UNIVERSITY OF OXFORD

DEPARTMENT OF COMPUTER SCIENCE

Games with bound guess actions

Thomas Colcombet

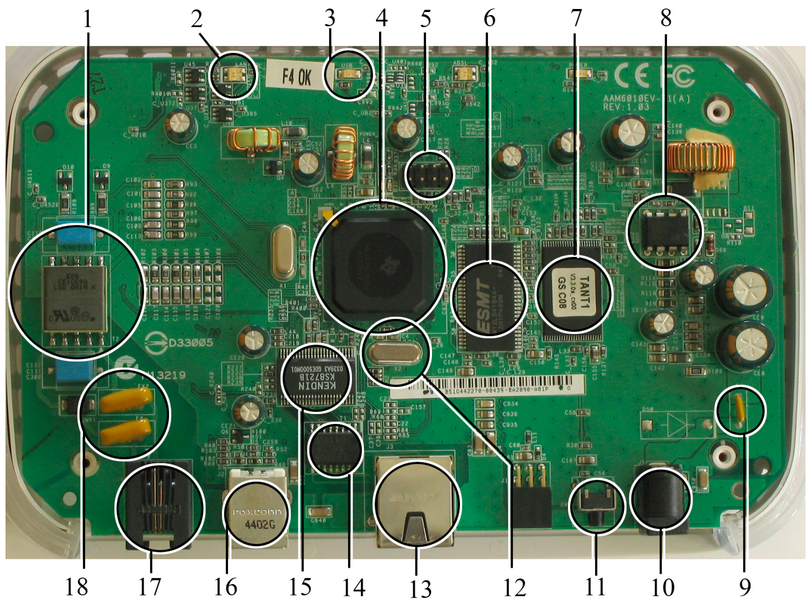
27 April 2016

joint work with
Stefan Göller
(at LICS'16)

INSTITUT DE RECHERCHE EN INFORMATIQUE FONDAMENTALE

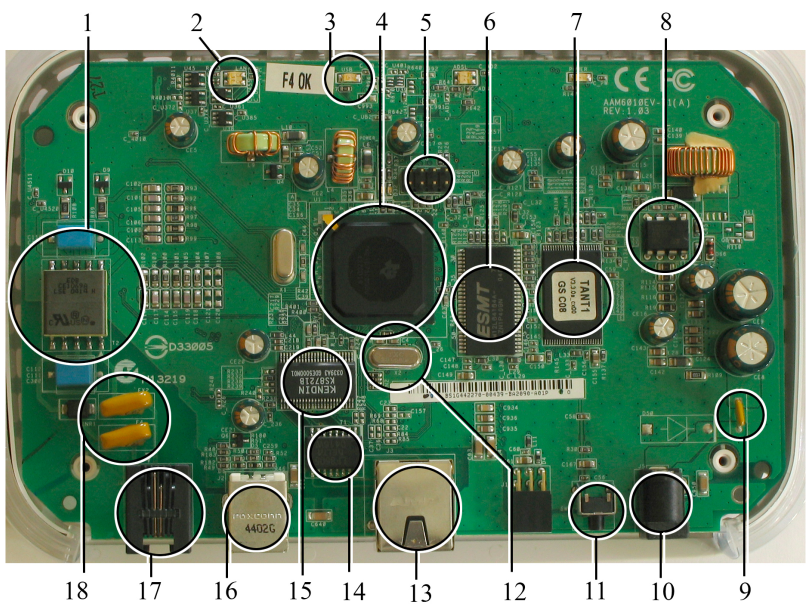
cnrs

Games for model checking



A system

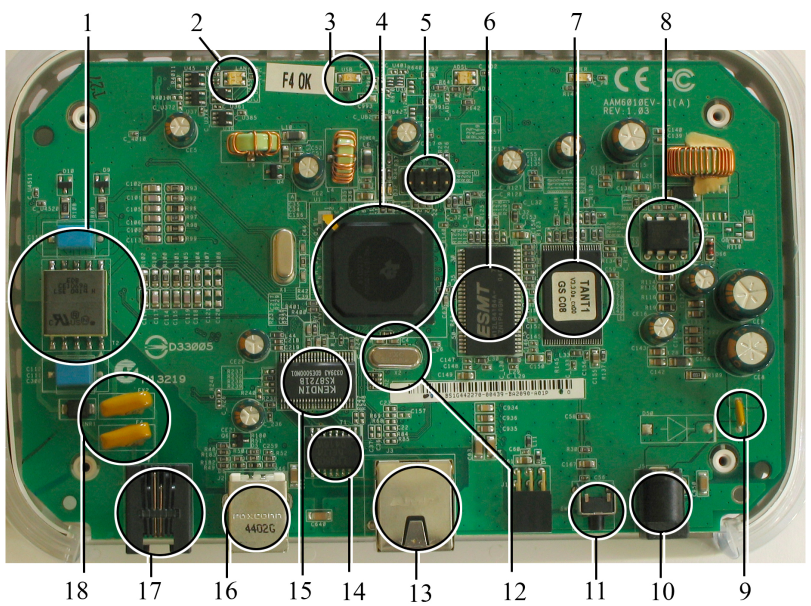
Games for model checking



A system

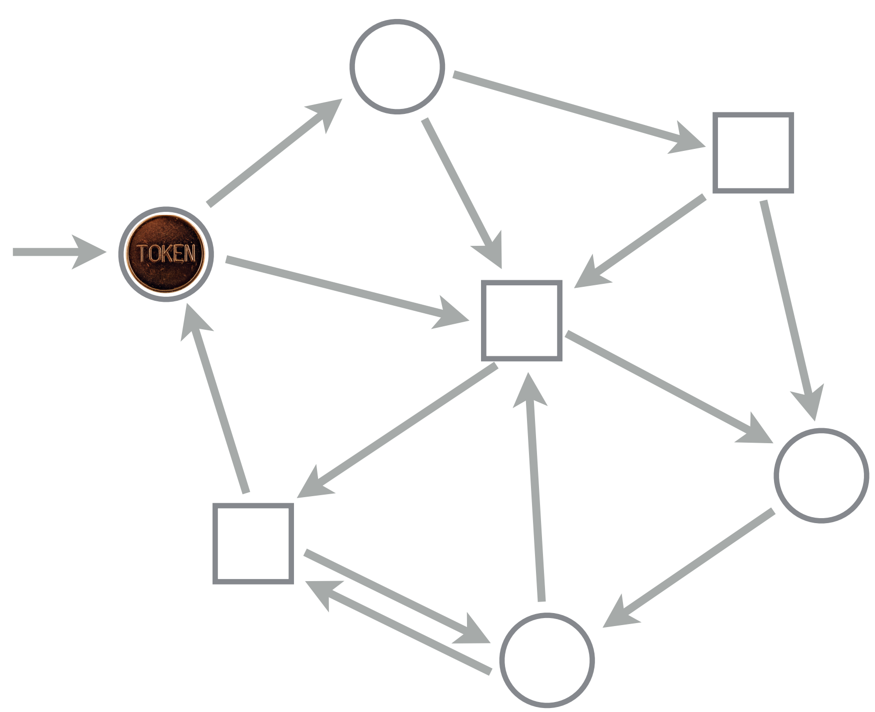
A specification that
we want to be guaranteed

Games for model checking



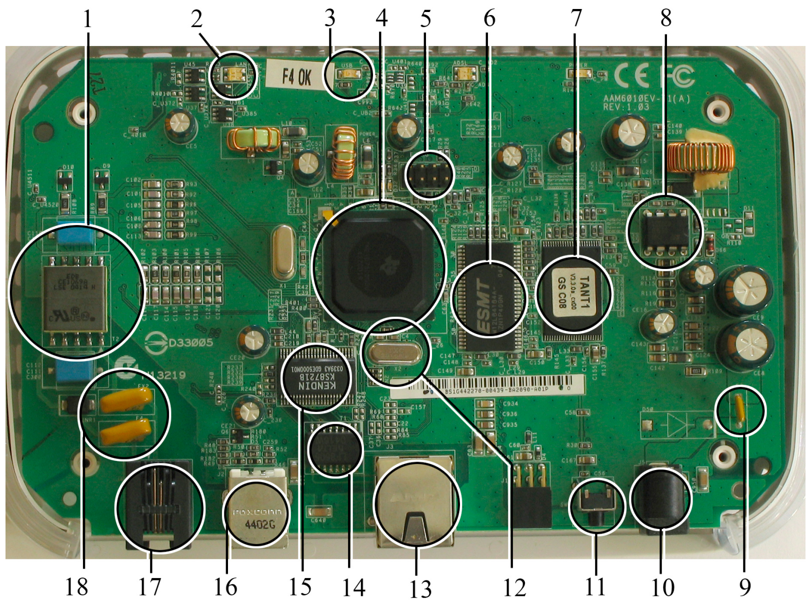
A system

A specification that
we want to be guaranteed



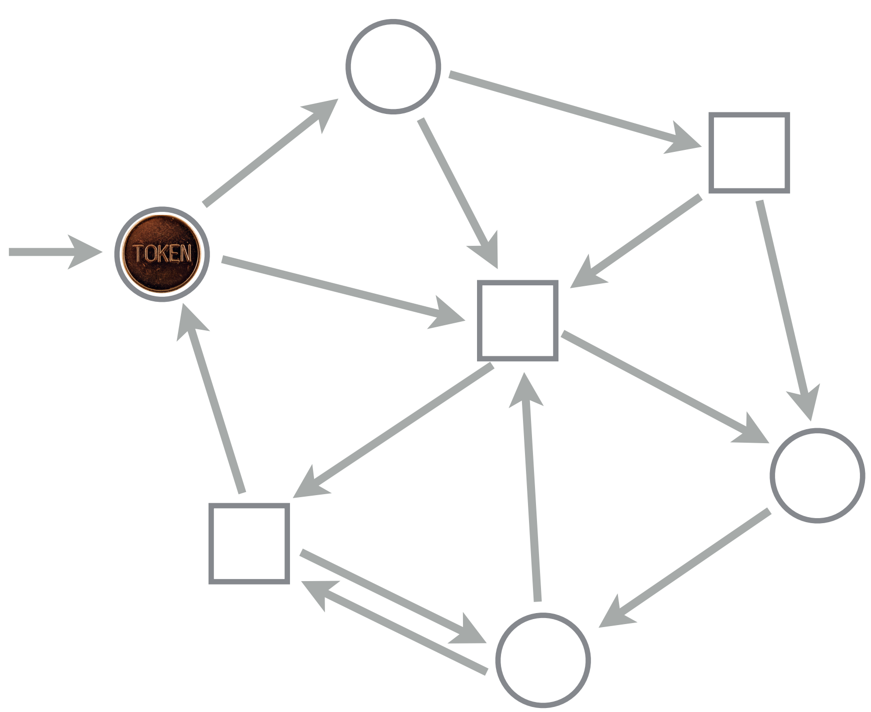
A game involving
- a prover
- a falsifier

Games for model checking



A system

A specification that
we want to be guaranteed

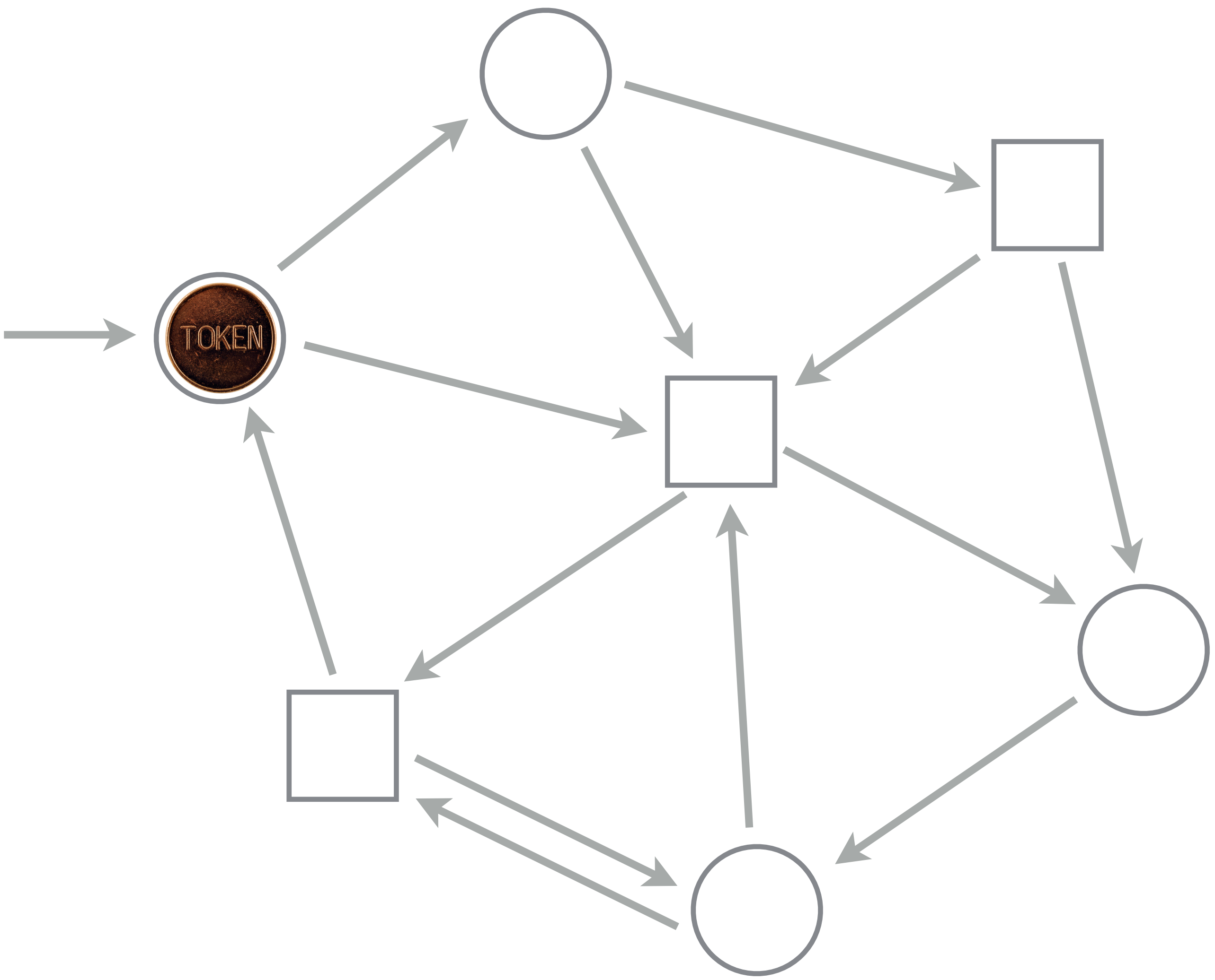


A game involving
- a prover
- a falsifier

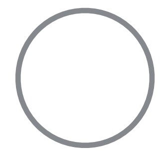
such that prover can win if and only if the system satisfies the specification.

Games

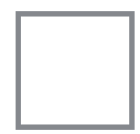
(Two players, antagonistic, turn-based)



A game is a graph in which vertices are controller either by:



the existential player
= the property prover, or

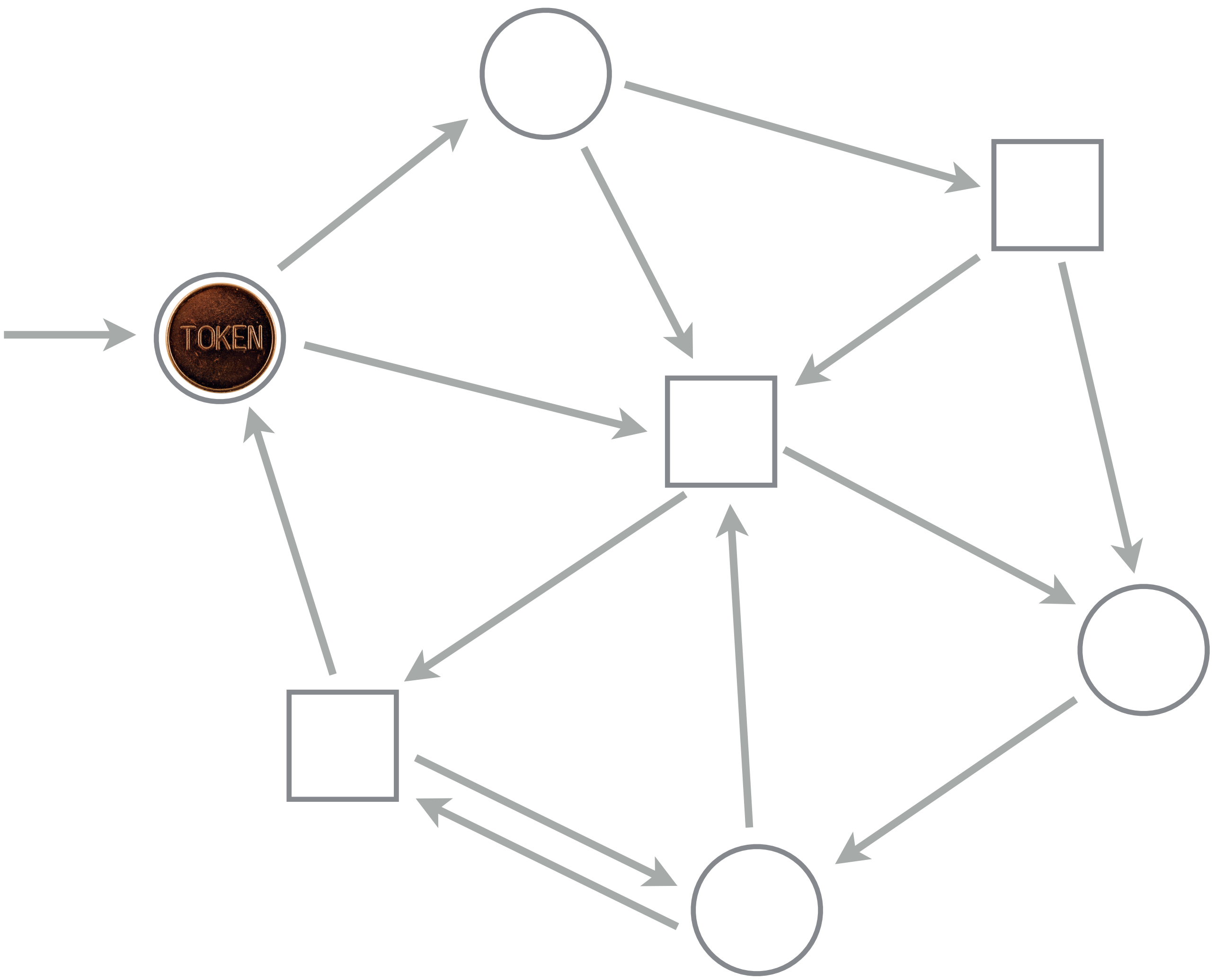


the universal player
= the environment = falsifier.

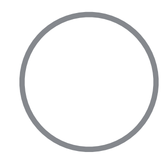
A unique token is placed, and is controlled by the owner of the vertex, choosing the transition to follow.

The winner is determined based on the infinite sequence of moves.

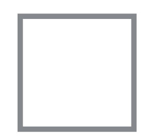
Games
(Two players, antagonistic, turn-based)



A **game** is a graph in which vertices are controller either by:



the **existential player** = the property prover, or



the **universal player** = the environment = falsifier.

A unique **token** is placed, and is controlled by the owner of the vertex, choosing the transition to follow.

The **winner** is determined based on the infinite sequence of moves.

Usually, moves are labelled by actions, and a (regular) set of winning sequences of actions is fixed.

Games with bound guess actions

Idea: players can play numbers (non-negative integers), which are promises on the evolution of some quantity.

Games with bound guess actions

Idea: players can play numbers (non-negative integers), which are promises on the evolution of some quantity.

A printer receives printing requests.

Games with bound guess actions

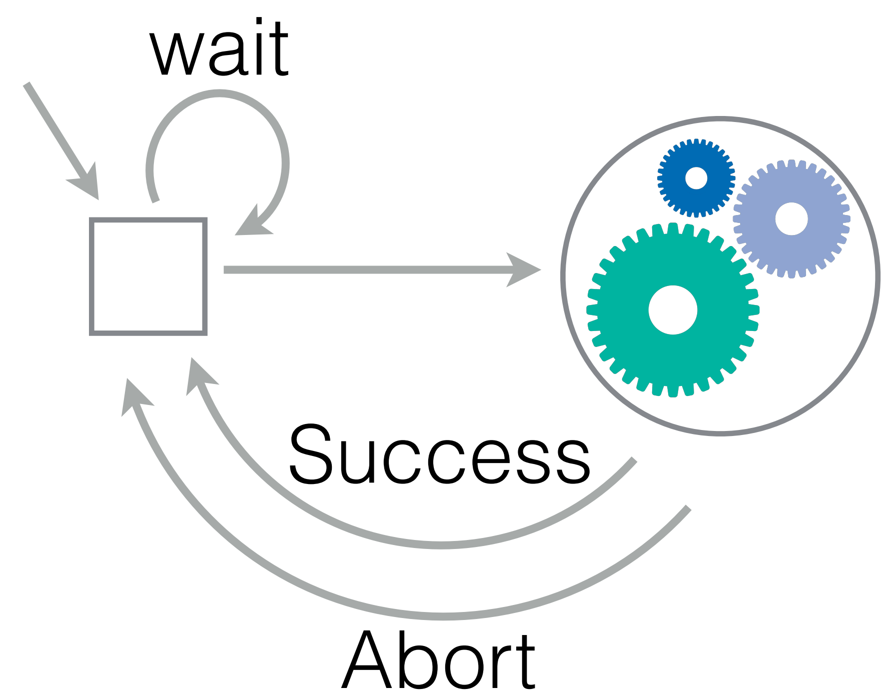
Idea: players can play numbers (non-negative integers), which are promises on the evolution of some quantity.

A printer receives printing requests.

Standard games can model specifications such as:

« every request is treated »

« system never stalls »

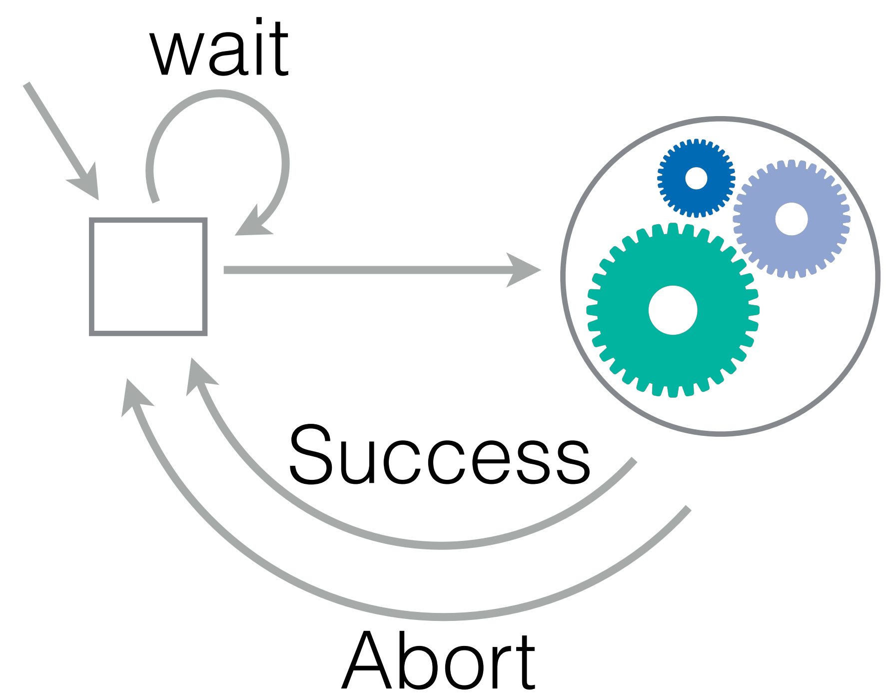


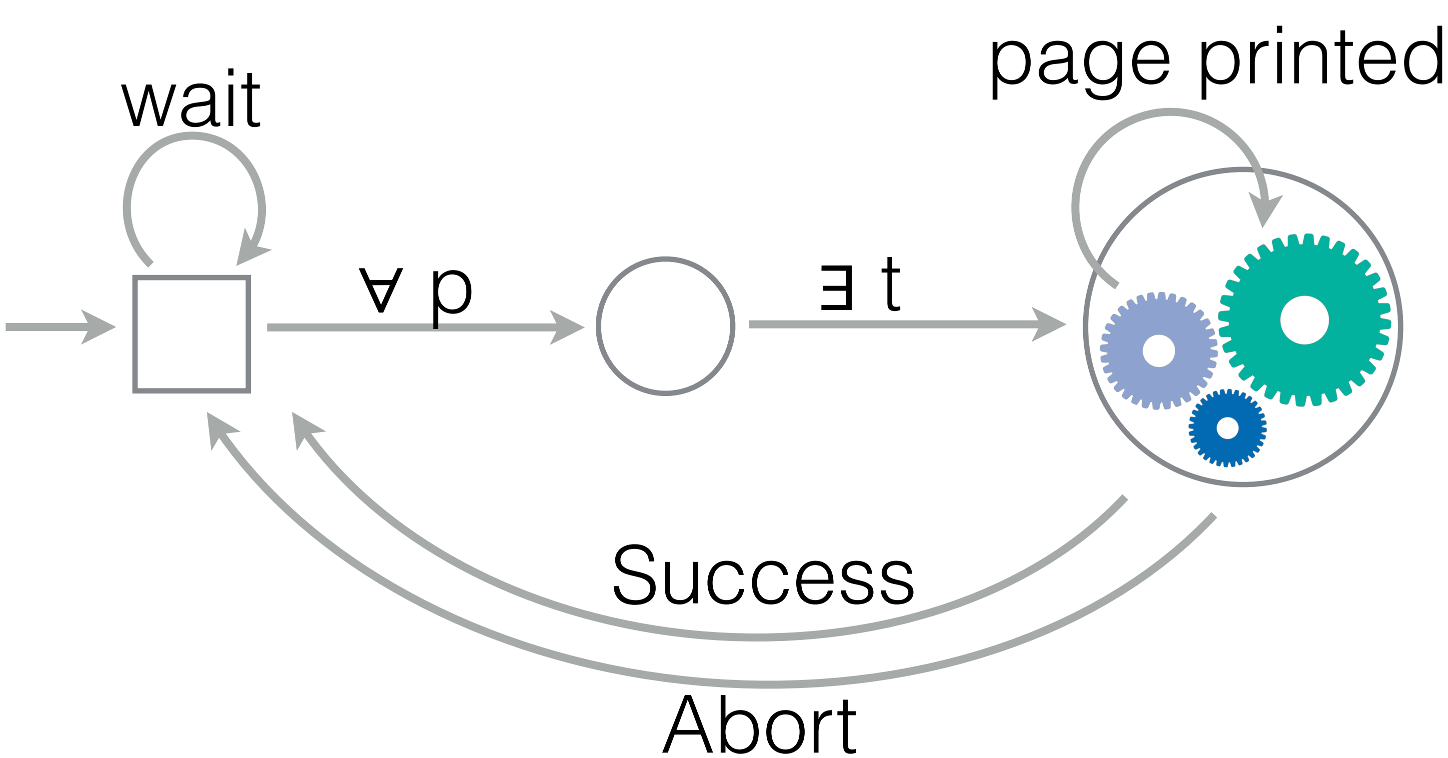
Games with bound guess actions

Idea: players can play numbers (non-negative integers), which are promises on the evolution of some quantity.

A printer receives printing requests.

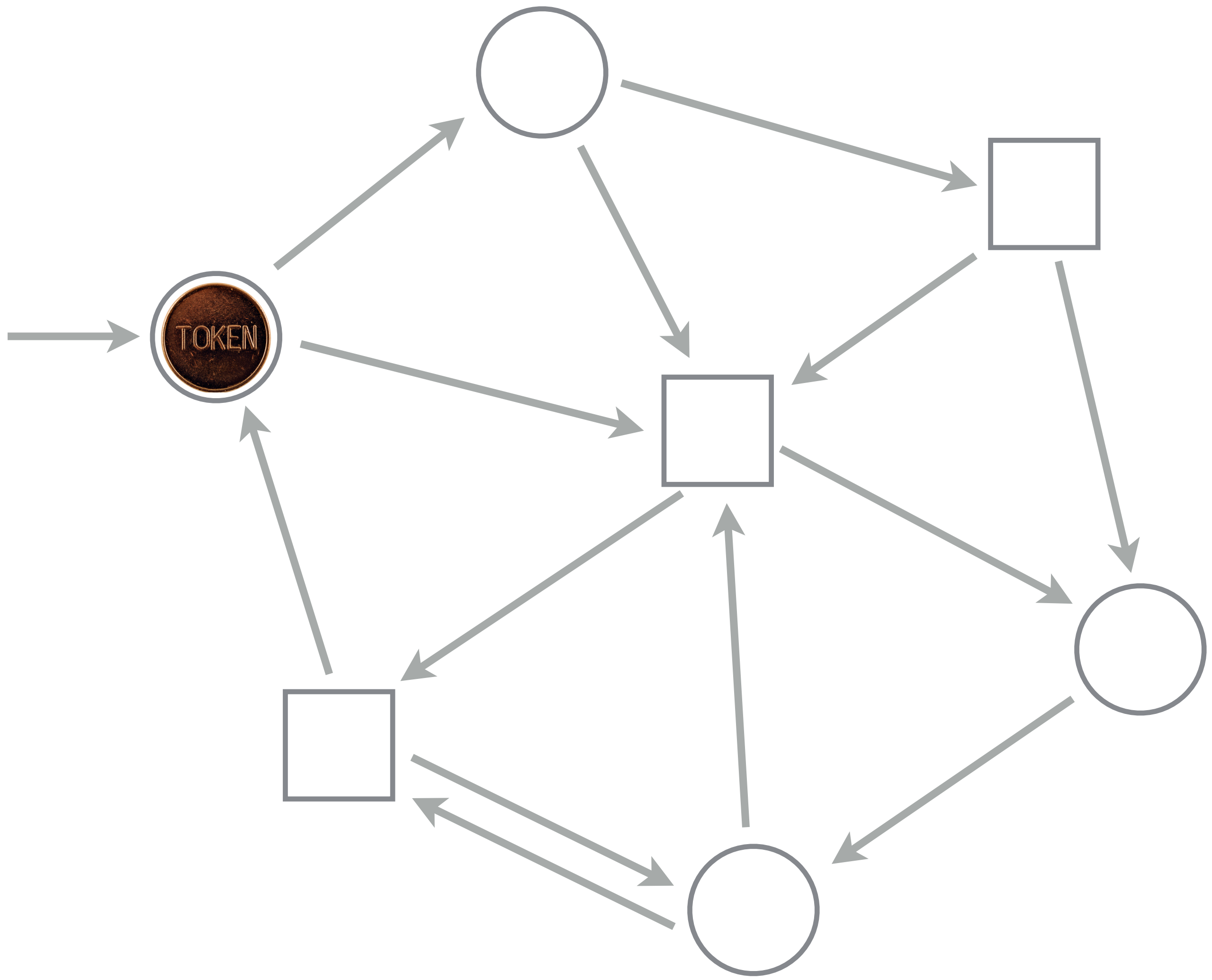
Standard games can model specifications such as:
« every request is treated »
« system never stalls »





Games with bound guess actions can model things like:
- the user declares the number p of pages to be printed,
- the printer has to guarantee to bound the printing time by t, as a function of p.

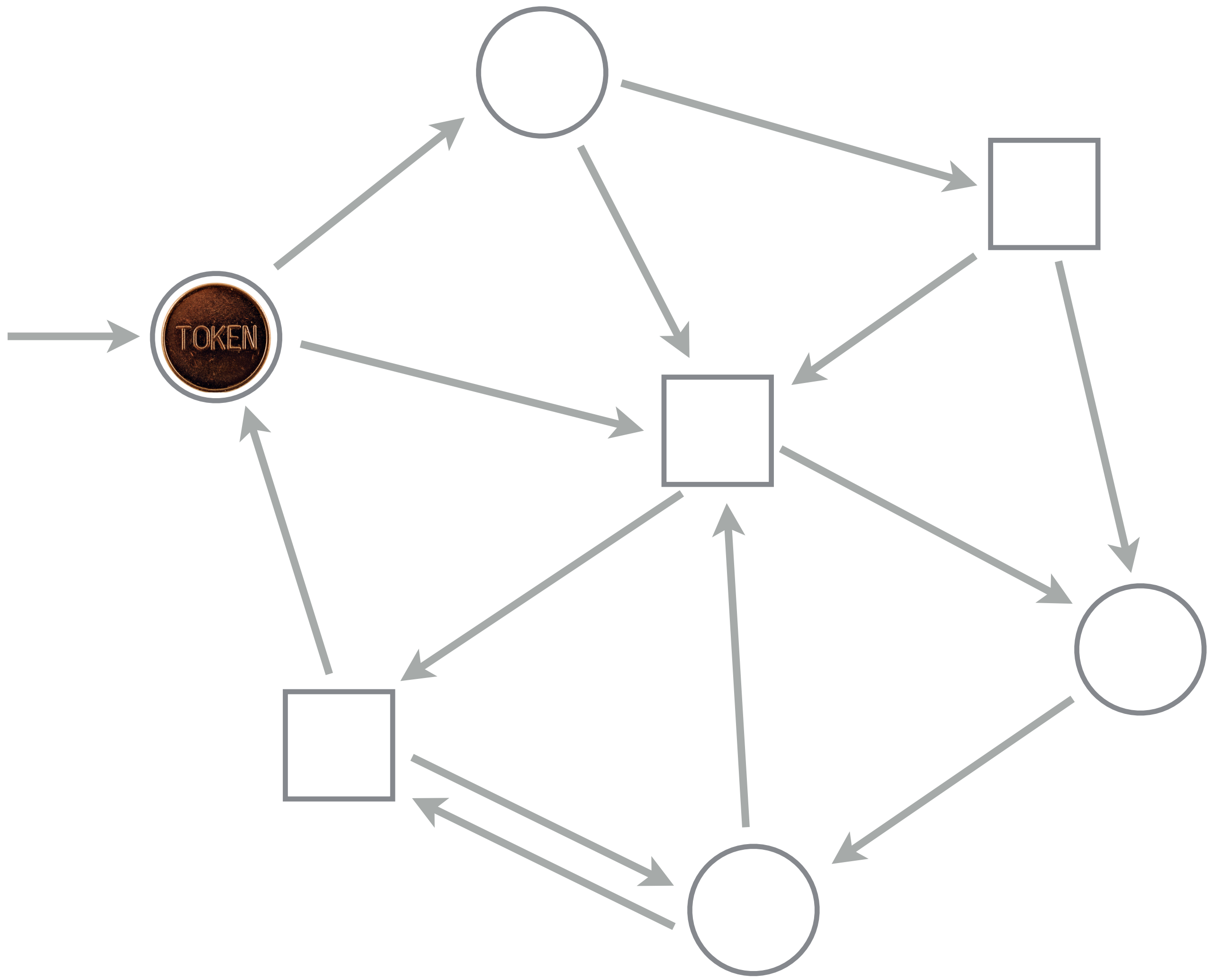
Games with bound guess actions



A game is a graph in which vertices are controller either by:

- ○ the existential player
- □ the universal player

Games with bound guess actions



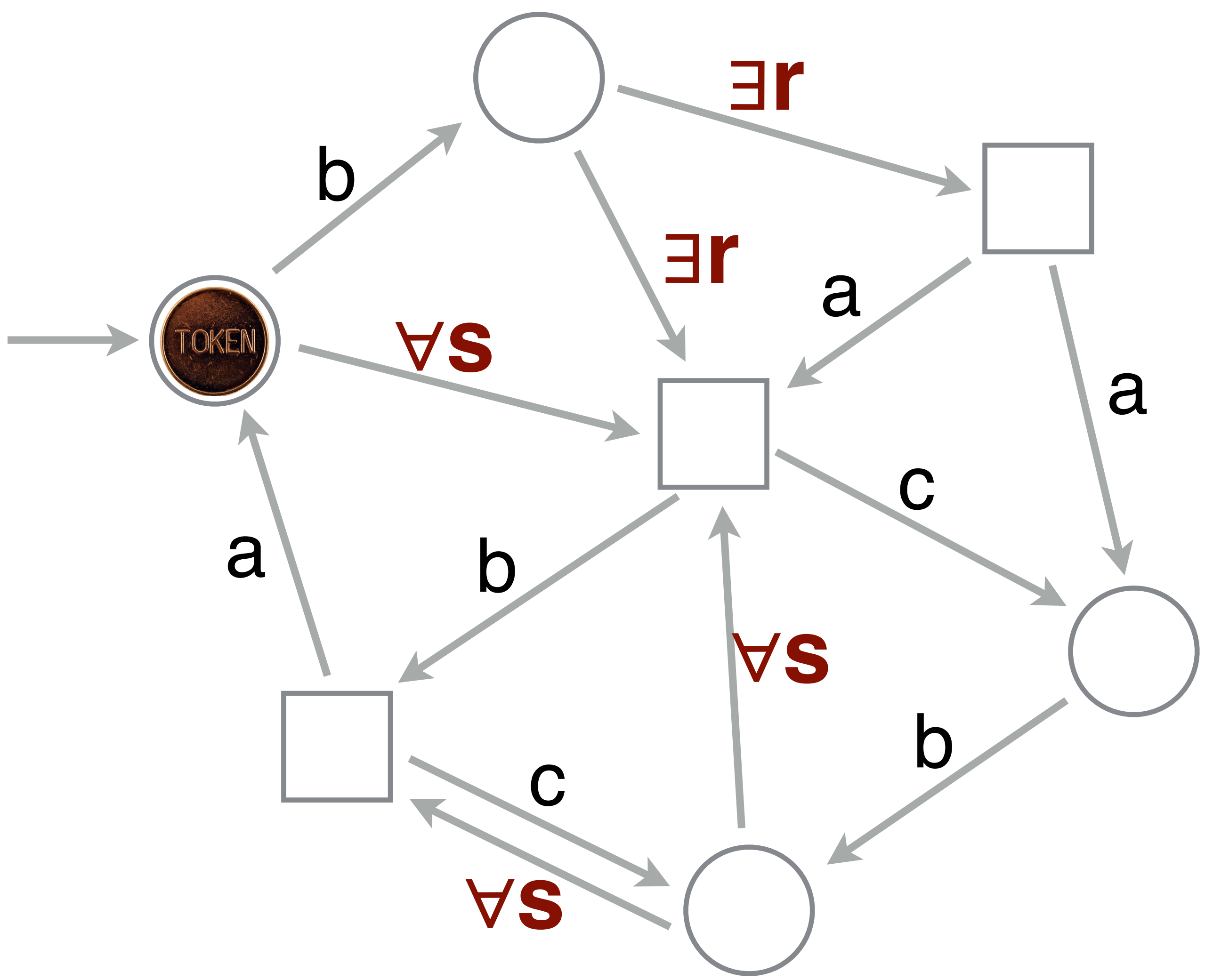
A **game** is a graph in which vertices are controller either by:

- ○ the **existential player**
- □ the **universal player**

A finite set of **registers** (r,s,t) is fixed (and are *owned* by the players ∃,∀).

Moves are labelled with normal actions or **bound guess actions ∃r , ∀s** (properly quantified).

Games with bound guess actions



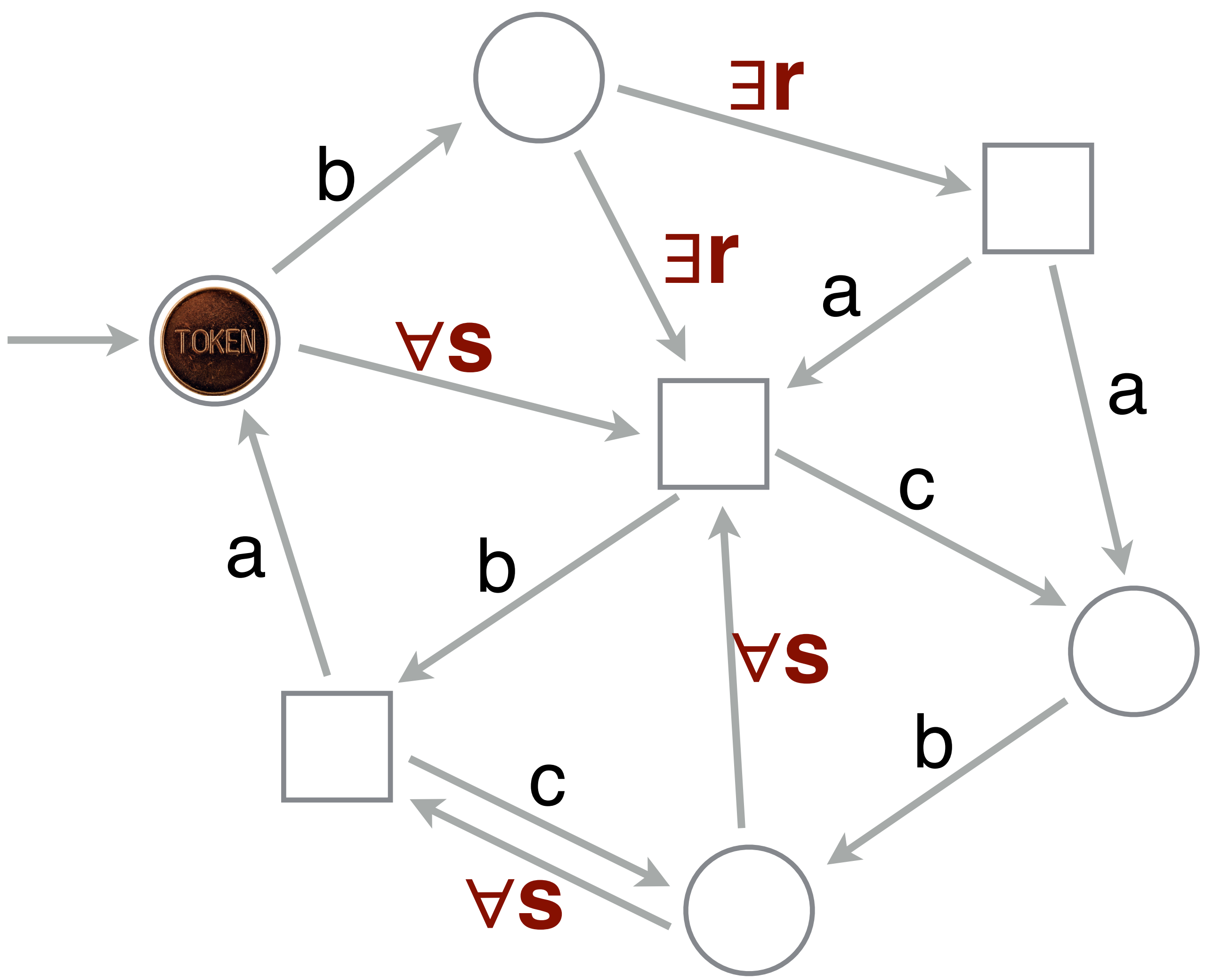
A game is a graph in which vertices are controller either by:

- ◯ the existential player
- ☐ the universal player

A finite set of registers (r,s,t) is fixed (and are owned by the players ∃,∀).

Moves are labelled with normal actions or bound guess actions ∃r , ∀s (properly quantified).

Games with bound guess actions



A **game** is a graph in which vertices are controller either by:

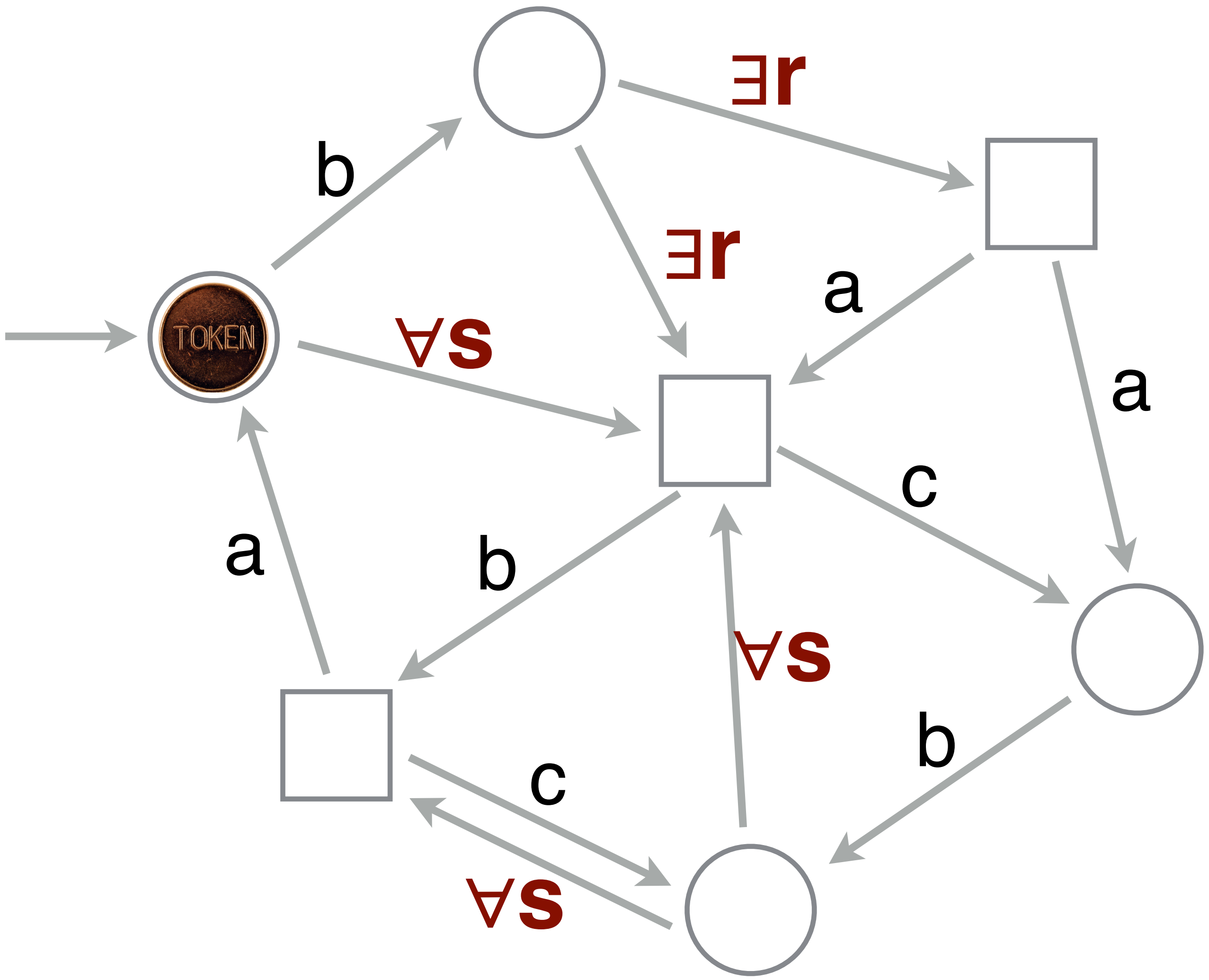
- ○ the **existential player**
- □ the **universal player**

A finite set of **registers** (r,s,t) is fixed (and are owned by the players ∃,∀).

Moves are labelled with normal actions or **bound guess actions ∃r** , **∀s** (properly quantified).

The token evolves as before, and furthermore, when bound guess actions are met, the player chooses the new **register value**.

Games with bound guess actions



A **game** is a graph in which vertices are controller either by:

- ○ the **existential player**
- □ the **universal player**

A finite set of **registers** (r,s,t) is fixed (and are **owned** by the players $\exists$,$\forall$).

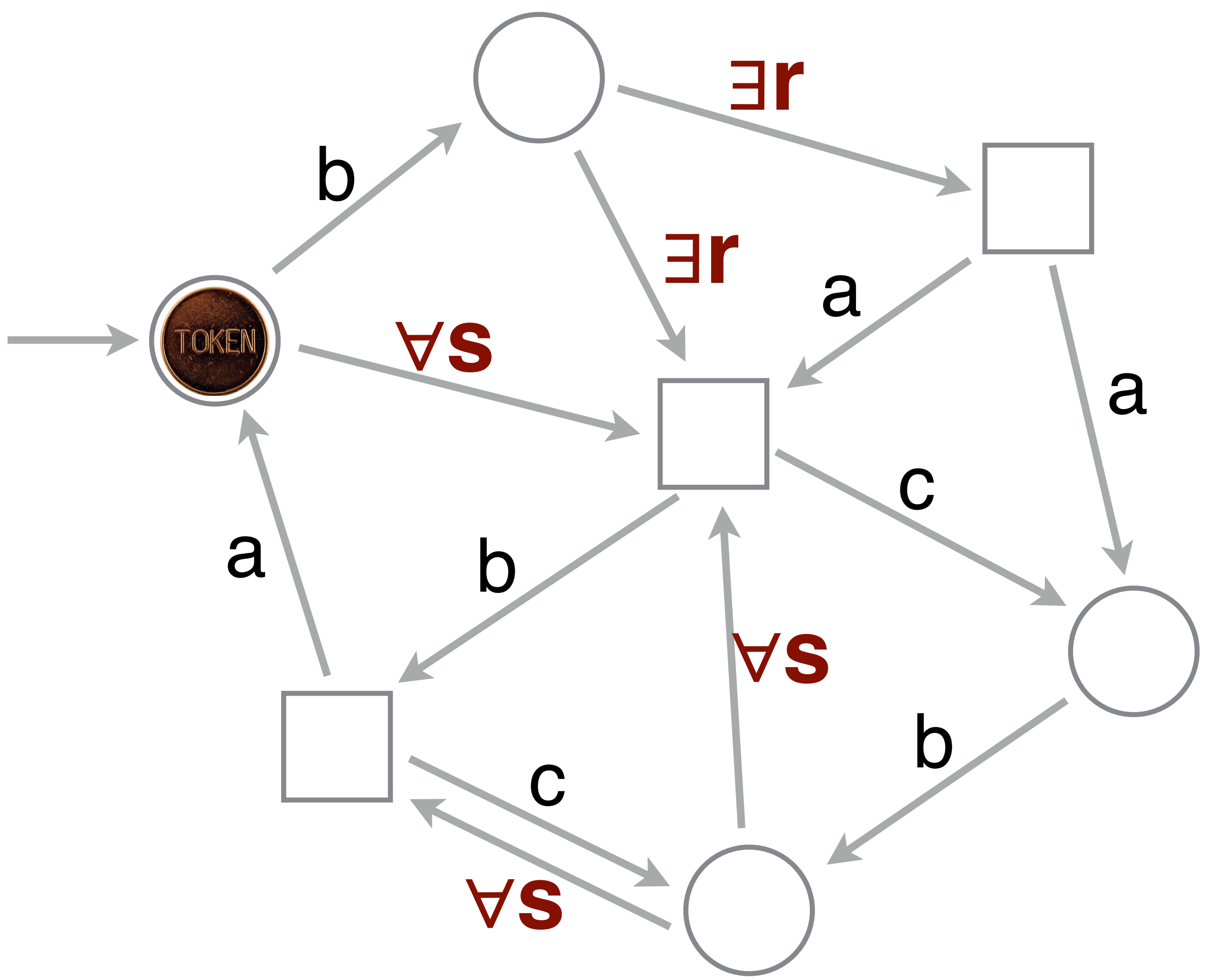
Moves are labelled with normal actions or **bound guess actions** $\exists$**r** , $\forall$**s** (properly quantified).

The token evolves as before, and furthermore, when bound guess actions are met, the player chooses the new **register value**.

The **winner** is chosen based:

- on the infinite sequence of moves, and
- how some quantities exceed the current register values or not.

Games with bound guess actions



A **game** is a graph in which vertices are controller either by:

- ○ the **existential player**
- □ the **universal player**

A finite set of **registers** (r,s,t) is fixed (and are owned by the players ∃,∀).

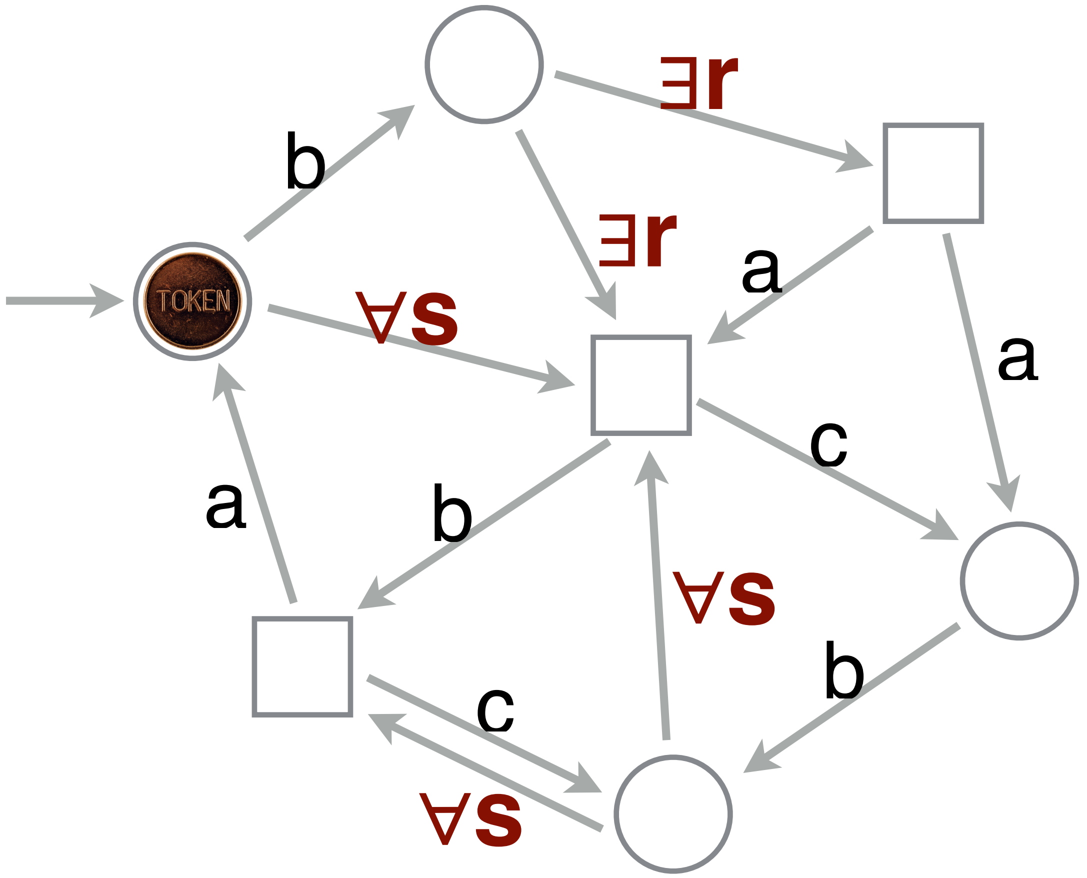
Moves are labelled with normal actions or **bound guess actions ∃r , ∀s** (properly quantified).

The token evolves as before, and furthermore, when bound guess actions are met, the player chooses the new **register value**.

The **winner** is chosen based:
- on the infinite sequence of moves, and
- how some quantities exceed the current register values or not.

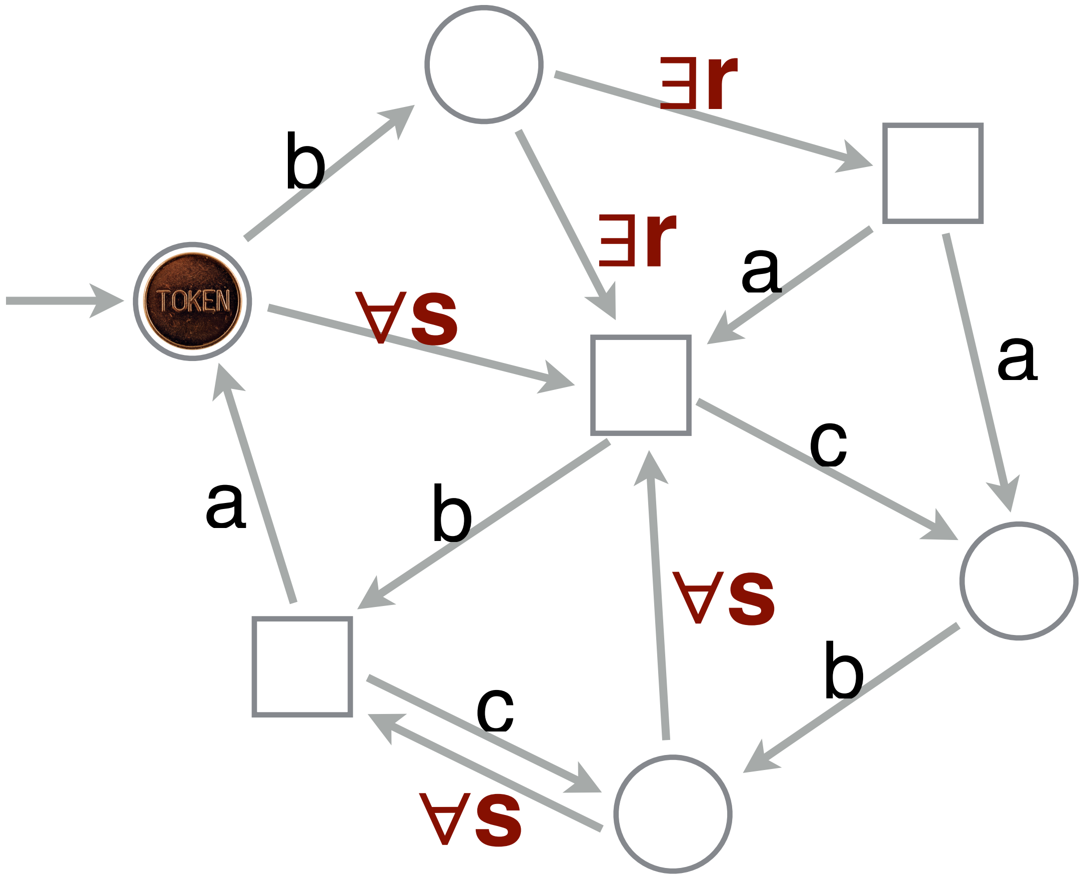
Positivity: the chooser of the value aims at respecting the promised bound.

Games with bound guess actions



- players declare values (in registers)
- these are promises on the future of some quantity
- positivity assumption: the values declared are always upper bounds.

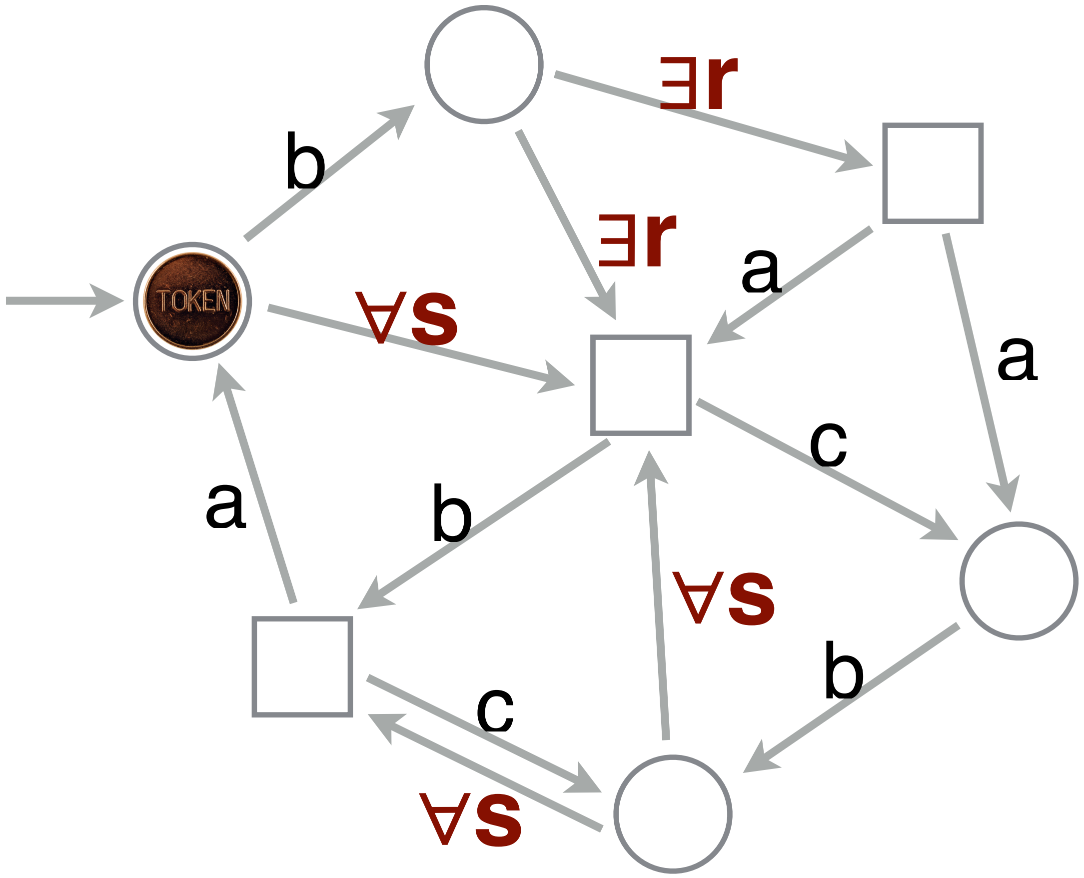
Games with bound guess actions



- players declare values (in registers)
- these are promises on the future of some quantity
- positivity assumption: the values declared are always upper bounds.

What are the quantities ?

Games with bound guess actions

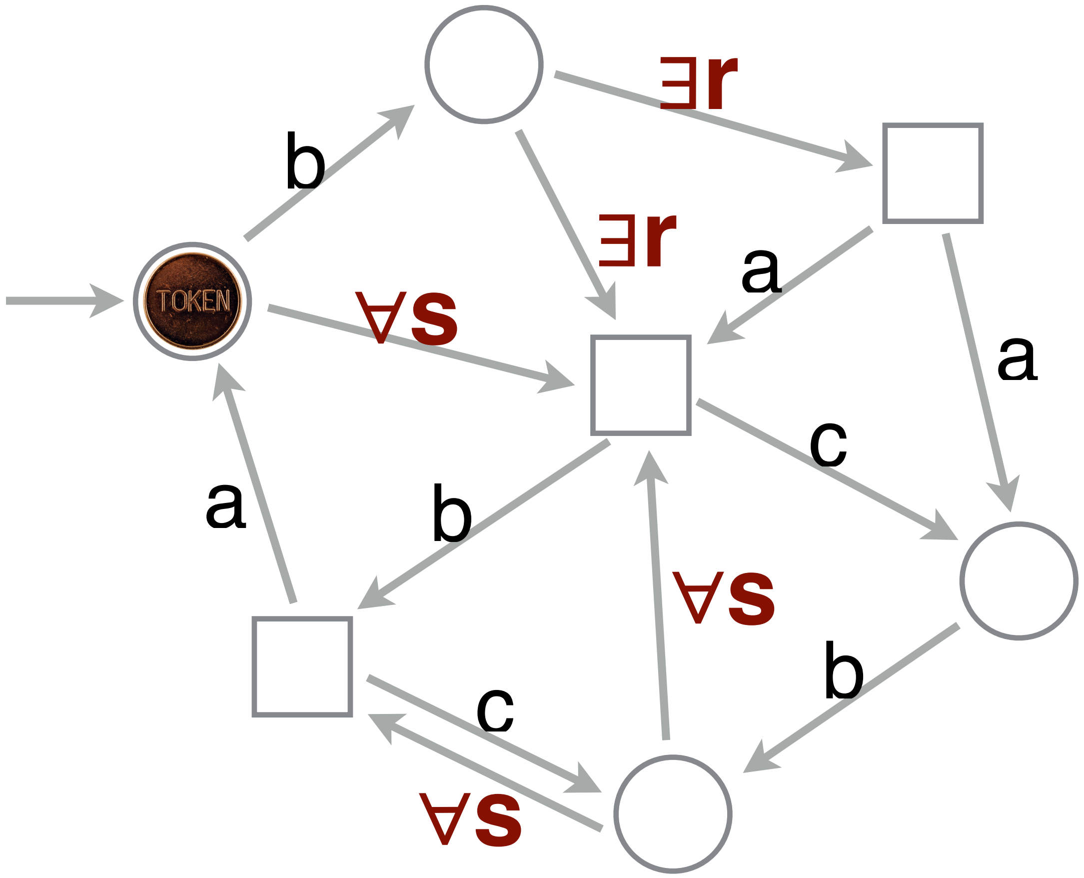


- players declare values (in registers)
- these are promises on the future of some quantity
- positivity assumption: the values declared are always upper bounds.

What are the quantities ?

What is the global condition ?

Games with bound guess actions



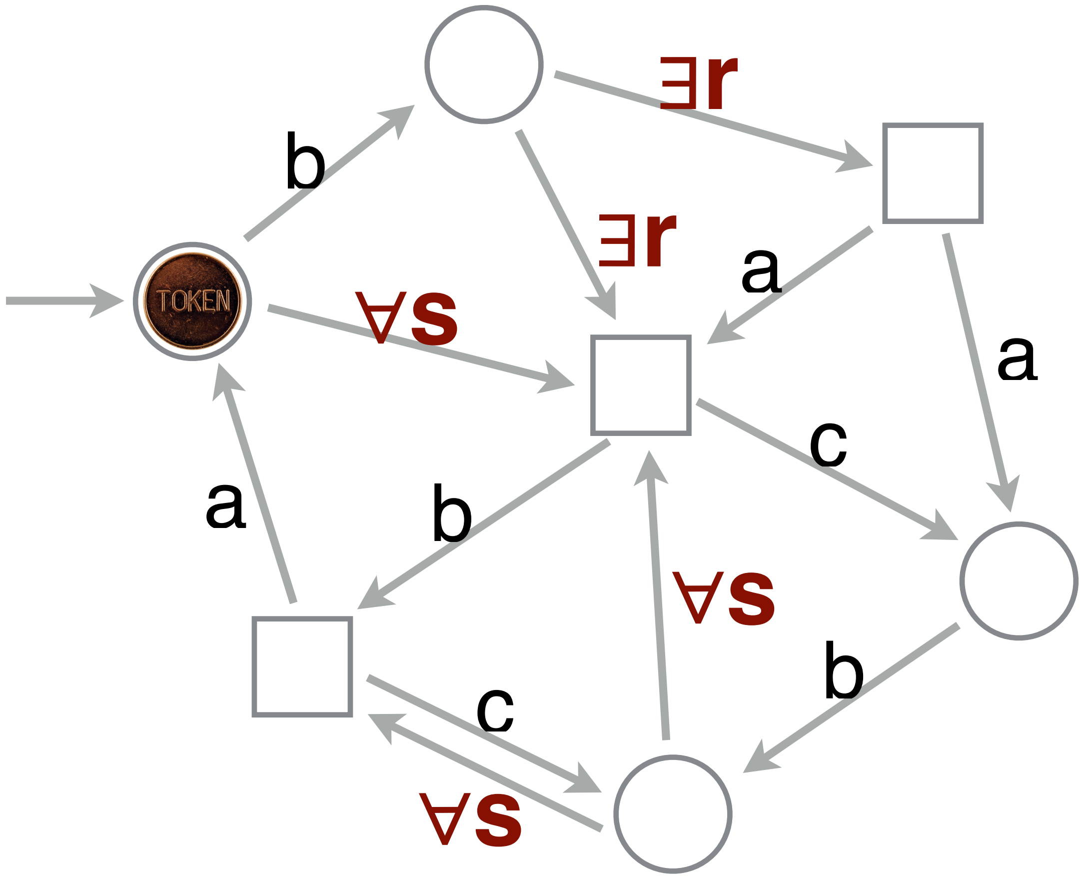
- players declare values (in registers)
- these are promises on the future of some quantity
- positivity assumption: the values declared are always upper bounds.

What are the quantities ?

What is the global condition ?

General games considered in this work

Games with bound guess actions



- players declare values (in registers)
- these are promises on the future of some quantity
- positivity assumption: the values declared are always upper bounds.

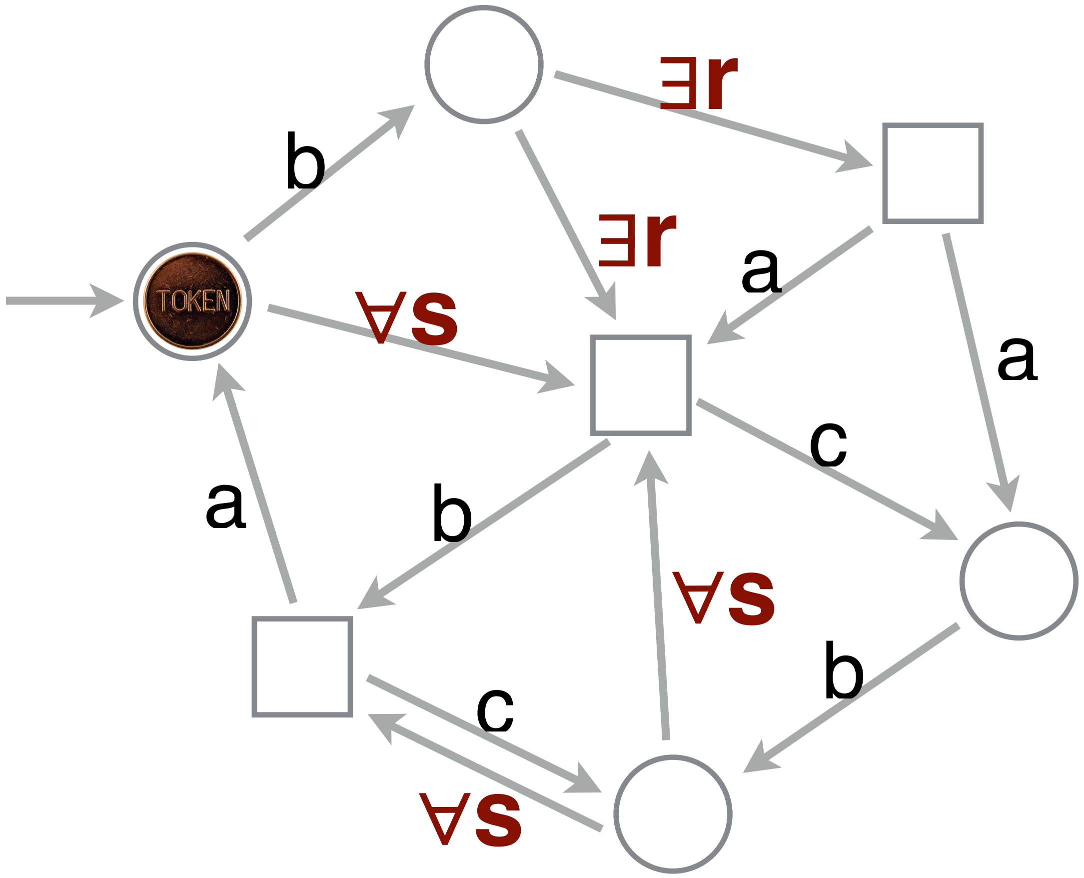
What are the quantities ?

What is the global condition ?

General games considered in this work

Quantities are regular cost functions:
(possibility to count and aggregate using min/inf and max/sup quite freely)

Games with bound guess actions



- players declare values (in registers)
- these are promises on the future of some quantity
- positivity assumption: the values declared are always upper bounds.

What are the quantities ?

What is the global condition ?

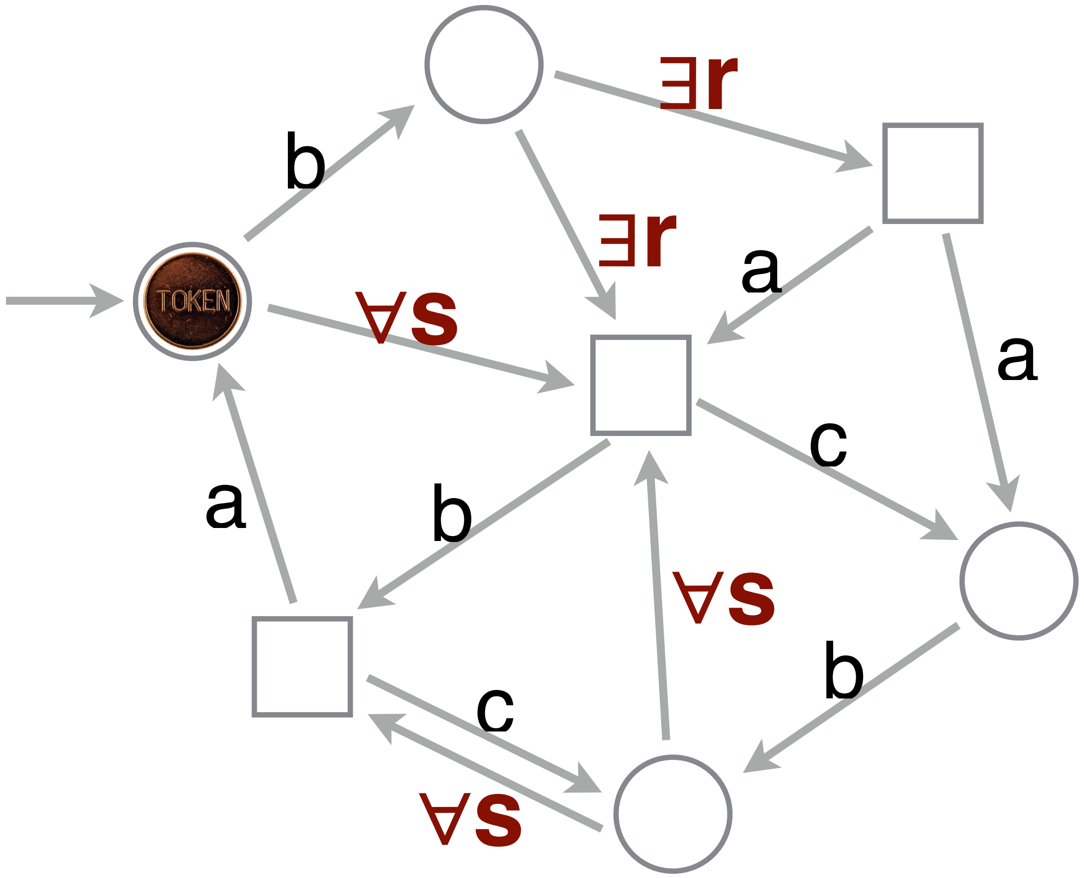
General games considered in this work

Quantities are regular cost functions:

(possibility to count and aggregate using min/inf and max/sup quite freely)

« number of pages printed since the job was last initiated »

Games with bound guess actions



- players declare values (in registers)
- these are promises on the future of some quantity
- positivity assumption: the values declared are always upper bounds.

What are the quantities ?

What is the global condition ?

General games considered in this work

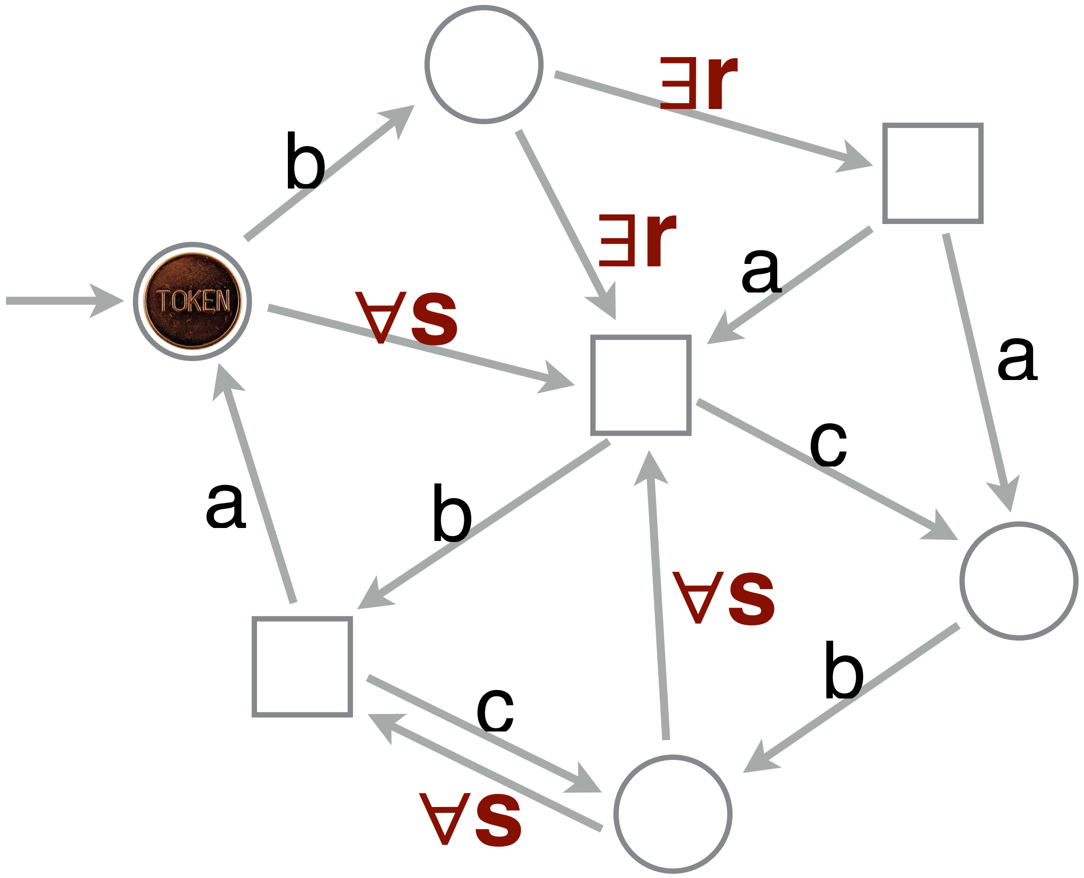
Quantities are regular cost functions:

(possibility to count and aggregate using min/inf and max/sup quite freely)

« number of pages printed since the job was last initiated »

« largest number of consecutive action a seen so far »

Games with bound guess actions



- players declare values (in registers)
- these are promises on the future of some quantity
- positivity assumption: the values declared are always upper bounds.

What are the quantities ?

What is the global condition ?

General games considered in this work

Quantities are regular cost functions:
(possibility to count and aggregate using min/inf and max/sup quite freely)
« number of pages printed since the job was last initiated »
« largest number of consecutive action a seen so far »

The global condition is a regular language of words over actions enriched with bits representing « has quantity f exceeded register r ».
This bits have to be used positively.

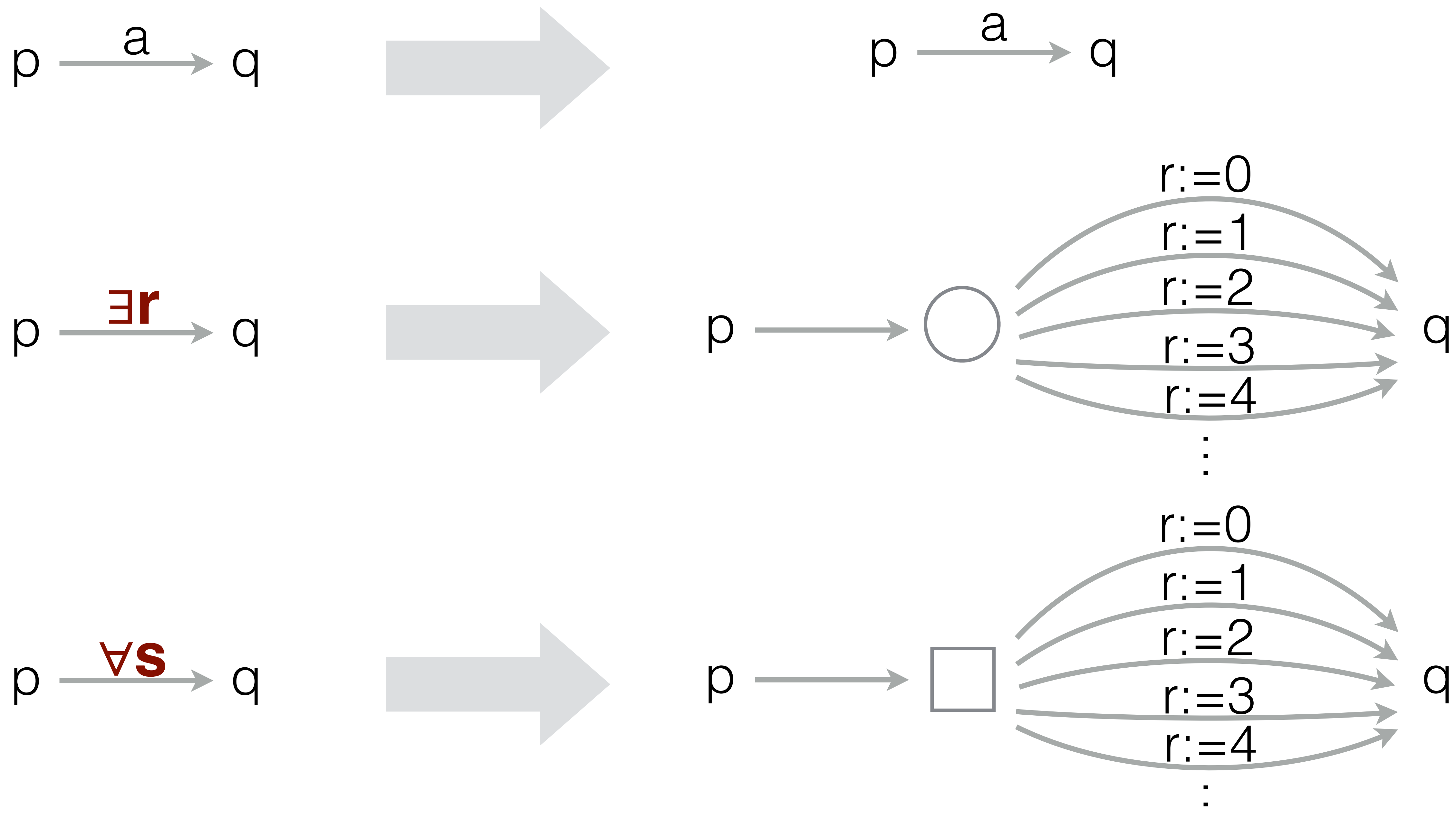
The result

Games with bound guess actions in general form:

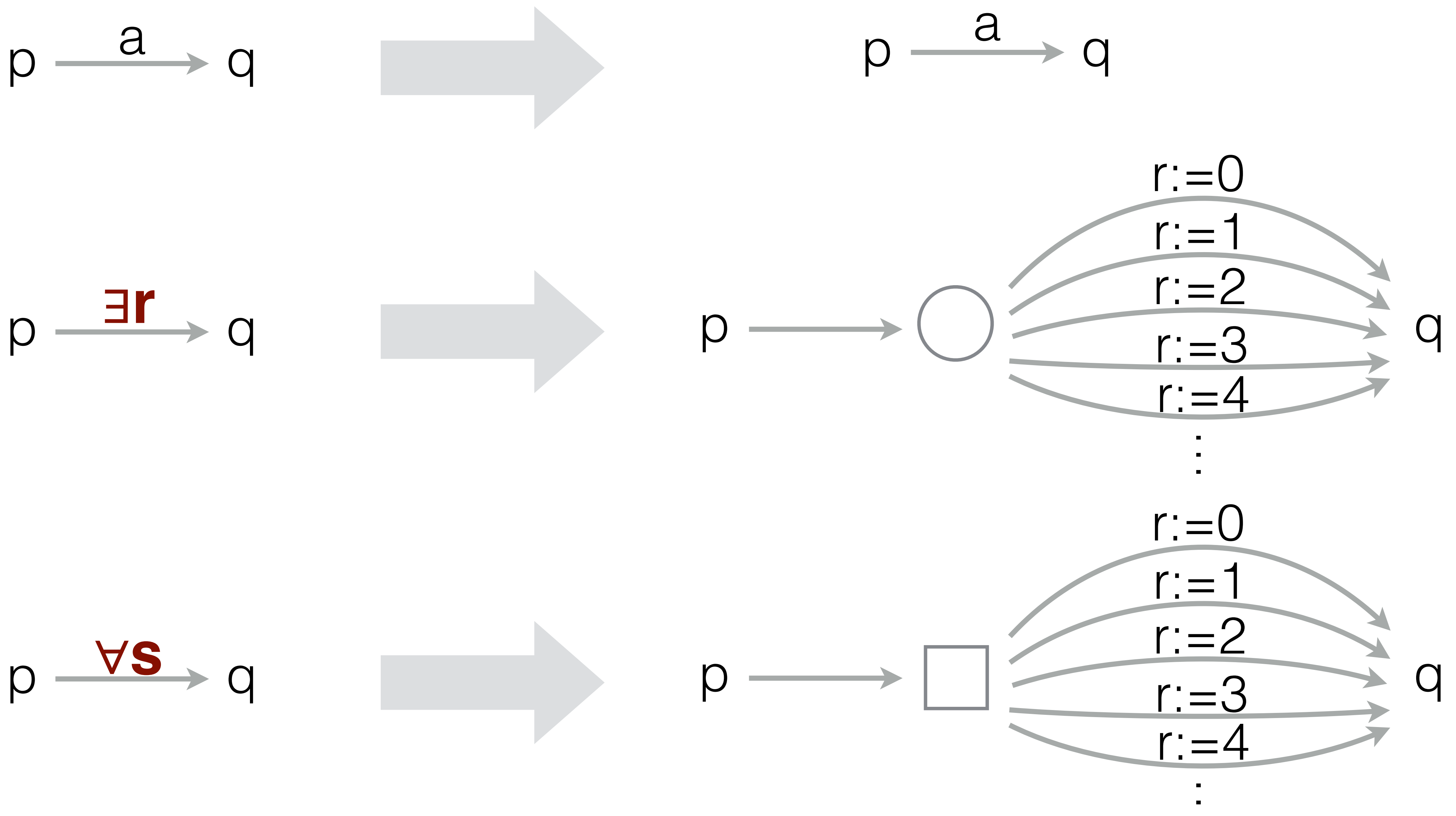
- quantities = regular cost function
- global condition = any ω-regular language (positive)

Theorem: The winner of a finite game with bound guess action in general form can be decided.

Translation into usual games



Translation into usual games



Formally, this translation is a way to describe the semantics of games with bound guess actions.

Strategies in games (wbga)

Strategies are used to define the property of being winning.

Strategies in games (wbga)

Strategies are used to define the property of being winning.

(Standard) strategies (for the existential player) are trees with vertex labelled nodes, such that

- either a vertex is owned by the existential player, and it has one child…
- or it is owned by the universal player, and it has as many children as successors in the arena …

Strategies in games (wbga)

Strategies are used to define the property of being winning.

(Standard) strategies (for the existential player) are trees with vertex labelled nodes, such that

- either a vertex is owned by the existential player, and it has one child…
- or it is owned by the universal player, and it has as many children as successors in the arena …

Now…

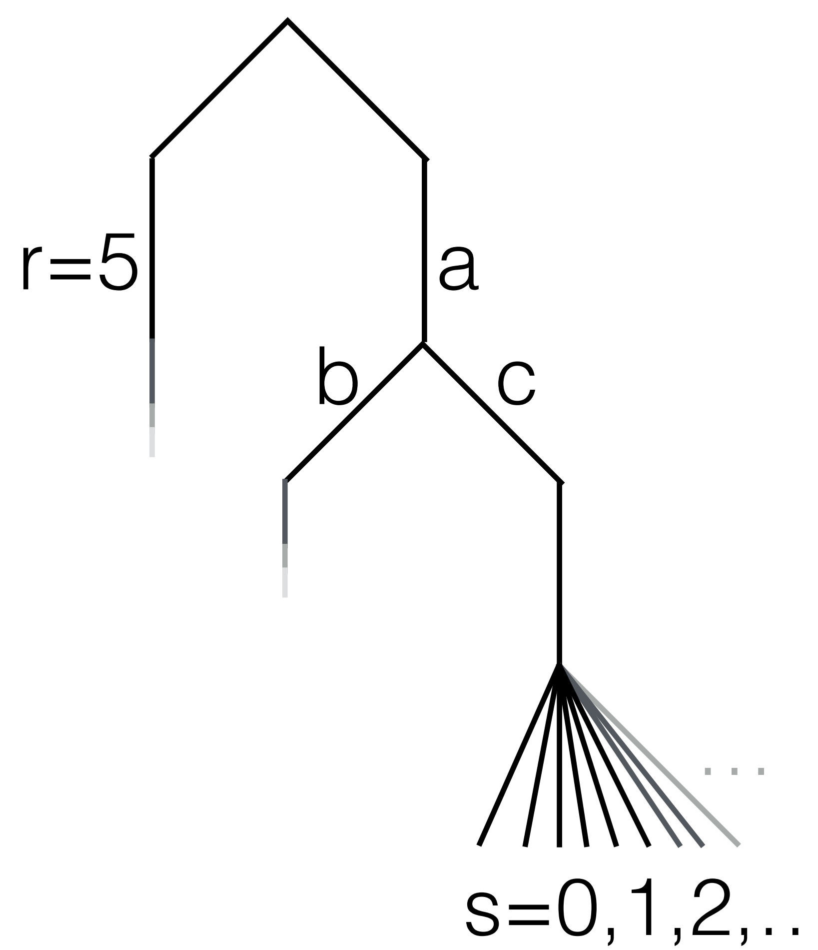
Strategies in games (wbga)

Strategies are used to define the property of being winning.

(Standard) strategies (for the existential player) are trees with vertex labelled nodes, such that

- either a vertex is owned by the existential player, and it has one child…
- or it is owned by the universal player, and it has as many children as successors in the arena …

Now…



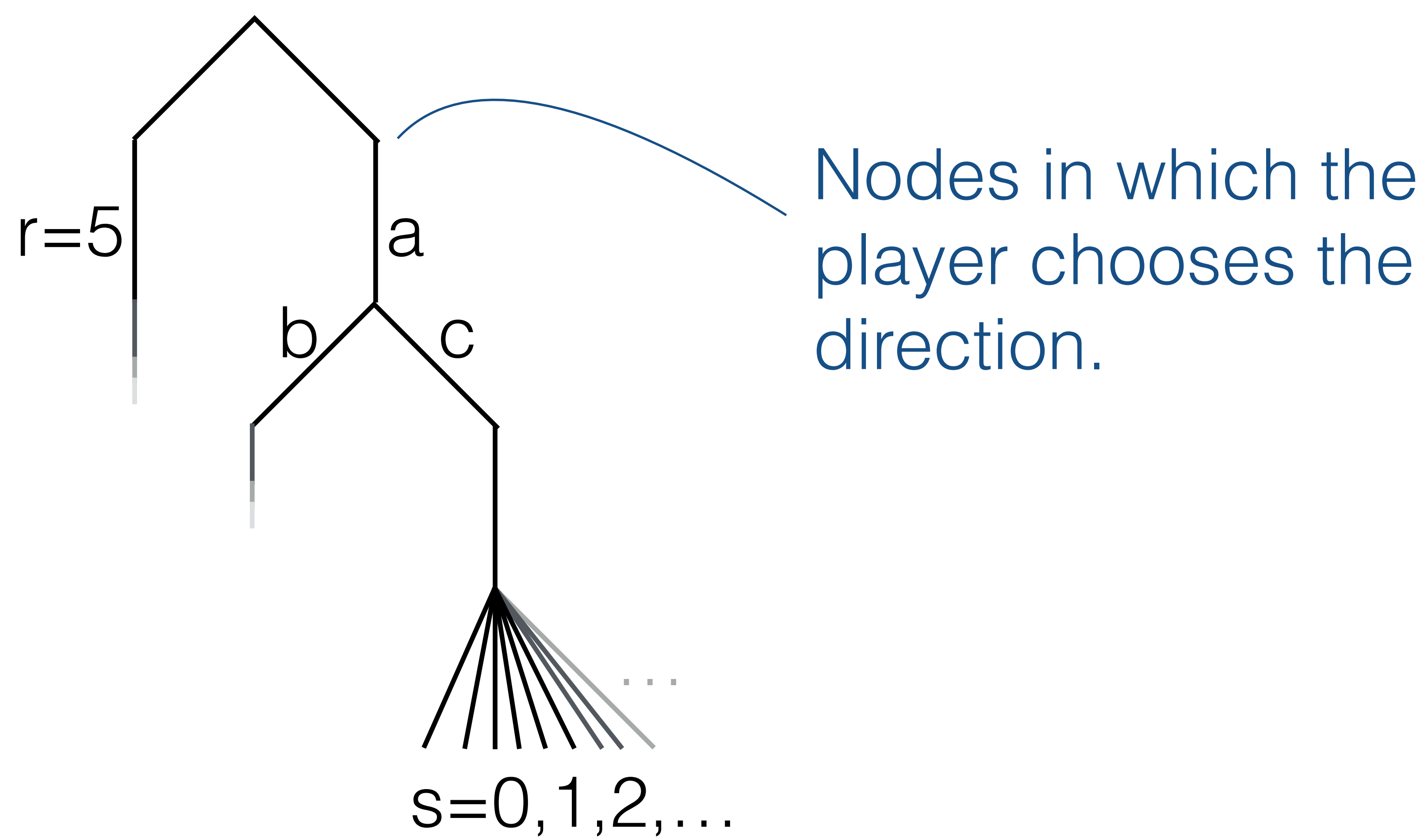
Strategies in games (wbga)

Strategies are used to define the property of being winning.

(Standard) strategies (for the existential player) are trees with vertex labelled nodes, such that

- either a vertex is owned by the existential player, and it has one child…
- or it is owned by the universal player, and it has as many children as successors in the arena …

Now…



Strategies in games (wbga)

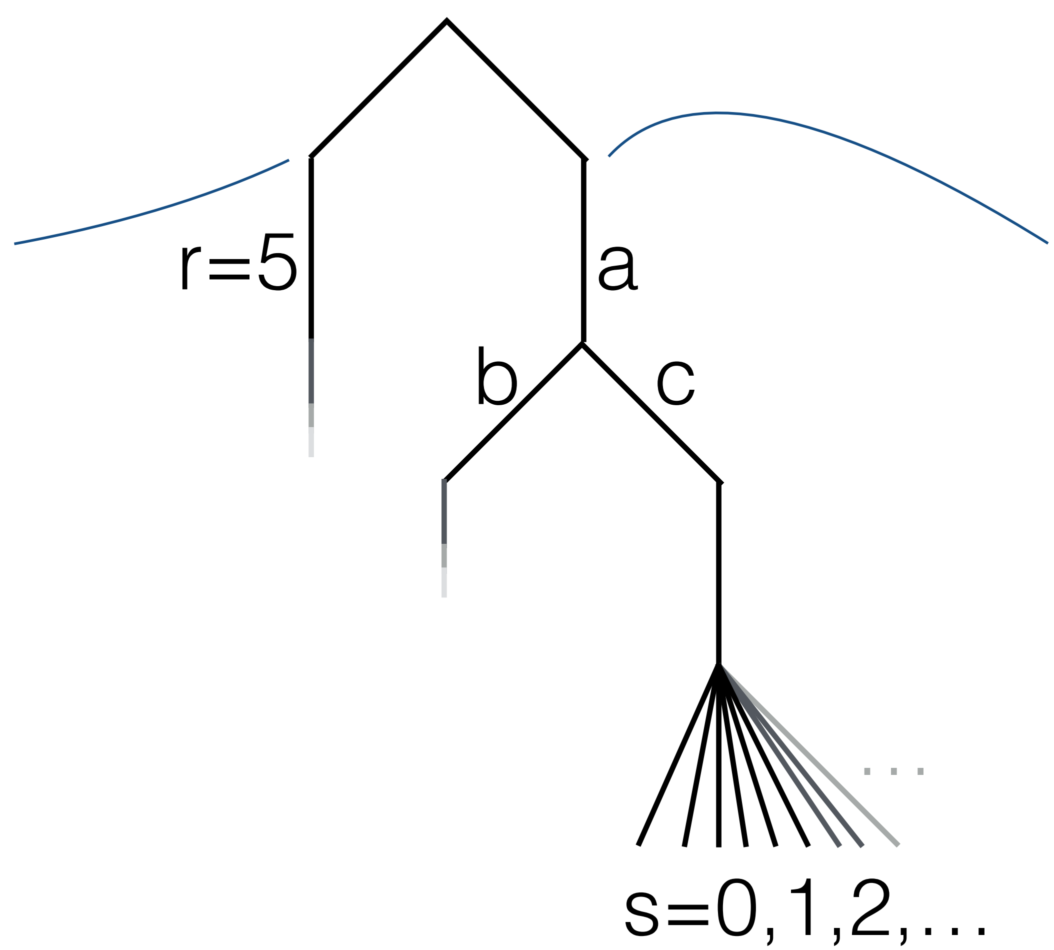
Strategies are used to define the property of being winning.

(Standard) strategies (for the existential player) are trees with vertex labelled nodes, such that

- either a vertex is owned by the existential player, and it has one child…
- or it is owned by the universal player, and it has as many children as successors in the arena …

Now…

Nodes in which the player chooses a value of a register



Nodes in which the player chooses the direction.

Strategies in games (wbga)

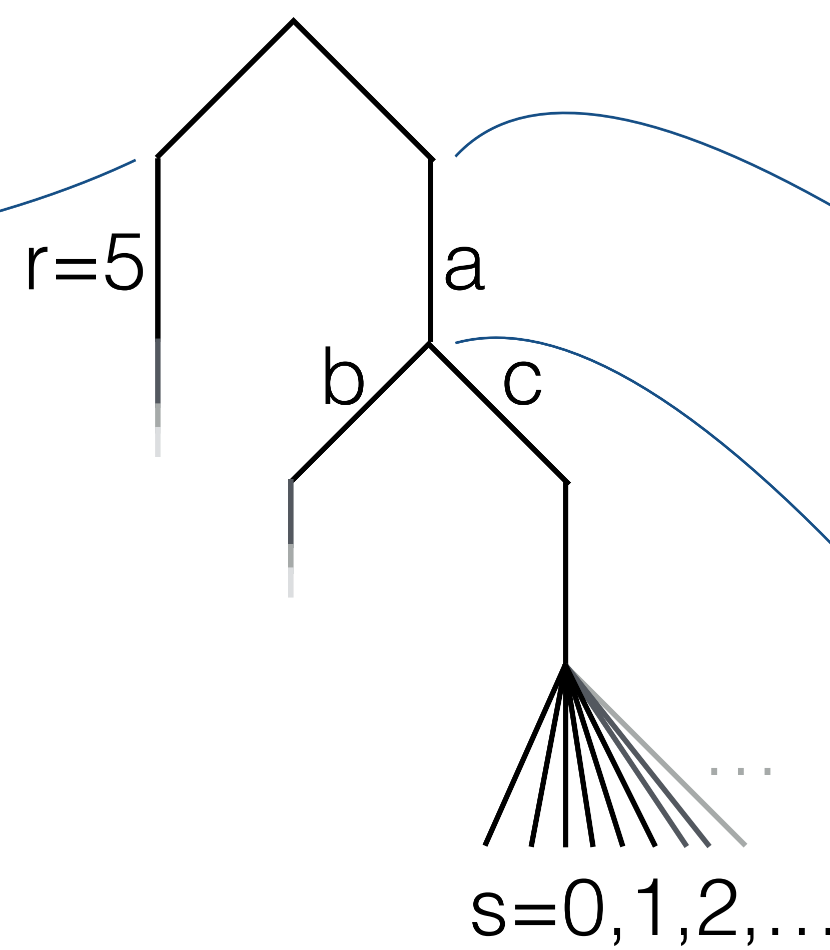
Strategies are used to define the property of being winning.

(Standard) **strategies** (for the existential player) are trees with vertex labelled nodes, such that

- either a vertex is owned by the existential player, and it has one child…
- or it is owned by the universal player, and it has as many children as successors in the arena …

Now…

Nodes in which the player chooses a value of a register



Nodes in which the player chooses the direction.

Nodes corresponding to the opponent choosing the direction.

Strategies in games (wbga)

Strategies are used to define the property of being winning.

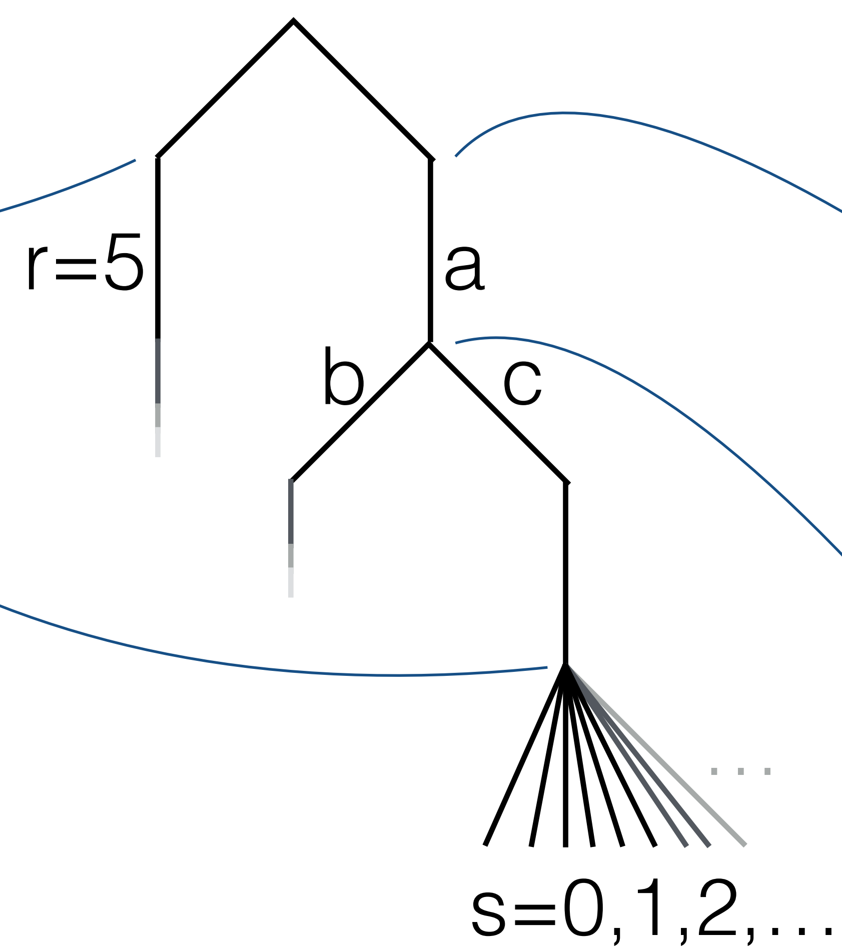
(Standard) strategies (for the existential player) are trees with vertex labelled nodes, such that

- either a vertex is owned by the existential player, and it has one child…
- or it is owned by the universal player, and it has as many children as successors in the arena …

Now…

Nodes in which the player chooses a value of a register

Infinitely branching nodes in which the opponent may choose any value for one of its registers.



Nodes in which the player chooses the direction.

Nodes corresponding to the opponent choosing the direction.

The results

The results

Games with bound guess actions **in general form**:

- quantities = regular cost function
- global condition = any ω-regular language (positive)

Theorem: The winner of a finite game with bound guess actions in general form can be decided.

The results

Games with bound guess actions **in general form**:
- quantities = regular cost function
- global condition = any ω-regular language (positive)

Lemma(reduction 1): A finite game with bound guess actions in general form can be effectively turned into a simple finite game with bound actions of same winner.

Theorem: The winner of a finite game with bound guess actions in general form can be decided.

The results

Games with bound guess actions **in general form**:

- quantities = regular cost function
- global condition = any ω-regular language (positive)

Simple games with bound guess actions:

- quantities = max over several counters ɣ of

  « the number of **inc**$_\gamma$ since the last **reset**$_\gamma$ or the beginning of the word »
- global condition =

  \+ first time a quantity exceeds its register, the owner immediately looses

  \+ if no quantity exceeds its value, an ω-regular language is used.

Lemma(reduction 1): A finite game with bound guess actions in general form can be effectively turned into a simple finite game with bound actions of same winner.

Theorem: The winner of a finite game with bound guess actions in general form can be decided.

The results

Games with bound guess actions **in general form**:

- quantities = regular cost function
- global condition = any ω-regular language (positive)

Simple games with bound guess actions:

- quantities = max over several counters ɣ of
  « the number of **inc**$_\gamma$ since the last **reset**$_\gamma$ or the beginning of the word »
- global condition =
  + first time a quantity exceeds its register, the owner immediately looses
  + if no quantity exceeds its value, an ω-regular language is used.

Lemma(reduction 1): A finite game with bound guess actions in general form can be effectively turned into a simple finite game with bound actions of same winner.

Lemma(reduction 2): A finite simple game with bound guess actions can be effectively turned into a finite ω-regular game with of same winner.

Theorem: The winner of a finite game with bound guess actions in general form can be decided.

First reduction

First reduction

Games with bound guess actions in general form:
- quantities = regular cost function
- global condition = any ω-regular language (positive)

Simple games with bound guess actions:
- quantities = max over several counters ɣ of
  « the number of **inc**$_\gamma$ since the last **reset**$_\gamma$ or the beginning of the word »
- global condition =
  + first time a quantity exceeds its register, the owner immediately looses
  + if no quantity exceeds its value, an ω-regular language is used.

Lemma(reduction 1): A finite game with bound guess actions in general form can be effectively turned into a simple finite game with bound actions of same winner.

First reduction

Games with bound guess actions in general form:

- quantities = regular cost function
- global condition = any ω-regular language (positive)

Main change

Simple games with bound guess actions:

- quantities = max over several counters γ of
  « the number of **inc**$_\gamma$ since the last **reset**$_\gamma$ or the beginning of the word »
- global condition =
  + first time a quantity exceeds its register, the owner immediately looses
  + if no quantity exceeds its value, an ω-regular language is used.

Lemma(reduction 1): A finite game with bound guess actions in general form can be effectively turned into a simple finite game with bound actions of same winner.

Regular cost functions

Def: regular cost functions are functions of the form

$$f : A^* \to \mathbb{N} \cup \{\infty\}$$

considered modulo an equivalence relation $\approx$ (that does not matter here).

Regular cost functions

Def: regular cost functions are functions of the form

$$f : A^* \to \mathbb{N} \cup \{\infty\}$$

considered modulo an equivalence relation $\approx$ (that does not matter here).

For a regular cost function, the following statements are equivalent:

- being definable in cost monadic second-order logic (costMSO)
- being described by a B-automaton, an S-automaton,
- being described a B-regular expression, or an S-regular expression,
- being recognized by a stabilisation monoid.

Regular cost functions

Def: regular cost functions are functions of the form

$$f : A^* \to \mathbb{N} \cup \{\infty\}$$

considered modulo an equivalence relation ≈ (that does not matter here).

For a regular cost function, the following statements are equivalent:

- being definable in cost monadic second-order logic (costMSO)
- being described by a B-automaton, an S-automaton,
- being described a B-regular expression, or an S-regular expression,
- being recognized by a stabilisation monoid.

Furthermore, several problems are decidable like the (modulo version of) equality of the (modulo version of) inequality.

Regular cost functions

Def: regular cost functions are functions of the form

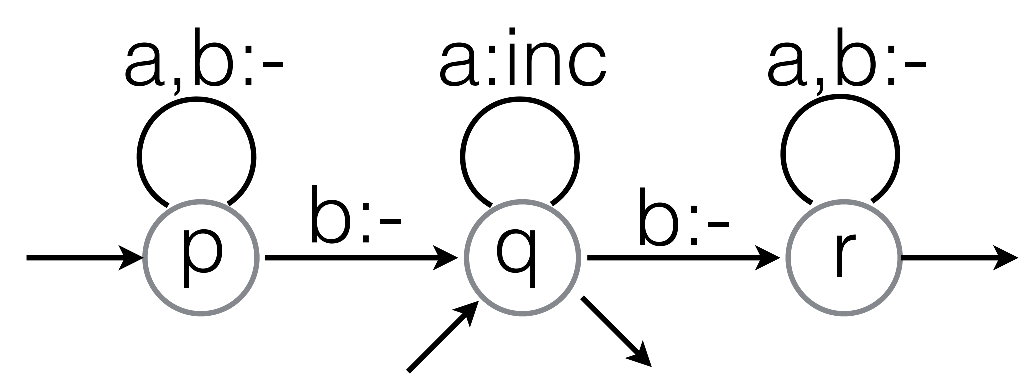
$$f : A^* \to \mathbb{N} \cup \{\infty\}$$

considered modulo an equivalence relation ≈ (that does not matter here).

For a regular cost function, the following statements are equivalent:
- being definable in cost monadic second-order logic (costMSO)
- being described by a B-automaton, an S-automaton,
- being described a B-regular expression, or an S-regular expression,
- being recognized by a stabilisation monoid.

Furthermore, several problems are decidable like the (modulo version of) equality of the (modulo version of) inequality.

A B-automaton has counters that can be incremented or reset
It accepts a word with value n if there exists an accepting run such that no counter exceeds value n.



First reduction

Games with bound guess actions in general form:

- quantities = regular cost function
- global condition = any ω-regular language (positive)

Main change

Simple games with bound guess actions:

- quantities = max over several counters γ of
  « the number of **inc**$_\gamma$ since the last **reset**$_\gamma$ or the beginning of the word »
- global condition =
  + first time a quantity exceeds its register, the owner immediately looses
  + if no quantity exceeds its value, an ω-regular language is used.

Lemma(reduction 1): A finite game with bound guess actions in general form can be effectively turned into a simple finite game with bound actions of same winner.

First reduction

Games with bound guess actions in general form:
- quantities = regular cost function
- global condition = any ω-regular language (positive)

Main change

Simple games with bound guess actions:
- quantities = ma B-condition rs ɣ of
  « the number eset$_\gamma$ or the beginning of the word »
- global condition =
  + first time a quantity exceeds its register, the owner immediately looses
  + if no quantity exceeds its value, an ω-regular language is used.

Lemma(reduction 1): A finite game with bound guess actions in general form can be effectively turned into a simple finite game with bound actions of same winner.

Reduction by transduction

Standard generic reduction technique (winning condition transduction):

L-game	$\otimes$	deterministic W-automaton for L	=	W-game of same winner

Reduction by transduction

Standard generic reduction technique (winning condition transduction):

L-game ⊗ **deterministic** W-automaton for L = W-game of same winner

- L-game: the winning condition
- W-automaton: the accepting condition
- for L: the accepted language
- W-game: the winning condition

Reduction by transduction

Standard generic reduction technique (winning condition transduction):

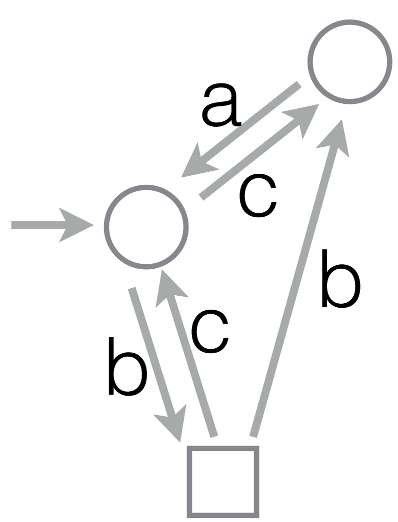
L-game $\otimes$ **deterministic** W-automaton for L = W-game of same winner

the winning condition

the accepting condition

the accepted language

the winning condition



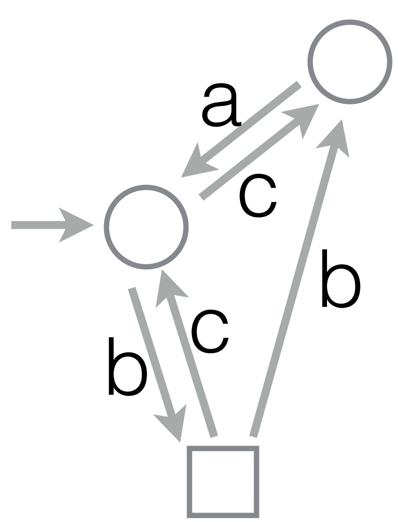
L=« infinitely many a's and infinitely many b's »

Reduction by transduction

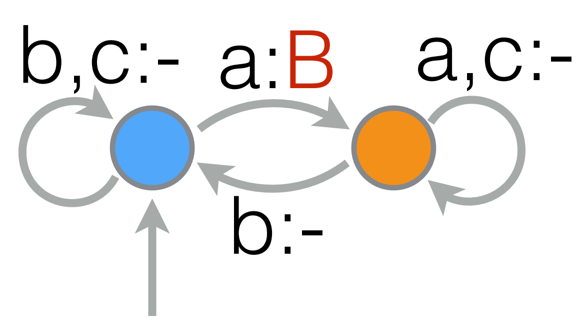
Standard generic reduction technique (winning condition transduction):

L-game ⊗ **deterministic** W-automaton for L = W-game of same winner

- L-game: L is the winning condition
- W-automaton for L: W is the accepting condition; L is the accepted language
- W-game: W is the winning condition



⊗



L=« infinitely many a's and infinitely many b's »

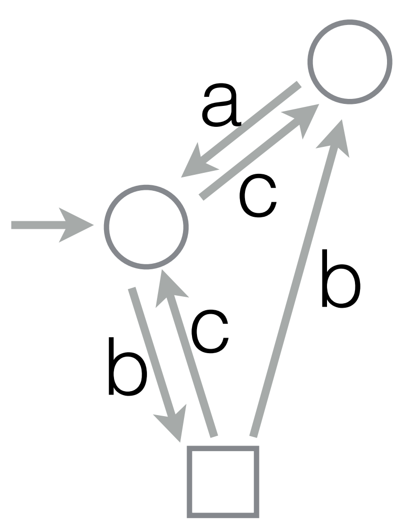
deterministic Büchi-automaton for L

Reduction by transduction

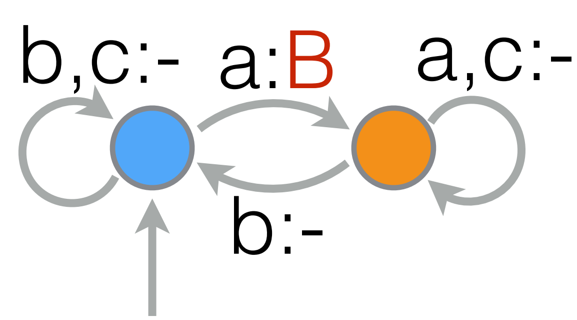
Standard generic reduction technique (winning condition transduction):

L-game ⊗ **deterministic** W-automaton for L = W-game of same winner

the winning condition | the accepting condition | the accepted language | the winning condition



⊗



L=« infinitely many a's and infinitely many b's »

deterministic Büchi-automaton for L

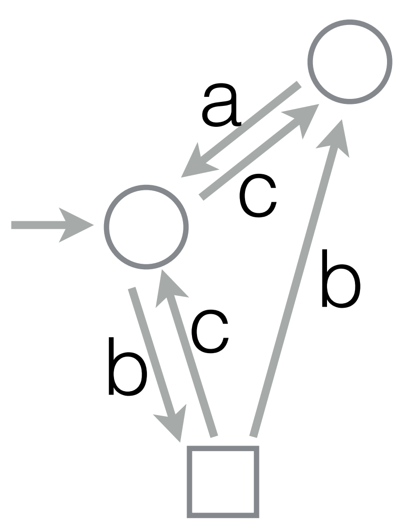
Büchi=« infinitely many B's »

Reduction by transduction

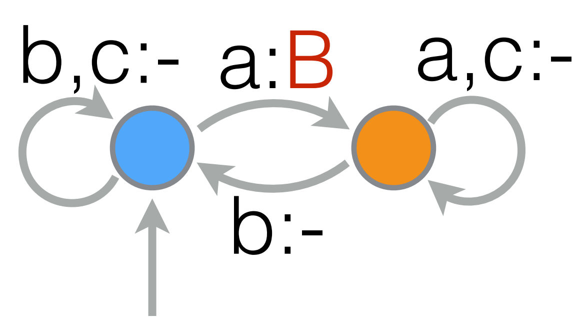
Standard generic reduction technique (winning condition transduction):

L-game ⊗ **deterministic** W-automaton for L = W-game of same winner

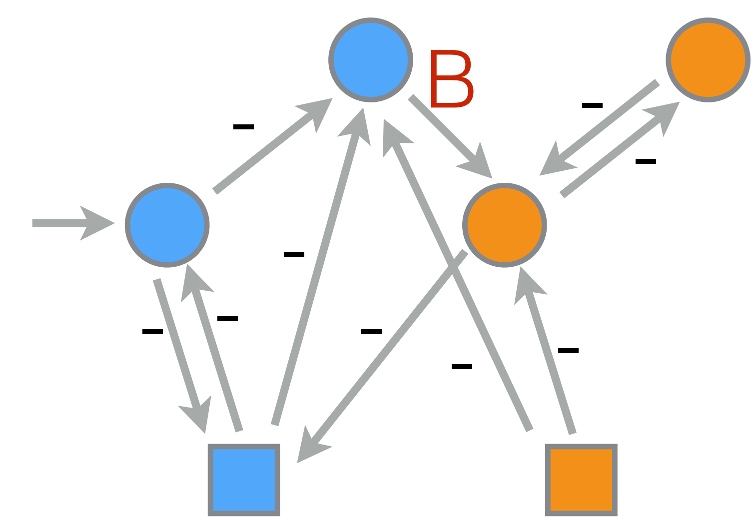
the winning condition (L-game) — the accepting condition / the accepted language (W-automaton for L) — the winning condition (W-game)



⊗



=



L=« infinitely many a's and infinitely many b's »

deterministic Büchi-automaton for L

Büchi=« infinitely many B's »

Büchi-game of same winner

First reduction

Games with bound guess actions in general form:

- quantities = regular cost function
- global condition = any ω-regular language (positive)

Main change

Simple games with bound guess actions:

- quantities = ma … rs ɣ of
  « the number … eset$_\gamma$ or the beginning of the word »

B-condition

- global condition =
  + first time a quantity exceeds its register, the owner immediately looses
  + if no quantity exceeds its value, an ω-regular language is used.

Lemma(reduction 1): A finite game with bound guess actions in general form can be effectively turned into a simple finite game with bound actions of same winner.

First reduction

Games with bound guess actions in general form:
- quantities = regular cost function
- global condition = any ω-regular language (positive)

Main change

Simple games with bound guess actions:
- quantities = ma rs ɣ of
  « the number eset$_{\gamma}$ or the beginning of the word »

B-condition

- global condition =
  + first time a quantity exceeds its register, the owner immediately looses
  + if no quantity exceeds its value, an ω-regular language is used.

Lemma(reduction 1): A finite game with bound guess actions in general form can be effectively turned into a simple finite game with bound actions of same winner.

It would be « sufficient » to compose with a deterministic B-automaton

History-determinism

L-game $\otimes$ deterministic W-automaton for L = W-game of same winner

History-determinism

L-game $\otimes$ deterministic W-automaton for L = W-game of same winner

For regular cost functions (as opposed to ω-regular languages),
not all regular cost functions are accepted by a deterministic automaton.

History-determinism

L-game	$\otimes$	deterministic W-automaton for L	=	W-game of same winner

For regular cost functions (as opposed to ω-regular languages),
not all regular cost functions are accepted by a deterministic automaton.

Remark: If the automaton is not deterministic (even alternating), $\otimes$ is well defined…

History-determinism

L-game	$\otimes$	deterministic W-automaton for L	=	W-game of same winner

For regular cost functions (as opposed to ω-regular languages),
not all regular cost functions are accepted by a deterministic automaton.

Remark: If the automaton is not deterministic (even alternating), $\otimes$ is well defined… but the product game may have a different winner.

History-determinism

L-game $\otimes$ deterministic W-automaton for L = W-game of same winner

For regular cost functions (as opposed to ω-regular languages), not all regular cost functions are accepted by a deterministic automaton.

Remark: If the automaton is not deterministic (even alternating), $\otimes$ is well defined… but the product game may have a different winner.

An automaton is good-for-game (=history-deterministic) if this product deserves the winner for all games.

History-determinism

L-game	$\otimes$	deterministic W-automaton for L	=	W-game of same winner

For regular cost functions (as opposed to ω-regular languages),
not all regular cost functions are accepted by a deterministic automaton.

Remark: If the automaton is not deterministic (even alternating), $\otimes$ is well defined… but the product game may have a different winner.

An automaton is good-for-game (=history-deterministic) if this product deserves the winner for all games.

Theorem (C.09/C.Unp/C.&Fijalkow 16): Every regular cost function is accepted by an history-deterministic B-automaton.

History-determinism

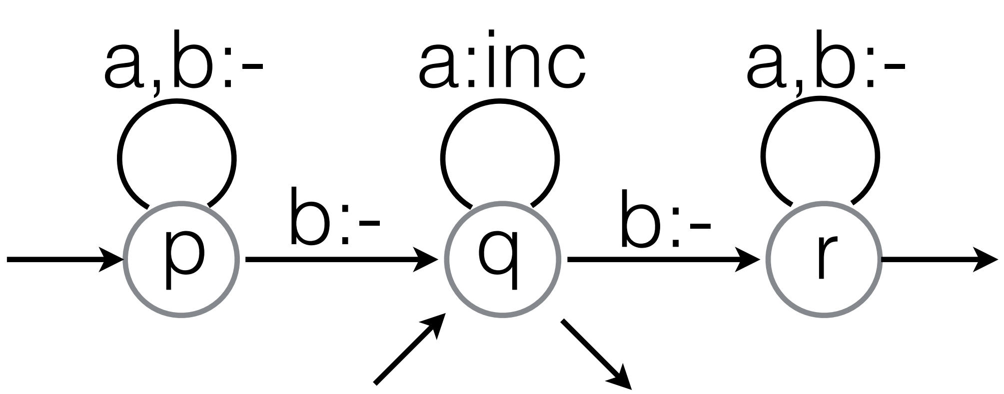
L-game $\otimes$ deterministic W-automaton for L = W-game of same winner

For regular cost functions (as opposed to ω-regular languages), not all regular cost functions are accepted by a deterministic automaton.

Remark: If the automaton is not deterministic (even alternating), $\otimes$ is well defined… but the product game may have a different winner.

An automaton is **good-for-game** (=history-deterministic) if this product deserves the winner for all games.

Theorem (C.09/C.Unp/C.&Fijalkow 16): Every regular cost function is accepted by an history-deterministic B-automaton.



History-determinism

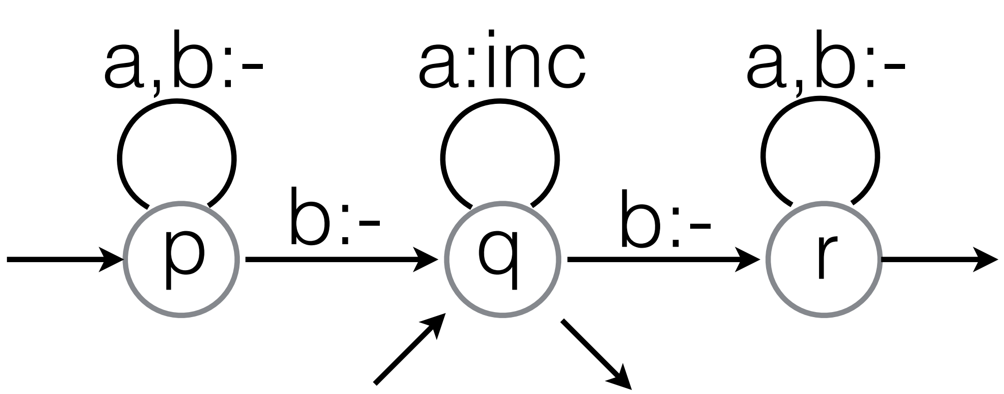
L-game ⊗ deterministic W-automaton for L = W-game of same winner

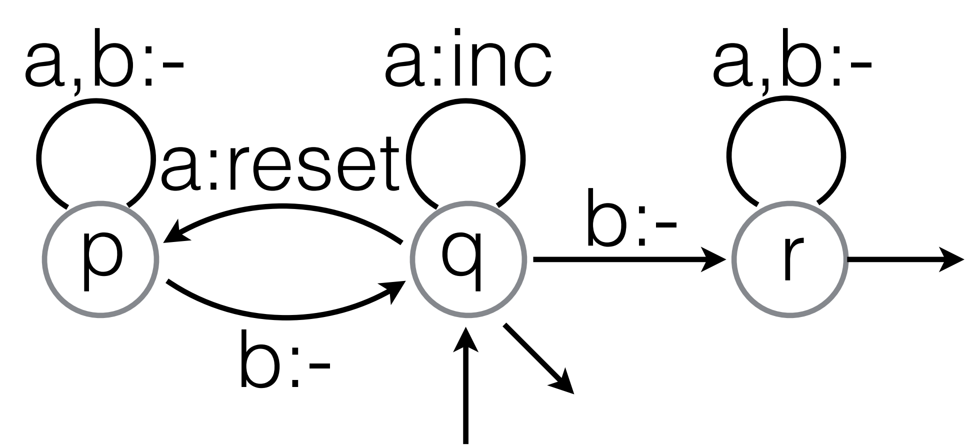
For regular cost functions (as opposed to ω-regular languages), not all regular cost functions are accepted by a deterministic automaton.

Remark: If the automaton is not deterministic (even alternating), ⊗ is well defined… but the product game may have a different winner.

An automaton is **good-for-game** (=history-deterministic) if this product deserves the winner for all games.

Theorem (C.09/C.Unp/C.&Fijalkow 16): Every regular cost function is accepted by an history-deterministic B-automaton.





First reduction

Games with bound guess actions in general form:
- quantities = regular cost function
- global condition = any ω-regular language (positive)

Main change

Simple games with bound guess actions:
- quantities = ma **B-condition** rs ɣ of
  « the number eset$_\gamma$ or the beginning of the word »
- global condition =
  + first time a quantity exceeds its register, the owner immediately looses
  + if no quantity exceeds its value, an ω-regular language is used.

Lemma(reduction 1): A finite game with bound guess actions in general form can be effectively turned into a simple finite game with bound actions of same winner.

It would be « sufficient » to compose with a deterministic B-automaton

First reduction

Games with bound guess actions in general form:

- quantities = regular cost function
- global condition = any ω-regular language (positive)

Main change

Simple games with bound guess actions:

- quantities = ma B-condition rs ɣ of
  « the number eset$_\gamma$ or the beginning of the word »
- global condition =
  + first time a quantity exceeds its register, the owner immediately looses
  + if no quantity exceeds its value, an ω-regular language is used.

Lemma(reduction 1): A finite game with bound guess actions in general form can be effectively turned into a simple finite game with bound actions of same winner.

~~It would be « sufficient » to compose with a deterministic B-automaton~~

First reduction

Games with bound guess actions in general form:

- quantities = regular cost function
- global condition = any ω-regular language (positive)

Main change

Simple games with bound guess actions:

- quantities = ma B-condition rs ɣ of
  « the number eset$_{\gamma}$ or the beginning of the word »
- global condition =
  + first time a quantity exceeds its register, the owner immediately looses
  + if no quantity exceeds its value, an ω-regular language is used.

Lemma(reduction 1): A finite game with bound guess actions in general form can be effectively turned into a simple finite game with bound actions of same winner.

~~It would be « sufficient » to compose with a deterministic B-automaton~~

It is sufficient to compose with history-deterministic B-automata.

Second reduction

Second reduction

Simple games with bound guess actions:

- quantities = max over several counters γ of
  « the number of **inc**$_\gamma$ since the last **reset**$_\gamma$ or the beginning of the word »
- global condition =
  + first time a quantity exceeds its register, the owner immediately looses
  + if no quantity exceeds its value, an ω-regular language is used.

Lemma(reduction 2): A finite simple game with bound guess actions can be effectively turned into a finite ω-regular game with of same winner.

How values change

Positivity assumption:

« Whenever a player choses a value (through of a bound guess action), the winning condition is required to use this value as an upper bound in the definition of what it is winning for this player. »

How values change

Positivity assumption:

> « Whenever a player choses a value (through of a bound guess action), the winning condition is required to use this value as an upper bound in the definition of what it is winning for this player. »

Hence, a player, if he wins using a strategy, also wins using any identical strategy in which he would choose higher values of (his) registers.

How values change

Positivity assumption:

> « Whenever a player choses a value (through of a bound guess action), the winning condition is required to use this value as an upper bound in the definition of what it is winning for this player. »

Hence, a player, if he wins using a strategy, also wins using any identical strategy in which he would choose higher values of (his) registers.

Consequence 1: a slight modification of quantities (like doubling) does not change the winner of the game.

How values change

Positivity assumption:

> « Whenever a player choses a value (through of a bound guess action), the winning condition is required to use this value as an upper bound in the definition of what it is winning for this player. »

Hence, a player, if he wins using a strategy, also wins using any identical strategy in which he would choose higher values of (his) registers.

Consequence 1: a slight modification of quantities (like doubling) does not change the winner of the game.

Consequence 2: when a player chooses a value, he can (and should be thought of as) choose a value very large in front of al the values seen so far

How values change

Positivity assumption:

> « Whenever a player choses a value (through of a bound guess action), the winning condition is required to use this value as an upper bound in the definition of what it is winning for this player. »

Hence, a player, if he wins using a strategy, also wins using any identical strategy in which he would choose higher values of (his) registers.

Consequence 1: a slight modification of quantities (like doubling) does not change the winner of the game.

Consequence 2: when a player chooses a value, he can (and should be thought of as) choose a value very large in front of al the values seen so far

Thus, the order in which registers have been guessed gives an idea of their relative values/magnitude.

How values change

Positivity assumption:

> « Whenever a player choses a value (through of a bound guess action), the winning condition is required to use this value as an upper bound in the definition of what it is winning for this player. »

Hence, a player, if he wins using a strategy, also wins using any identical strategy in which he would choose higher values of (his) registers.

Consequence 1: a slight modification of quantities (like doubling) does not change the winner of the game.

Consequence 2: when a player chooses a value, he can (and should be thought of as) choose a value very large in front of al the values seen so far

Thus, the order in which registers have been guessed gives an idea of their relative values/magnitude.

by maintaining a permutation of the registers one may « know » during the game what is this order.

Second reduction

Simple games with bound guess actions:

- quantities = max over several counters ɣ of
  « the number of **inc**$_\gamma$ since the last **reset**$_\gamma$ or the beginning of the word »
- global condition =
  + first time a quantity exceeds its register, the owner immediately looses
  + if no quantity exceeds its value, an ω-regular language is used.

Lemma(reduction 2): A finite simple game with bound guess actions can be effectively turned into a finite ω-regular game with of same winner.

Second reduction

Simple games with bound guess actions:
- quantities = max over several counters ɣ of
  « the number of **inc**$_\gamma$ since the last **reset**$_\gamma$ or the beginning of the word »
- global condition =
  + first time a quantity exceeds its register, the owner immediately looses
  + if no quantity exceeds its value, an ω-regular language is used.

Lemma(reduction 2): A finite simple game with bound guess actions can be effectively turned into a finite ω-regular game with of same winner.

Using the permutation or register techniques, one can « essentially » restricts to a situation where
1) the registers are not guessed anymore,
2) their relative order (of magnitudes) is known.

From simple to ω-regular

We assume r1 « r2« r2« … « rk known (as if bound guess actions at init).

From simple to ω-regular

We assume r1 « r2« r2« … « rk known (as if bound guess actions at init).

For simplicity, we assume one counter per register.

From simple to ω-regular

We assume r1 « r2« r2« … « rk known (as if bound guess actions at init).

For simplicity, we assume one counter per register.

Simple condition:

- if some register gets its value exceeded, and it is the first such register, then its owner immediately looses,
- else the long term condition W decides the winner.

From simple to ω-regular

We assume r1 « r2« r2« … « rk known (as if bound guess actions at init).

For simplicity, we assume one counter per register.

Simple condition:

- if some register gets its value exceeded, and it is the first such register, then its owner immediately looses,
- else the long term condition W decides the winner.

Corresponding ω-regular condition:

- if there are infinitely many **inc1**, finitely many **reset1**, then owner1 looses, else
- …
- if there are infinitely many **inck**, finitely many **resetk**, then ownerk looses, else
- the long term condition W decides the winner.

From simple to ω-regular

We assume r1 « r2« r2« … « rk known (as if bound guess actions at init).

For simplicity, we assume one counter per register.

Simple condition:

- if some register gets its value exceeded, and it is the first such register, then its owner immediately looses,
- else the long term condition W decides the winner.

Corresponding ω-regular condition:

- if there are infinitely many **inc1**, finitely many **reset1**, then owner1 looses, else
- …
- if there are infinitely many **inck**, finitely many **resetk**, then ownerk looses, else
- the long term condition W decides the winner.

Lemma: For **finite** games (with bound guess actions at init), the simple condition, and the corresponding ω-regular condition have same winner.

From simple to ω-regular

We assume r1 « r2« r2« … « rk known (as if bound guess actions at init).

For simplicity, we assume one counter per register.

Simple condition:

- if some register gets its value exceeded, and it is the first such register, then its owner immediately looses,
- else the long term condition W decides the winner.

Corresponding ω-regular condition:

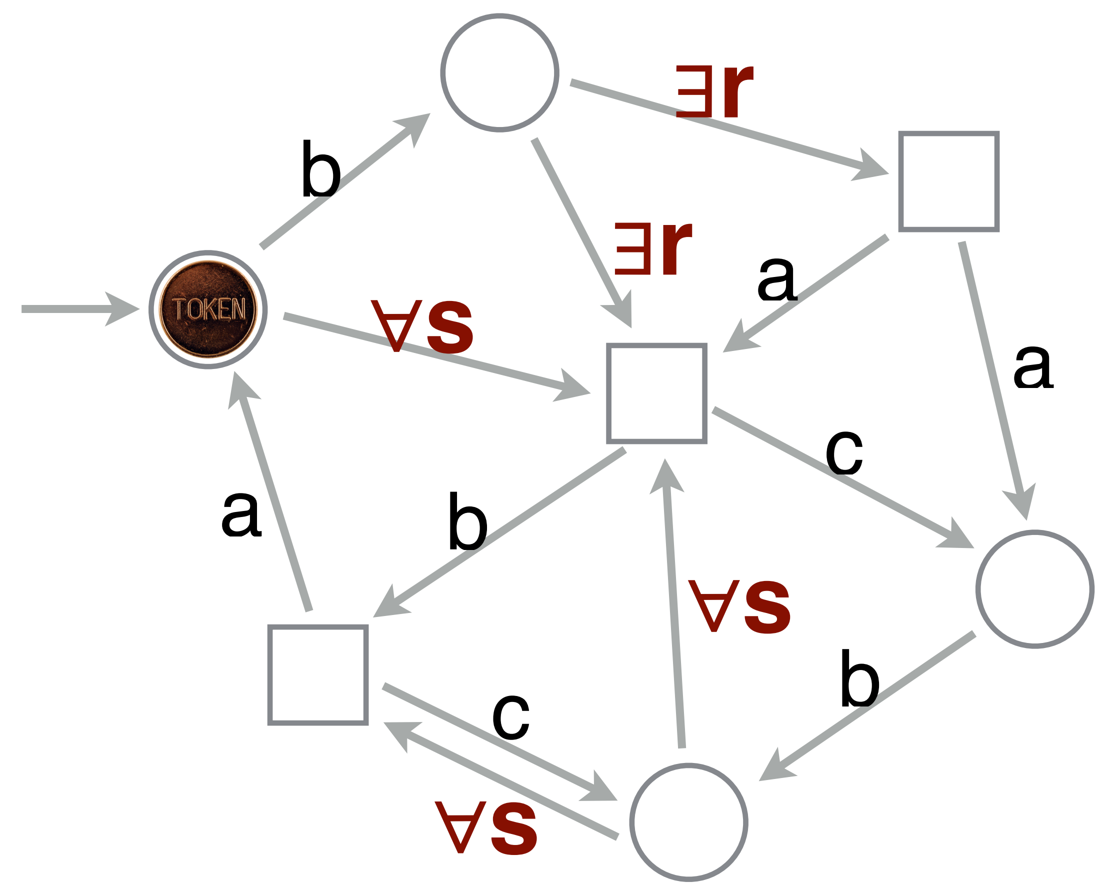
- if there are infinitely many **inc1**, finitely many **reset1**, then owner1 looses, else
- …
- if there are infinitely many **inck**, finitely many **resetk**, then ownerk looses, else
- the long term condition W decides the winner.

Lemma: For **finite** games (with bound guess actions at init), the simple condition, and the corresponding ω-regular condition have same winner.

The proof crucially uses the finiteness of the game, and the existence of finite memory strategies in ω-regular games.

Conclusion

Games with bound guess actions allow to describe phenomenon that virtually happen in infinite games.

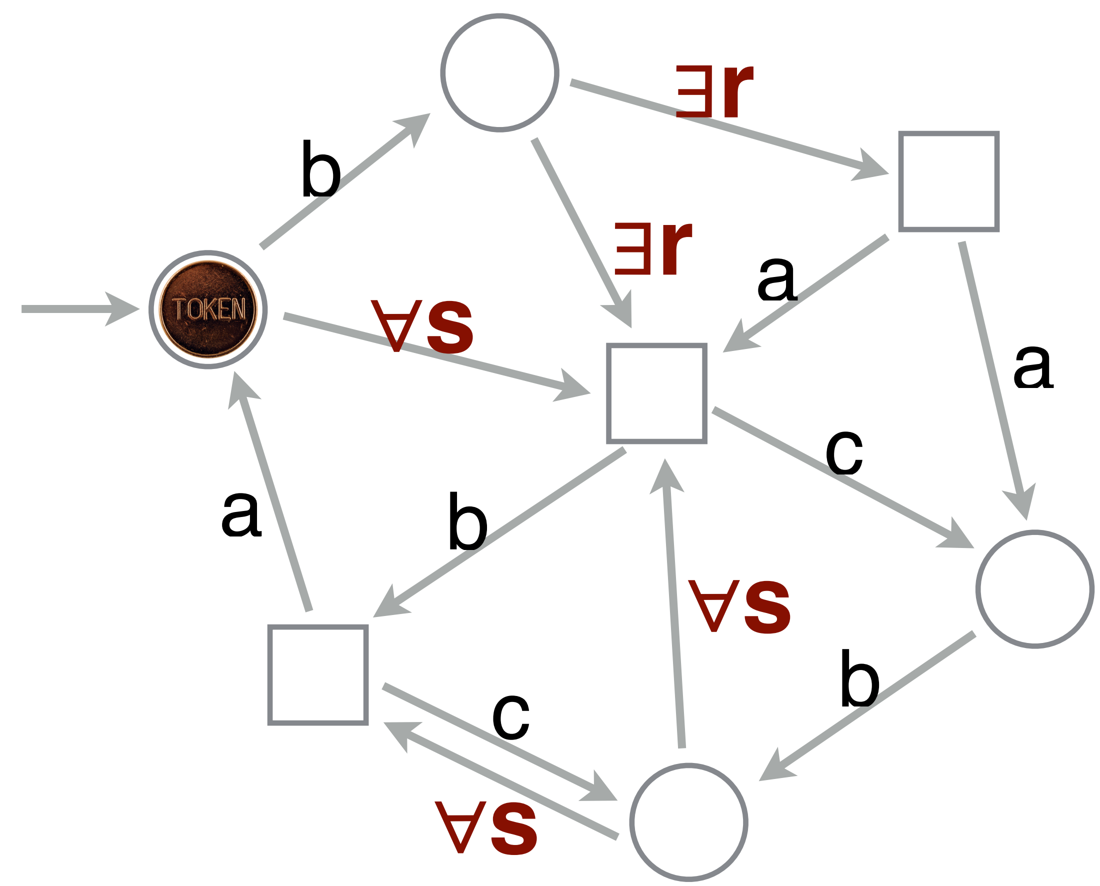


Conclusion

Games with bound guess actions allow to describe phenomenon that virtually happen in infinite games.

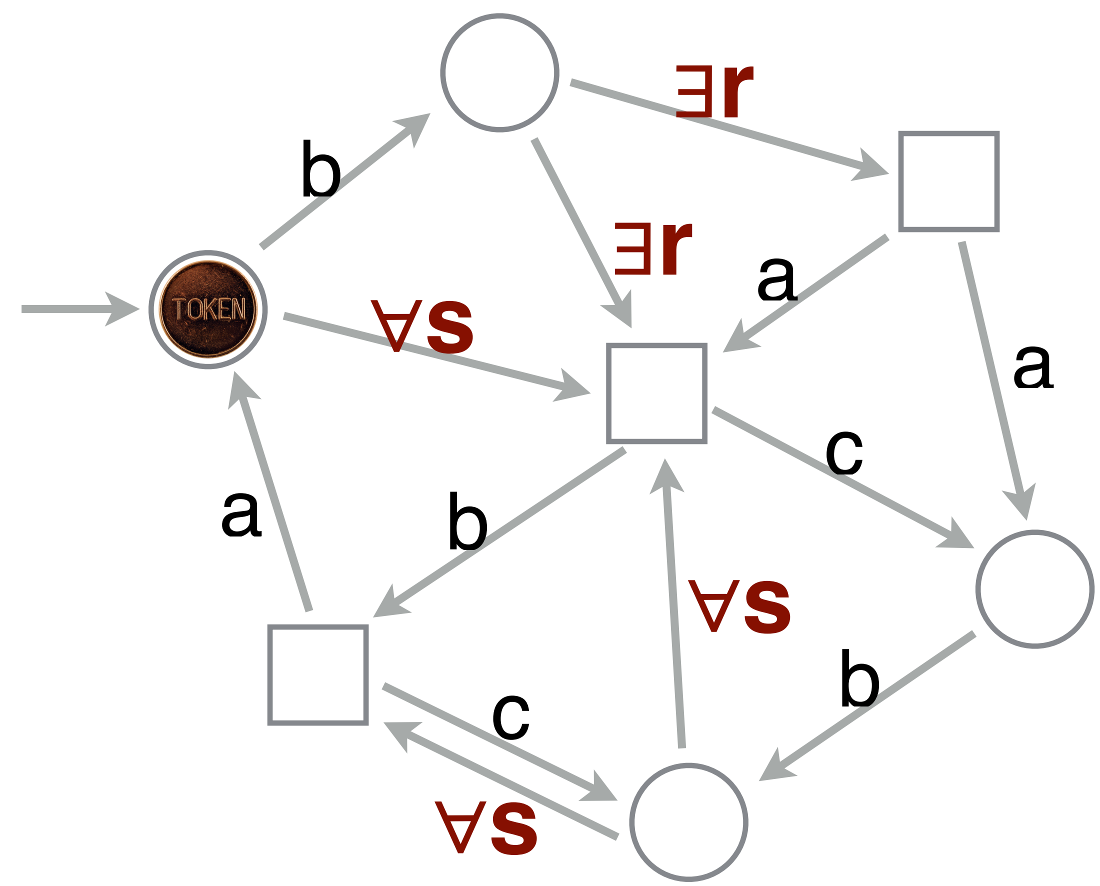
Finite such games with a reasonable class of conditions
- regular cost functions as quantities,
- regular condition as long term goal,

are decidable.



Conclusion

Games with bound guess actions allow to describe phenomenon that virtually happen in infinite games.



Finite such games with a reasonable class of conditions

- regular cost functions as quantities,
- regular condition as long term goal,

are decidable.

The proof goes into several step of reduction involving:

- history-deterministic cost automata,
- LAR-like technique for assessing relative magnitudes of register values,
- a final reduction to ω-regular condition.