

# Games in Logic, Semantics, and Verification

Julian Gutierrez

University of Cambridge Computer Laboratory

Event Structures, Stable Families and Games  
Aarhus University  
Aarhus, September, 2011

## Introduction

### Games with winning conditions

Equivalence-checking games

Model-checking games

### Games without winning conditions

Equivalence-checking games

Model-checking games

## Extensions

# Games in mathematical logic, semantics, and verification

Games are formal representations of the *interactions* or flow of information between various, possibly *independent*, agents.

Mathematical *logic games* (LG) were defined in the 1950's by Lorenzen, Ehrenfeucht, Fraïssé, and Hintikka.

Van Benthem: “Any logical task can be “gamified,” by pulling it apart into roles for two players whose dynamic interaction tests the notion involved.”

Some important games:

- ▶ Model comparison
- ▶ Model evaluation
- ▶ Model construction

## Examples

- ▶ Model comparison (Ehrenfeucht–Fraïssé):  
Input:  $(M, T, \sim)$   
Output: Yes iff  $M \sim T$  ; No iff  $M \not\sim T$
- ▶ Model evaluation (Hintikka):  
Input:  $(M, \phi, \models)$   
Output: Yes iff  $M \models \phi$  ; No iff  $M \not\models \phi$
- ▶ Model construction (Lorenzen, Hintikka):  
Input: a specification, e.g., a formula, a program description, etc.  
Output: a model, if any.

All these very different-looking games share something in common: their solutions can be extracted from a particular strategy of one of the players.

## Games in computer science

In computer science, games are used, mainly, for *modelling* and *verification*.

The most common case is that of two-player games:

- ▶ Games for modelling:  
played between a 'system' and an 'environment'.
- ▶ Games for verification:  
played between a 'verifier' and a 'falsifier'.

In both cases one can think that there is a Positive (the system or verifier) and a Negative (the environment or falsifier) player. They have many different names in the literature, for instance: Eve/Adam, Existential/Universal, Proponent/Opponent, Eloise/Abelard, Player/Opponent, and many more! Here we will call them P and O, respectively.

## Two-player games

- ▶ Players: P and O.
- ▶ Arena: a structure/board that contains moves and positions.
- ▶ Strategies: maps or functions that determine how to play.
- ▶ Rules: determine what is valid or invalid within the game (axioms).
- ▶ Winning conditions: determine the winner of a play, if any.

Usually, one is interested only in some special strategies, for instance:

- ▶ Strategies which ensure that the player who uses it can win any play.
- ▶ Strategies that can be composed with any counter-strategy.

## Games with winning conditions

- ▶ Model comparison (bisimulation checking):  $M \sim_b T$
- ▶ Model evaluation (model checking):  $M \models \phi$

# Bisimulation checking

Bisimulation checking:  $M \sim_b T$

## Models: graphical notation

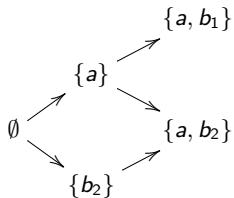
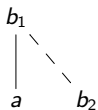
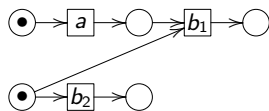


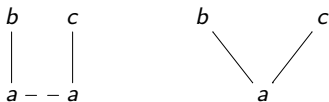
Figure: A Petri net, an event structure, and a stable family

# Bisimulation games

$$M \sim_b T$$

- ▶ Players: P and O.
- ▶ Arena:  $A \subseteq M \times T$ .
- ▶ Strategies: functions in  $A$ .
- ▶ Rules: which determine what is valid or invalid within the game (axioms):
  - ▶ O always plays first.
  - ▶ O and P alternate.
  - ▶ O must choose an event  $e$  in either  $M$  or  $T$ .
  - ▶ P must choose an event  $e'$  with the same label in the opposite model.
- ▶ Winning conditions:
  - ▶ If P cannot choose an event  $e'$ , then O wins.
  - ▶ Otherwise, P wins.

## Example



Two event structures that are not bisimilar



Two bisimilar event structures

## Properties of a bisimulation game

- ▶ Soundness: if  $M \not\sim_b T$ , then O has a winning strategy.
- ▶ Completeness: if  $M \sim_b T$ , then P has a winning strategy.
- ▶ Determinacy: for all plays either P or O has a winning strategy.
- ▶ Positional: winning strategies are history-free/memoryless.
- ▶ Decidability: solvable in PTIME.

# Model checking

Model checking:  $M \models \phi$

# Hennessy–Milner logic

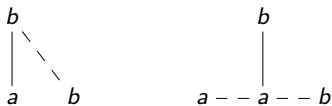
Hennessy–Milner logic (HML) has formulae  $\phi$  built from a set  $\Sigma$  of labels  $a, b, \dots$  by the following grammar:

$\phi$	$::=$	$tt \mid ff \mid$	Boolean constants
		$\phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid$	Boolean operators
		$\langle a \rangle \phi_1 \mid [a] \phi_1$	Modal operators

## HML semantics by example



$$\phi = \langle a \rangle [b] \text{tt}$$



$$\phi = [a] \langle b \rangle \text{tt}$$

# Model checking games

$$M \models \phi$$

- ▶ Players: P and O.
- ▶ Arena:  $A \subseteq M \times \text{Sub}(\phi)$ .
- ▶ Strategies: functions in  $A$ .
- ▶ Rules: which determine what is valid or invalid within the game (axioms):
  - ▶ P must choose a subformula when  $\vee$  and an event when  $\langle a \rangle$
  - ▶ O must choose a subformula when  $\wedge$  and an event when  $[a]$
- ▶ Winning conditions:
  - ▶ If  $(x, \text{tt})$  is played or O cannot make a move, then P wins.
  - ▶ If  $(x, \text{ff})$  is played or P cannot make a move, then O wins.

## Examples

$$\begin{array}{c} b \quad c \\ | \quad | \\ a \text{ --- } a \end{array} \not\equiv [a](\langle b \rangle \text{tt} \wedge \langle c \rangle \text{tt})$$

$$\begin{array}{c} b \text{ --- } c \\ \diagdown \quad \diagup \\ a \end{array} \models [a](\langle b \rangle \text{tt} \wedge \langle c \rangle \text{tt})$$

## Properties of a HML model-checking game

- ▶ Soundness: if  $M \not\models \phi$ , then O has a winning strategy.
- ▶ Completeness: if  $M \models \phi$ , then P has a winning strategy.
- ▶ Determinacy: for all plays either P or O has a winning strategy.
- ▶ Positional: the winning strategies are history-free/memoryless.
- ▶ Decidability: solvable in PTIME.

## Games without winning conditions

One important example in computer science is that of game semantics.

Game semantics is used to provide models for programs. Because of this, they can be regarded as model construction games where the 'specification' is a program. Unlike logical specifications, every program can have a model.

As a consequence, in game semantics the focus is on having games that not just construct a model, but instead, that provide 'fully abstract' models.

A game semantics is fully abstract whenever

$$M_1 \sim_{obs} M_2 \text{ iff } [M_1] = [M_2]$$

## Model construction via game semantics

**Input:** a program specification  $M$ .

**Output:** a model  $[M]$  of  $M$ , which is obtained from the strategy for P.

- ▶ Players: P and O.
- ▶ Arena:  $A$ , which is constructed from  $M$ .
- ▶ Strategies: maps between games.
- ▶ Rules: depend on the kind of semantic game, e.g.,:
  - ▶ Alternating, with O playing first.
  - ▶ Innocent.
  - ▶ Well-bracketed.
  - ▶ and so on...
- ▶ Winning conditions: none!

## Game semantics (by example)

In game semantics a program (i.e., a strategy for P) is represented by the set of plays it admits. In some cases this set may be regular or context-free.

**Example:** function application for the following lambda term:

$$(\lambda x.x + 1) 3$$

Player	$((A$	$\multimap$	$B)$	$\otimes$	$A)$	$\multimap$	$B)$
$O$							$r_o$
$P$			$r_o$				
$O$	$r_i$						
$P$					$r_i$		
$O$					3		
$P$	3						
$O$			4				
$P$							4

## Equivalence-checking games

Observational equivalence checking:  $M_1 \sim_{obs} M_2$

The algorithmic game semantics approach:

- ▶ If the strategies  $\sigma_1$  and  $\sigma_2$  for P, which represent the behaviour of  $M_1$  and  $M_2$ , are regular languages or context-free grammars, then one can represent them as FSAs or DPDAs  $A_{M_1}$  and  $A_{M_2}$ , respectively.
- ▶ Then, check for language equivalence, i.e.,  $\mathcal{L}(A_{M_1}) = \mathcal{L}(A_{M_2})$ .

$$M_1 \sim_{obs} M_2 \text{ iff } \mathcal{L}(A_{M_1}) = \mathcal{L}(A_{M_2})$$

## Model-checking games

Model checking:  $M \models \phi$

The algorithmic game semantics approach:

- ▶ If the strategy for P is a regular language  $M$ , then one can represent it as a FSA  $A_M$ .
- ▶ Take the property/formula  $\phi$  to be verified and represent its complement  $\bar{\phi}$  as a FSA  $A_{\bar{\phi}}$ .
- ▶ Check for emptiness of the product automaton, i.e., that  $\mathcal{L}(A_M \times A_{\bar{\phi}}) = \emptyset$ .

$$M \models \phi \text{ iff } \mathcal{L}(A_M \times A_{\bar{\phi}}) = \emptyset$$

## Properties of a game model

- ▶ Compositional: one can construct “little models” for each program part.
- ▶ Fully abstract for various (fragments of) programming languages.
- ▶ Decidability: on fragments of some functional and imperative languages.
- ▶ As there are neither winning conditions nor winning strategies, can we talk about soundness, completeness, or determinacy in any reasonable way? Maybe...

## Games without (explicit!) winning conditions?

In some sense, the winning conditions are built-in (or implicit!) when full abstraction is the goal: P has a “winning strategy” if it can play equivalently in two observationally equivalent programs; and O has a “winning strategy” if it can make visible the differences between two observationally different programs.

Then, a question arises: Is full abstraction a form of determinacy?

## Is full abstraction determinacy?

Recall *soundness* and *completeness* in the two first games.

Soundness in LG is like adequacy in GS: if  $M_1 \not\sim_{obs} M_2$  then

- ▶  $[M_1] \neq [M_2]$  iff
- ▶ O has a “winning strategy” to spot the difference between  $M_1$  and  $M_2$ .

Completeness in LG is like full adequacy (fullness) in GS: if  $M_1 \sim_{obs} M_2$  then

- ▶  $[M_1] = [M_2]$  iff
- ▶ P has a “winning strategy” to play equivalently in both  $M_1$  and  $M_2$ .

A GM is fully abstract iff it is both adequate and fully adequate, i.e., iff

$$M_1 \sim_{obs} M_2 \text{ iff } [M_1] = [M_2]$$

Then, under this interpretation, ‘Full abstraction’ is like *determinacy* in a LG.

## Extensions: What's next?

- ▶ Games for (true) concurrency.
- ▶ Games on infinite-state systems.
- ▶ Infinite games (e.g., for modelling fixpoint and temporal logics).
- ▶ Games as an approach to “very operational” denotational semantics.
- ▶ Games with imperfect information (e.g., probabilistic or concurrent).
- ▶ Games as a unifying framework for understanding interactions.
- ▶ Different combinations of the above.
- ▶ More...