alt} is the set of all $s \in M_G^*$ such that

$$\forall i : 1 \leq i \leq |s| \qquad \begin{array}{rl} & \text{even}(i) \Longrightarrow \lambda_G(s_i) = P \\ \wedge & \text{odd}(i) \Longrightarrow \lambda_G(s_i) = O \end{array}$$

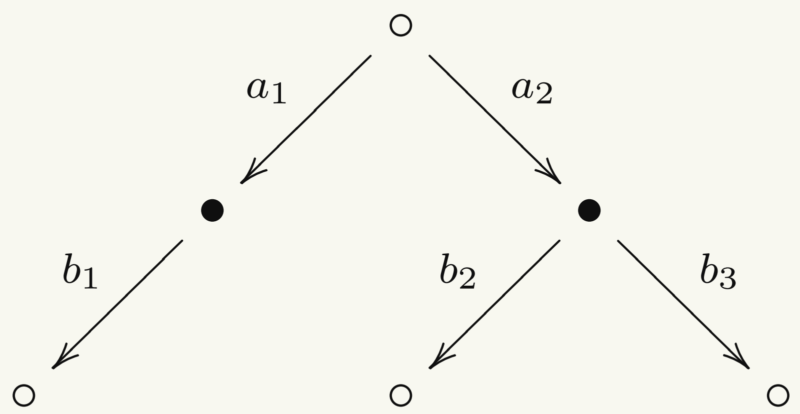
$$\begin{array}{cccccccc} s & = & a_1 & a_2 & \cdots & a_{2k+1} & a_{2k+2} & \cdots . \\ & \lambda_G & \downarrow & \downarrow & & \downarrow & \downarrow & \\ & & O & P & & O & P & \end{array}$$

The game

$$(\{a_1, a_2, b_1, b_2, b_3\}, \lambda, \{\epsilon, a_1, a_1b_1, a_2, a_2b_2, a_2b_3\})$$

$$\lambda \quad : \quad a_1, a_2 \mapsto O, \quad b_1, b_2, b_3 \mapsto P$$

represents the tree



Strategies

Formally, we define a (deterministic) strategy σ on a game G to be a non-empty subset $\sigma \subseteq P_G^{\mathrm{even}}$ of the game tree, satisfying:

$$
\begin{array}{ll}
(\mathbf{s1}) & \epsilon \in \sigma \\
(\mathbf{s2}) & sab \in \sigma \implies s \in \sigma \\
(\mathbf{s3}) & sab, sac \in \sigma \implies b = c.
\end{array}
$$

To understand this definition, think of

$$s = a_1 b_1 \cdots a_k b_k \in \sigma$$

as a record of repeated interactions with the Environment following σ. It can be read as follows:

If the Environment initially does a_1,
then respond with b_1;
If the Environment then does a_2,
then respond with b_2;
⋮
If the Environment finally does a_k,
then respond with b_k.

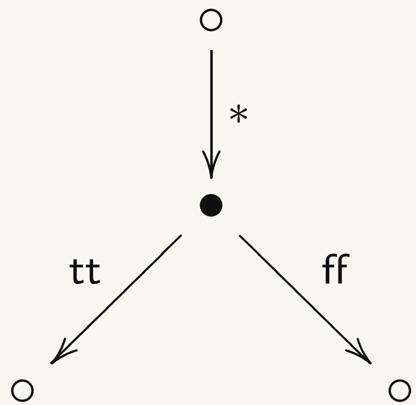
The first two conditions on σ say that it is a sub-tree of P_G of even-length paths. The third is a determinacy condition.

This can be seen as generalizing the notion of graph of a relation, *i.e.* of a set of ordered pairs, which can be read as a set of stimulus-response instructions. The generalization is that ordinary relations describe a single stimulus-response event only (giving rules for what the response to any given stimulus may be), whereas strategies describe repeated interactions between the System and the Environment. We can regard $sab \in \sigma$ as saying: 'when given the stimulus a in the context s, respond with b'. Note that, with this reading, the condition (s3) generalizes the usual single-valuedness condition for (the graphs of) partial functions. Thus a useful slogan is:

> "Strategies are (partial) functions extended in time."

Let $\mathbb{B}$ be the game

$$(\{*, \mathsf{tt}, \mathsf{ff}\}, \{* \mapsto O, \mathsf{tt} \mapsto P, \mathsf{ff} \mapsto P\}, \{\epsilon, *, *\mathsf{tt}, *\mathsf{ff}\})$$

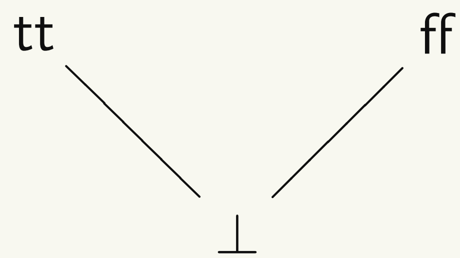


This game can be seen as representing the data type of booleans. The opening move $*$ is a request by Opponent for the data, which can be answered by either $\mathtt{tt}$ or ff by Player.

Strategies on $\mathbb{B}$

$$\{\epsilon\} \quad \mathrm{Pref}\{*\mathtt{tt}\} \quad \mathrm{Pref}\{*\mathsf{ff}\}$$

The first of these is the undefined strategy ('$\bot$'), the second and third correspond to the boolean values $\mathtt{tt}$ and ff. Taken with the inclusion ordering, this "space of strategies" corresponds to the usual flat domain of booleans:



We will now describe some fundamental constructions on games.

Tensor Product Given games A, B, we describe the tensor product $A \otimes B$.

$$
\begin{array}{rcl}
M_{A\otimes B} & = & M_A + M_B \\
\lambda_{A\otimes B} & = & [\lambda_A, \lambda_B] \\
P_{A\otimes B} & = & \{s \in M^{\mathrm{alt}}_{A\otimes B} \mid s{\restriction}M_A \in P_A \wedge s{\restriction}M_B \in P_B\}
\end{array}
$$

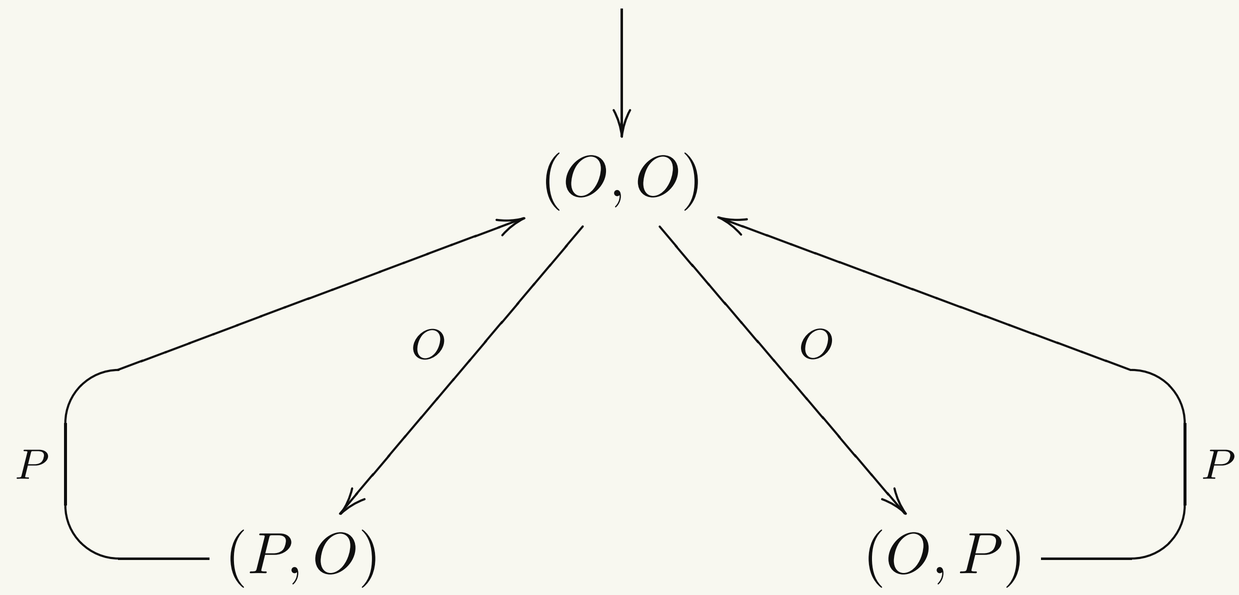
We can think of $A \otimes B$ as allowing play to proceed in *both* the subgames A and B in an interleaved fashion. It is a form of 'disjoint (*i.e.* non-communicating or interacting) parallel composition'.

Switching Condition for Tensor Product

A first hint of the additional subtleties introduced by the explicit representation of both System and Environment is given by the following result.

Proposition 1 *(Switching condition)*
In any play $s \in P_{A \otimes B}$, if successive moves s_i, s_{i+1} are in different subgames (i.e. *one is in A and the other in B*), *then $\lambda_{A \otimes B}(s_i) = P$, $\lambda_{A \otimes B}(s_{i+1}) = O$. In other words, only Opponent can switch from one subgame to another; Player must always respond in the same subgame that Opponent just moved in.*

State transition diagram for Tensor Product



We see immediately from this that the switching condition holds; and also that the state (P, P) can never be reached (*i.e.* for no $s \in P_{A \otimes B}$ is $\ulcorner s \urcorner = (P, P)$).

Linear Implication

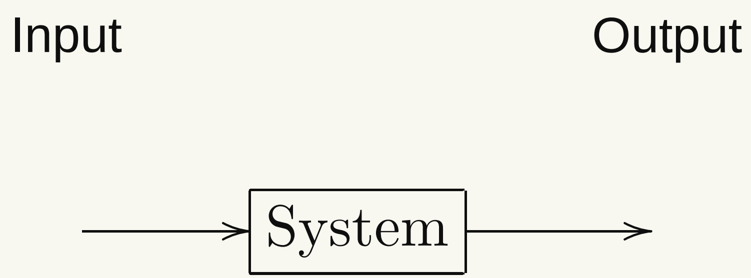
Given games A, B, we define the game $A \multimap B$ as follows:

$$
\begin{array}{lcl}
M_{A \multimap B} & = & M_A + M_B \\
\lambda_{A \otimes B} & = & [\overline{\lambda_A}, \lambda_B] \qquad \text{where } \overline{\lambda}_A(m) = \begin{cases} P \text{ when } \lambda_A(m) = O \\ O \text{ when } \lambda_A(m) = P \end{cases} \\
P_{A \multimap B} & = & \{ s \in M^{\text{alt}}_{A \multimap B} \mid s \restriction M_A \in P_A \wedge s \restriction M_B \in P_B \}
\end{array}
$$

This definition is *almost* the same as that of $A \otimes B$. The crucial difference is the inversion of the labelling function on the moves of A, corresponding to the idea that on the left of the arrow the rôles of Player and Opponent are interchanged.

Linear Implication Continued

If we think of 'function boxes', this is clear enough:



On the output side, the System is the producer and the Environment is the consumer; these rôles are reversed on the input side.

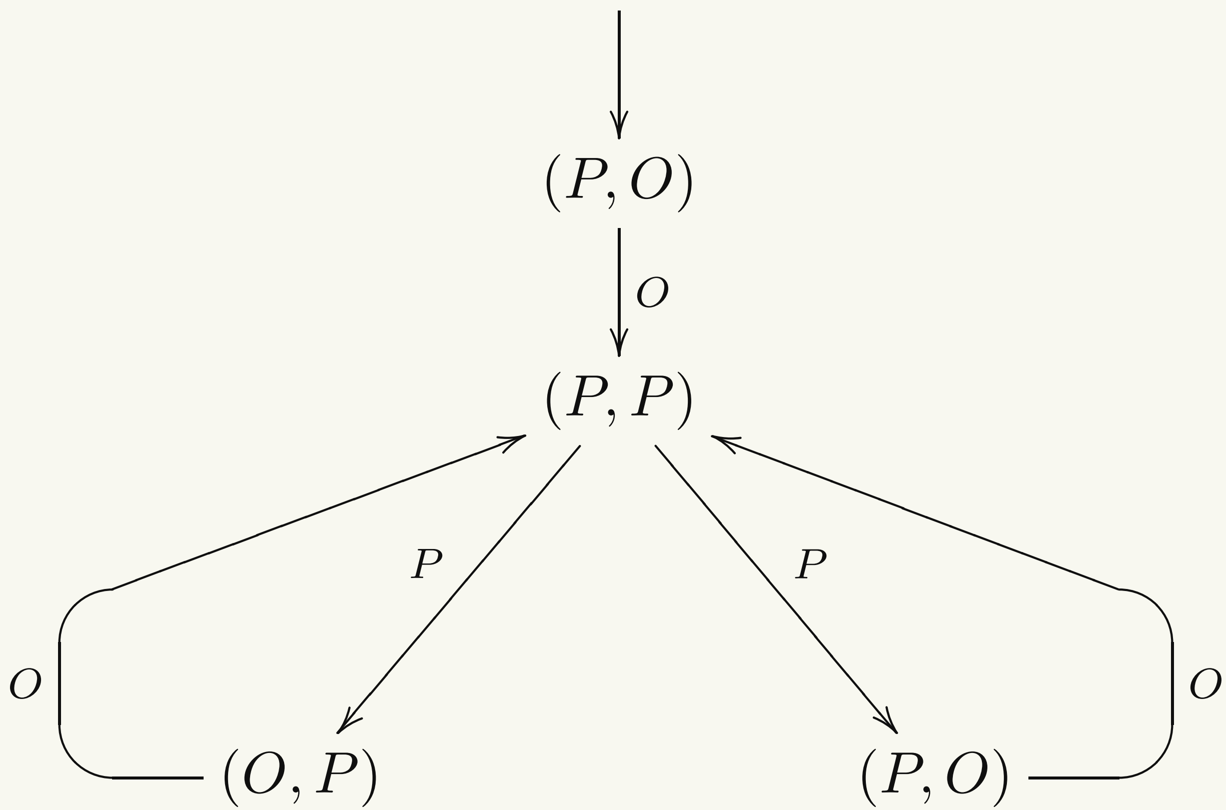
Note that $M^{\mathrm{alt}}_{A \multimap B}$, and hence $P_{A \multimap B}$, are in general quite different to $M^{\mathrm{alt}}_{A \otimes B}$, $P_{A \otimes B}$ respectively. In particular, the first move in $P_{A \multimap B}$ must always be in B, since the first move must be by Opponent, and all opening moves in A are labelled P by $\overline{\lambda_A}$.

Switching Condition for Linear Implication

We obtain the following switching condition for $A \multimap B$:

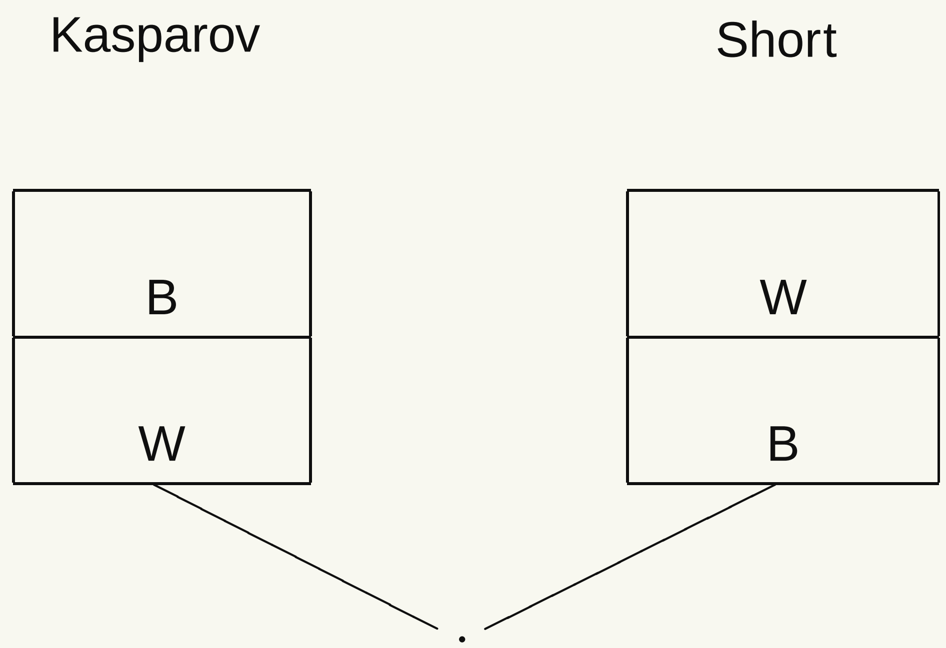
> If two consecutive moves are in different components, the first was by Opponent and the second by Player; so only Player can switch components.

This is supported by the following state-transition diagram:

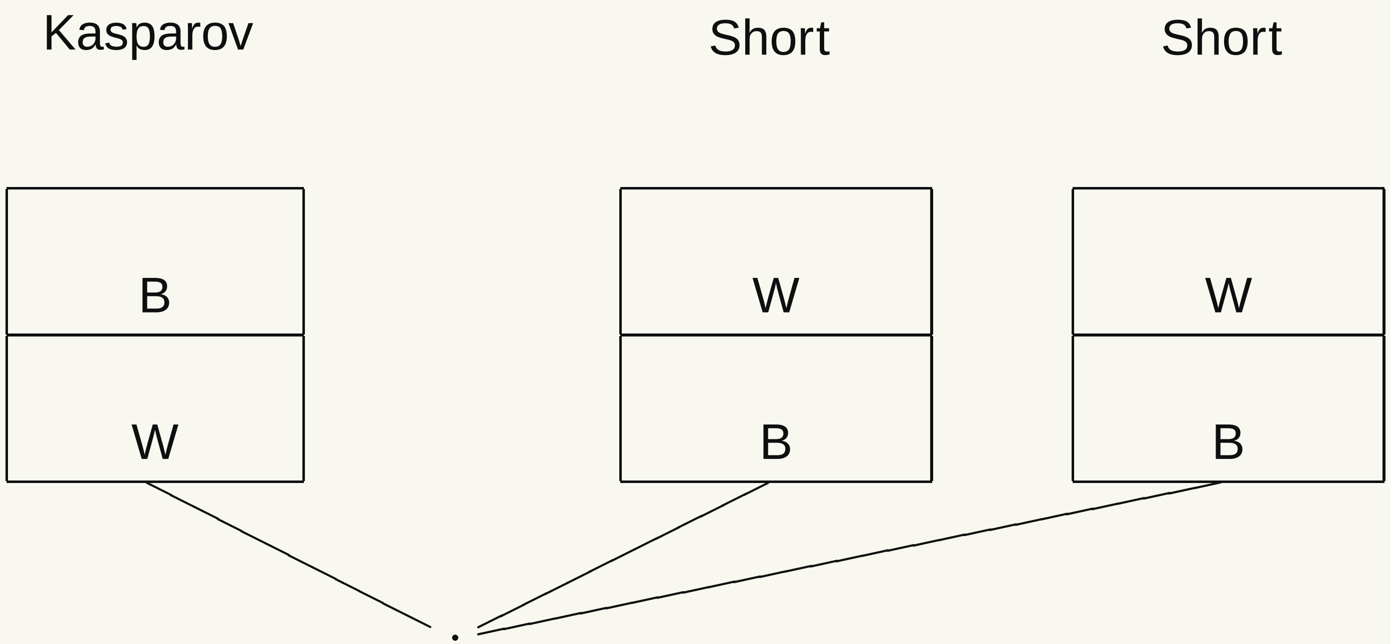


The Copy-Cat Strategy

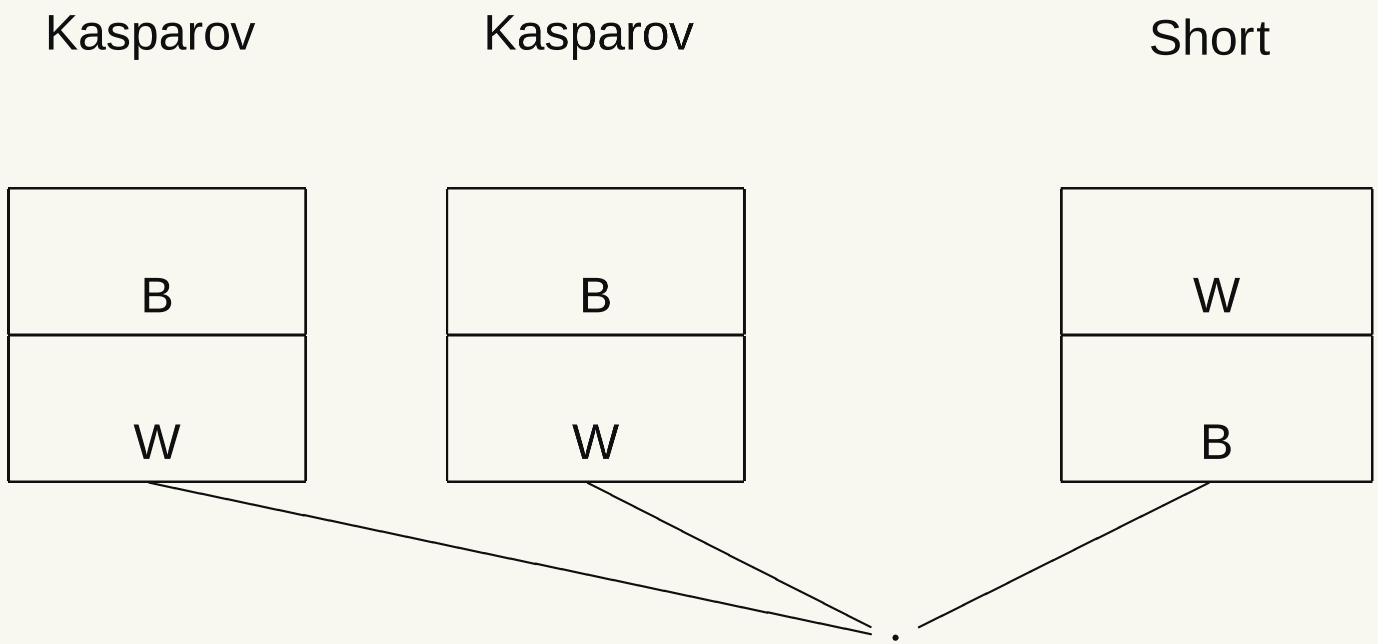
How to beat an International Grand-Master at chess by the power of Logic.



Does Copy-Cat still work here?



And here?



General definition of Copy-Cat strategy

Time	A	$\multimap$	A	
1			a_1	O
2	a_1			P
3	a_2			O
4			a_2	P
$\vdots$		$\vdots$		$\vdots$

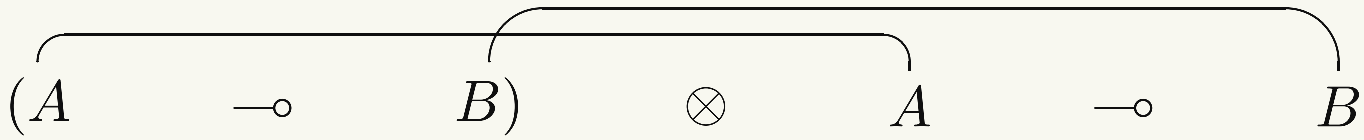
$$\mathrm{id}_A = \{s \in P^{\mathrm{even}}_{A_1 \multimap A_2} \mid \forall t \text{ even-length prefix of } s:\ t{\restriction}A_1 = t{\restriction}A_2\}$$

We indicate such a strategy briefly by $\overbrace{A \multimap A}$

Application (*Modus Ponens*)

$$\mathtt{Ap}_{A,B} : (A \multimap B) \otimes A \multimap B$$

This is the conjunction of two copy-cat strategies



Note that A and B each occur once positively and once negatively in this formula; we simply connect up the positive and negative occurrences by 'copy-cats'.

$$\mathtt{Ap}_{A,B} = \{s \in P^{\text{even}}_{(A_1 \multimap B_1)\otimes A_2 \multimap B_2} \mid \forall t \text{ even-length prefix of } s : \\ t{\upharpoonright}A_1 = t{\upharpoonright}A_2 \;\wedge\; t{\upharpoonright}B_1 = t{\upharpoonright}B_2\}$$

The `Ap` strategy as a protocol for (linear) function application.

	$(A$	$\multimap$	$B)$	$\otimes$	A	$\multimap$	B
O							ro
P			ro				
O	ri						
P					ri		
O					id		
P	id						
O			od				
P							od

ro — request output
ri — request input
id — input data
od — output data

- Objects: Games
- Morphisms: $\sigma : A \longrightarrow B$ are strategies σ on $A \multimap B$.
- Composition: **interaction between strategies**.

$$\frac{\sigma : A \to B \quad \tau : B \to C}{\sigma ; \tau : A \to C}$$

Composition as Interaction: The idea

A	$\overset{\sigma}{\multimap}$	B		B	$\overset{\tau}{\multimap}$	C
						c_1
				b_1		
		b_1				
		b_2				
				b_2		
		$\vdots$		$\vdots$		
				b_k		
		b_k				
a_1						

Continuing in this way, we obtain a uniquely determined sequence.

$$c_1 b_1 b_2 \cdots b_k \cdots$$

If the sequence ends in a visible action in A or C, this is the response by the strategy $\sigma;\tau$ to the initial move c_1, with the internal dialogue between σ and τ in B being hidden from the Environment. Note that σ and τ may continue their internal dialogue in B forever. This is "infinite chattering" in CSP terminology, and "divergence by an infinite τ-computation" in CCS terminology.

Composition Continued

Continuing in this way, we obtain a uniquely determined sequence.

$$c_1 b_1 b_2 \cdots b_k \cdots$$

If the sequence ends in a visible action in A or C, this is the response by the strategy $\sigma;\tau$ to the initial move c_1, with the internal dialogue between σ and τ in B being hidden from the Environment. Note that σ and τ may continue their internal dialogue in B forever. This is "infinite chattering" in CSP terminology, and "divergence by an infinite τ-computation" in CCS terminology.

As this discussion clearly shows composition in $\mathcal{G}$ expresses interaction between strategies.

'Parallel Composition + Hiding'

$$\frac{\sigma : A \rightarrow B \quad \tau : B \rightarrow C}{\sigma ; \tau : A \rightarrow C}$$

$$\sigma;\tau \;=\; (\sigma \parallel \tau)/B \;=\; \{s{\restriction}A, C \mid s \in \sigma \parallel \tau\}$$

$$\sigma \parallel \tau \;=\; \{s \in (M_A + M_B + M_C)^* \mid s{\restriction}A, B \in \sigma \;\wedge\; s{\restriction}B, C \in \tau\}.$$

'Parallel Composition + Hiding'

$$\frac{\sigma : A \rightarrow B \quad \tau : B \rightarrow C}{\sigma ; \tau : A \rightarrow C}$$

$$\sigma ; \tau \ = \ (\sigma \parallel \tau)/B \ = \ \{s{\restriction}A, C \mid s \in \sigma \parallel \tau\}$$

$$\sigma \parallel \tau \ = \ \{s \in (M_A + M_B + M_C)^* \mid s{\restriction}A, B \in \sigma \ \wedge \ s{\restriction}B, C \in \tau\}.$$

(Note that we extend our abuse of notation for restriction here; by $s{\restriction}A, B$ we mean the restriction of s to $M_A + M_B$ as a "subset" of $M_A + M_B + M_C$, and similarly for $s{\restriction}A, C$ and $s{\restriction}B, C$.)

Proposition 2 $\mathcal{G}$ *is a category.*

Proposition 3 *$\mathcal{G}$ is a category.*

In particular, $\mathtt{id}_A : A \longrightarrow A$ is the copy-cat strategy described previously.

Proposition 4 *$\mathcal{G}$ is a category.*

In particular, $\mathtt{id}_A : A \longrightarrow A$ is the copy-cat strategy described previously.

Composition and Copy-Cat (Identity Axiom and Cut) are two sides of the same coin:

- Copy-cat makes the same thing happen in two different places
- Composition makes two different things happen in the same place (the ‘locus of interaction’).

Tensor structure of $\mathcal{G}$

We have already defined the tensor product $A \otimes B$ on objects. Now we extend it to morphisms:

Tensor structure of $\mathcal{G}$

We have already defined the tensor product $A \otimes B$ on objects. Now we extend it to morphisms:

$$\frac{\sigma : A \to B \quad \tau : A' \to B'}{\sigma \otimes \tau : A \otimes A' \to B \otimes B'}$$

$$\sigma \otimes \tau \;=\; \{s \in P^{\text{even}}_{A \otimes A' \multimap B \otimes B'} \mid s \upharpoonright A, B \in \sigma \;\wedge\; s \upharpoonright A', B' \in \tau\}.$$

Tensor structure of $\mathcal{G}$

We have already defined the tensor product $A \otimes B$ on objects. Now we extend it to morphisms:

$$\frac{\sigma : A \rightarrow B \quad \tau : A' \rightarrow B'}{\sigma \otimes \tau : A \otimes A' \rightarrow B \otimes B'}$$

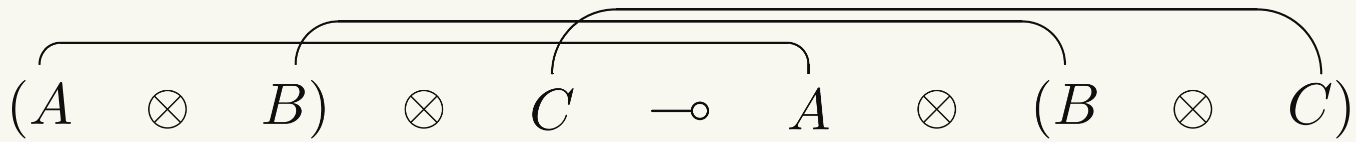
$$\sigma \otimes \tau \;=\; \{s \in P^{\text{even}}_{A \otimes A' \multimap B \otimes B'} \mid s \upharpoonright A, B \in \sigma \;\wedge\; s \upharpoonright A', B' \in \tau\}.$$

This can be seen as disjoint (i.e. non-communicating) parallel composition of σ and τ.

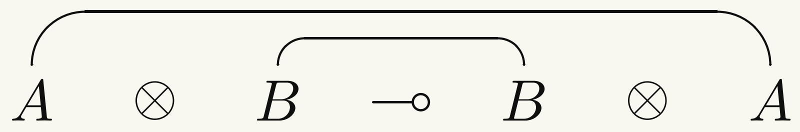
‘Canonical isomorphisms’ for monoidal structure

These arise as conjunctions of copy-cat strategies.

$$\mathtt{assoc}_{A,B,C} : \qquad (A \otimes B) \otimes C \xrightarrow{\sim} A \otimes (B \otimes C)$$



$$\mathtt{symm}_{A,B} : \qquad A \otimes B \xrightarrow{\sim} B \otimes A$$



The application (or evaluation) morphisms

$$\mathtt{Ap}_{A,B} : (A \multimap B) \otimes A \longrightarrow B$$

have already been defined. For currying, given

$$\sigma : A \otimes B \multimap C$$

define

$$\Lambda(\sigma) : A \longrightarrow (B \multimap C)$$

by

$$\Lambda(\sigma) = \{\alpha^*(s) \mid s \in \sigma\}$$

where $\alpha : (M_A + M_B) + M_C \xrightarrow{\sim} M_A + (M_B + M_C)$ is the canonical isomorphism in $\mathbf{Set}$.

Copying in Game Semantics

The resource sensitivity of games means that copying does not come for free; but it **can** be modelled explicitly.

The resource sensitivity of games means that copying does not come for free; but it **can** be modelled explicitly.

We begin with a simpler construction: the 'Tensor product of countably many copies of A', which we write as $\bigotimes^{\omega} A$:

Interpreting the Linear exponential $!$ in $\mathcal{G}$

The resource sensitivity of games means that copying does not come for free; but it **can** be modelled explicitly.

We begin with a simpler construction: the 'Tensor product of countably many copies of A', which we write as $\otimes^{\omega} A$:

- $M_{\otimes^{\omega} A} = \mathbb{N} \times M_A$, *i.e.* the disjoint union of countably many copies of M_A.
- $\lambda_{\otimes^{\omega} A}(n, a) = \lambda_A(a)$.
- $P_{\otimes^{\omega} A}$ is the set of all alternating sequences of moves in $M_{\otimes^{\omega} A}$ such that for all n, $s{\restriction}n \in P_A$.

The resource sensitivity of games means that copying does not come for free; but it **can** be modelled explicitly.

We begin with a simpler construction: the 'Tensor product of countably many copies of A', which we write as $\otimes^\omega A$:

- $M_{\otimes^\omega A} = \mathbb{N} \times M_A$, *i.e.* the disjoint union of countably many copies of M_A.
- $\lambda_{\otimes^\omega A}(n, a) = \lambda_A(a)$.
- $P_{\otimes^\omega A}$ is the set of all alternating sequences of moves in $M_{\otimes^\omega A}$ such that for all n, $s{\restriction}n \in P_A$.

(Switching conditions?)

We can define a copying strategy

$$\delta_A : \otimes^\omega A \longrightarrow \otimes^\omega A \otimes \otimes^\omega A$$

using the bijection on moves

$$\begin{array}{rcl} M_{\otimes^\omega A} & \equiv & \mathbb{N} \times M_A \\ & \cong & (\mathbb{N} + \mathbb{N}) \times A \\ & \cong & \mathbb{N} \times M_A + \mathbb{N} \times M_A \\ & \cong & M_{\otimes^\omega A} + M_{\otimes^\omega A} \end{array}$$

based on the (rather: a) bijection $\mathbb{N} \cong \mathbb{N} + \mathbb{N}$ (‘Hilbert hotel’).

We can define a copying strategy

$$\delta_A : \bigotimes^{\omega} A \longrightarrow \bigotimes^{\omega} A \otimes \bigotimes^{\omega} A$$

using the bijection on moves

$$\begin{array}{rcl} M_{\bigotimes^{\omega} A} & \equiv & \mathbb{N} \times M_A \\ & \cong & (\mathbb{N} + \mathbb{N}) \times A \\ & \cong & \mathbb{N} \times M_A + \mathbb{N} \times M_A \\ & \cong & M_{\bigotimes^{\omega} A} + M_{\bigotimes^{\omega} A} \end{array}$$

based on the (rather: a) bijection $\mathbb{N} \cong \mathbb{N} + \mathbb{N}$ (‘Hilbert hotel’).

The strategy simply plays copy-cat between the copies of A paired by this bijection.

Co-Monadic Operations

We also have the following operations:

Co-Monadic Operations

We also have the following operations:

Dereliction: $\mathtt{der}_A : \otimes^\omega A \longrightarrow A$.
We choose some index, e.g. 0, to play copycat with.

We also have the following operations:

Dereliction: $\mathtt{der}_A : \bigotimes^\omega A \longrightarrow A$.
We choose some index, e.g. 0, to play copycat with.

Promotion: $\bigotimes^\omega A \longrightarrow \bigotimes^\omega \bigotimes^\omega A$.
Another copycat based on the bijection of moves

$$\begin{aligned} M_{\bigotimes^\omega A} &\equiv \mathbb{N} \times M_A \\ &\cong (\mathbb{N} \times \mathbb{N}) \times M_A \\ &\cong \mathbb{N} \times (\mathbb{N} \times M_A) \\ &\equiv M_{\bigotimes^\omega \bigotimes^\omega A} \end{aligned}$$

using some pairing function $\mathbb{N} \cong \mathbb{N} \times \mathbb{N}$.

Co-Monadic Operations

We also have the following operations:

Dereliction: $\mathtt{der}_A : \bigotimes^{\omega} A \longrightarrow A$.
We choose some index, e.g. 0, to play copycat with.

Promotion: $\bigotimes^{\omega} A \longrightarrow \bigotimes^{\omega} \bigotimes^{\omega} A$.
Another copycat based on the bijection of moves

$$\begin{array}{rcl} M_{\bigotimes^{\omega} A} & \equiv & \mathbb{N} \times M_A \\ & \cong & (\mathbb{N} \times \mathbb{N}) \times M_A \\ & \cong & \mathbb{N} \times (\mathbb{N} \times M_A) \\ & \equiv & M_{\bigotimes^{\omega} \bigotimes^{\omega} A} \end{array}$$

using some pairing function $\mathbb{N} \cong \mathbb{N} \times \mathbb{N}$.

Functorial action:

$$\frac{\sigma : A \longrightarrow B}{\bigotimes^{\omega}\sigma : \bigotimes^{\omega} A \longrightarrow \bigotimes^{\omega} B}$$

playing σ in each index.

Comonads are the categorical or type-theoretic equivalents of S4 necessity modalities.

Cf.

$$\Box A \longrightarrow A$$
$$\Box A \longrightarrow \Box\Box A$$
$$(A \longrightarrow B) \longrightarrow (\Box A \longrightarrow \Box B)$$

Comonads are the categorical or type-theoretic equivalents of S4 necessity modalities.
Cf.

$$\Box A \longrightarrow A$$
$$\Box A \longrightarrow \Box\Box A$$
$$(A \longrightarrow B) \longrightarrow (\Box A \longrightarrow \Box B)$$

However: this treatment of copying based on $\otimes^{\omega}$ is all coding-dependent and does not satisfy the appropriate equational properties.

Comonads are the categorical or type-theoretic equivalents of S4 necessity modalities.
Cf.

$$\Box A \longrightarrow A$$
$$\Box A \longrightarrow \Box\Box A$$
$$(A \longrightarrow B) \longrightarrow (\Box A \longrightarrow \Box B)$$

However: this treatment of copying based on $\otimes^{\omega}$ is all coding-dependent and does not satisfy the appropriate equational properties.

The basic reason that we are not content with $\otimes^{\omega} A$ is that the various copies have distinct 'identities' via their indices $i \in \mathbb{N}$.

Comonads for Modal Logicians

Comonads are the categorical or type-theoretic equivalents of S4 necessity modalities.
Cf.

$$\Box A \rightarrow A$$
$$\Box A \rightarrow \Box\Box A$$
$$(A \rightarrow B) \rightarrow (\Box A \rightarrow \Box B)$$

However: this treatment of copying based on $\bigotimes^\omega$ is all coding-dependent and does not satisfy the appropriate equational properties.

The basic reason that we are not content with $\bigotimes^\omega A$ is that the various copies have distinct 'identities' via their indices $i \in \mathbb{N}$.

Cf. Fock space in quantum physics: bosons (such as electrons) do not have individual identities.

Copying is comonoidal

Copying is comonoidal

In any category with products, the diagonal — which expresses copyability — has an algebraic structure; it is a **cocommutative comonoid** — the dual of a commutative monoid.

Copying is comonoidal

In any category with products, the diagonal — which expresses copyability — has an algebraic structure; it is a **cocommutative comonoid** — the dual of a commutative monoid.

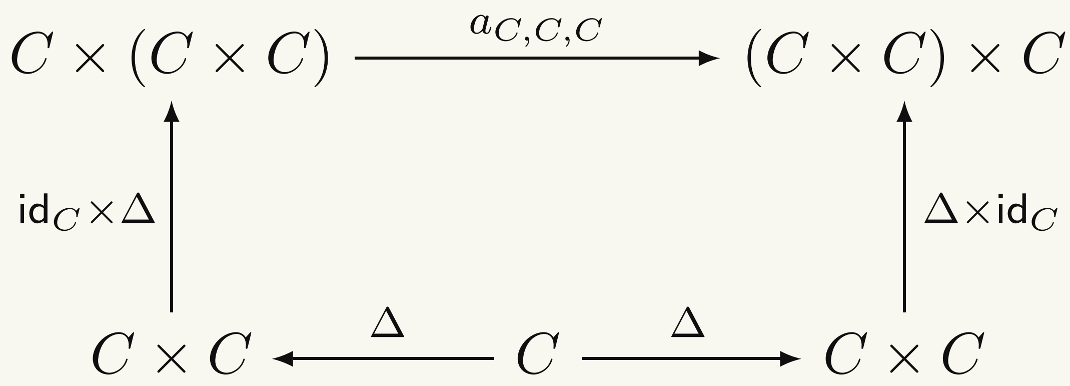
That is, we have, for any object C:

Copying is comonoidal

In any category with products, the diagonal — which expresses copyability — has an algebraic structure; it is a **cocommutative comonoid** — the dual of a commutative monoid.

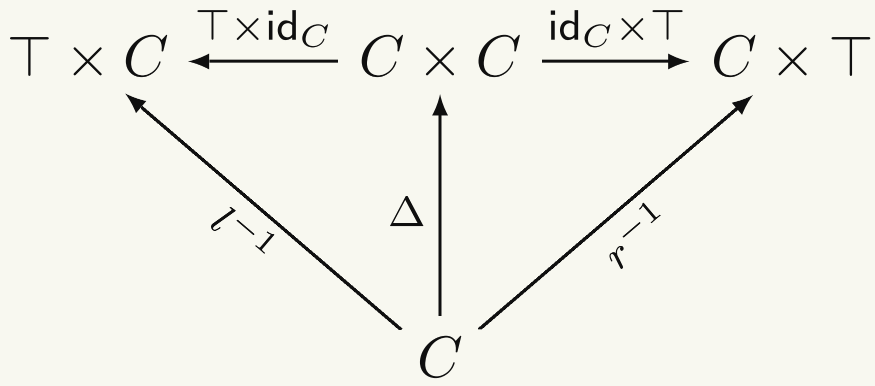
That is, we have, for any object C:

(1) **Coassociativity**

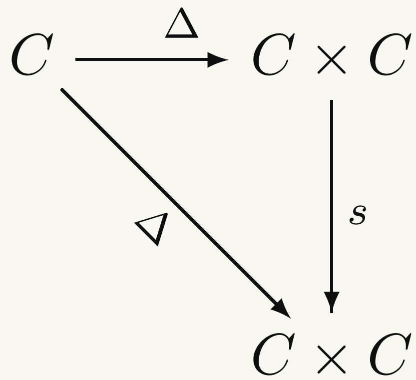


Comonoid Axioms Continued

(2) **Counit**



(3) **Cocommutativity**



The notion of cocommutative comonoid (in future: **coalgebra** for short) makes sense in **any** symmetric monoidal category.

The notion of cocommutative comonoid (in future: **coalgebra** for short) makes sense in **any** symmetric monoidal category.

Let $(\mathbf{C}, \otimes, I, a, l, r, s)$ be a symmetric monoidal category. A **comonoid** in $\mathbf{C}$ is a triple (C, δ, ϵ) where C is an object, and $\delta : C \longrightarrow C \otimes C$ and $\epsilon : C \longrightarrow I$ are morphisms satisfying the commutative diagrams for Coassociativity, Counit, and Cocommutativity.

The notion of cocommutative comonoid (in future: **coalgebra** for short) makes sense in **any** symmetric monoidal category.

Let $(\mathbf{C}, \otimes, I, a, l, r, s)$ be a symmetric monoidal category. A **comonoid** in $\mathbf{C}$ is a triple (C, δ, ϵ) where C is an object, and $\delta : C \longrightarrow C \otimes C$ and $\epsilon : C \longrightarrow I$ are morphisms satisfying the commutative diagrams for Coassociativity, Counit, and Cocommutativity.

(N.B. coalgebraic structures are important in current mathematics, e.g. Hopf algebras, Quantum groups, Frobenius algebras etc.)

Solution

We obtain the 'right' notion of $!$ with all the appropriate properties by factoring out modulo the action of permutations (of finite support) on $\mathbb{N}$. This means that indices become 'generic' and have no specific identities; the only operation available on them is comparison for equality. All operations and relations on indices are required to be **equivariant**.

Solution

We obtain the 'right' notion of $!$ with all the appropriate properties by factoring out modulo the action of permutations (of finite support) on $\mathbb{N}$. This means that indices become 'generic' and have no specific identities; the only operation available on them is comparison for equality. All operations and relations on indices are required to be **equivariant**.

The most elegant way of formulating this is in the setting of **(multi)nominal sets**.

Solution

We obtain the 'right' notion of $!$ with all the appropriate properties by factoring out modulo the action of permutations (of finite support) on $\mathbb{N}$. This means that indices become 'generic' and have no specific identities; the only operation available on them is comparison for equality. All operations and relations on indices are required to be **equivariant**.

The most elegant way of formulating this is in the setting of **(multi)nominal sets**.

Introduction to nominal sets in talk by Nikos Tzevelekos.

A Cartesian Closed Category of Games

Corresponding the the syntactic translation of $\supset$, $\wedge$ logic into Linear Logic using $\otimes$, $\multimap$, $\&$, $!$, we can build a **cartesian closed** category of games using the comonadic structure of $!$.

Corresponding the the syntactic translation of $\supset$, $\wedge$ logic into Linear Logic using $\otimes$, $\multimap$, $\&$, $!$, we can build a **cartesian closed** category of games using the comonadic structure of $!$.

We build a category $\mathcal{K}_!(\mathcal{G})$ (the 'co-Kleisli category') as follows:

Objects: same as in $\mathcal{G}$.

Morphisms: $\mathcal{K}_!(\mathcal{G})A, B = \mathcal{G}(!A, B)$.

Composition:

$$\frac{!A \xrightarrow{\sigma} B \qquad !B \xrightarrow{\tau} C}{!A \xrightarrow{\delta_A} !!A \xrightarrow{!\sigma} !B \xrightarrow{\tau} C}$$

Products: $A \times B = A \& B$

Exponentials: $A \Rightarrow B = !A \multimap B$.

Given A, B define $A \& B$ by

$$
\begin{array}{rcl}
M_{A\&B} & = & M_A + M_B \\
\lambda_{A\&B} & = & [\lambda_A, \lambda_B] \\
P_{A\&B} & = & \{\mathtt{inl}^*(s) \mid s \in P_A\} \cup \{\mathtt{inr}^*(t) \mid t \in P_B\}.
\end{array}
$$

$A\&B$ is the product of A and B in $\mathcal{G}$. We can define projections

$$A \xleftarrow{\mathsf{fst}} A\&B \xrightarrow{\mathsf{snd}} B$$

(Partial copy-cats) and pairing

$$\frac{\sigma : C \longrightarrow A \qquad \tau : C \longrightarrow B}{\langle \sigma, \tau \rangle : C \longrightarrow A \& B}$$

(Disjoint union)

Exponential isomorphisms

$$
\begin{array}{ccc}
!(A \,\&\, B) & \cong & !A \otimes !B \\
!\top & \cong & I
\end{array}
$$

Exponential isomorphisms

$$
\begin{array}{rcl}
!(A \,\&\, B) & \cong & !A \otimes !B \\
!\top & \cong & I
\end{array}
$$

Cartesian Closure:

$$
\begin{array}{rcl}
\mathcal{K}_!(A \times B, C) & = & \mathcal{G}(!(A \,\&\, B), C) \\
 & \cong & \mathcal{G}(!A \otimes !B, C) \\
 & \cong & \mathcal{G}(!A, !B \multimap C) \\
 & = & \mathcal{K}_!(A, B \Rightarrow C).
\end{array}
$$

Now we have a cartesian closed category of games, we are in business to model lambda-calculus based typed theories and programming languages.

The Story From Here

Now we have a cartesian closed category of games, we are in business to model lambda-calculus based typed theories and programming languages.

Now we can begin! We roll up our sleeves and start proving full completeness, full abstraction, etc. . . .

Some References

Papers available from my webpages

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

- S. Abramsky and R. Jagadeesan, "Games and Full Completeness for Multiplicative Linear Logic", *Journal of Symbolic Logic*, (1994), vol. 59, no. 2, 543–574.
- S. Abramsky, Semantics of Interaction: an introduction to Game Semantics, in Proceedings of the 1996 CLiCS Summer School, Isaac Newton Institute, P. Dybjer and A. Pitts, eds. (Cambridge University Press) 1997, 1–31.
- S. Abramsky and G. McCusker, Game Semantics, in Computational Logic: Proceedings of the 1997 Marktoberdorf Summer School, H. Schwichtenberg and U. Berger, eds. (Springer-Verlag) 1999, 1–56.
- S. Abramsky, Algorithmic Game Semantics: A Tutorial Introduction, in Proof and System Reliability, H. Schichtenberg and R. Steinbruggen, eds., Proceedings of the NATO Advanced Study Institute, Marktoberdorf, Kluwer Academic Publishers, 2001, 21–47.
- S. Abramsky, Information, Processes and Games. To appear in: *Handbook of the Philosophy of Information*, ed. P. Adriaans and J. van Benthem, Elsevier, 2008.