Winning CaRet Games with Modular Strategies

Ilaria De Crescenzo and Salvatore La Torre

University of Salerno

Pescara, 31st August - CILC 2011

Pushdown systems

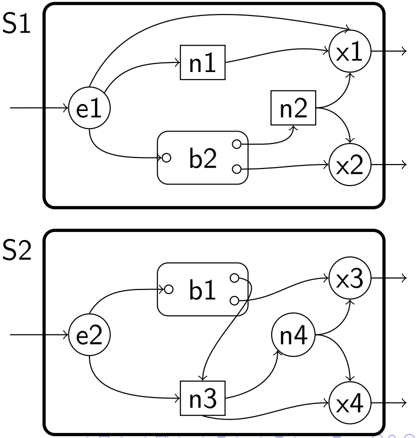
- In this work we focus on **pushdown systems**
- They can model systems with potentially recursive procedure calls, as:
  - control flows in programs of sequential imperative and object oriented programming languages
  - distributed architectures
  - communication protocols
- In an open pushdown system, some of the choices depend upon the controllable inputs and some represent uncontrollable nondeterminism (as a two-players game)

Recursive Game Graphs

We chose to use **recursive game graphs** as model.
An RGG is composed of **game modules**, one of which is the start module.

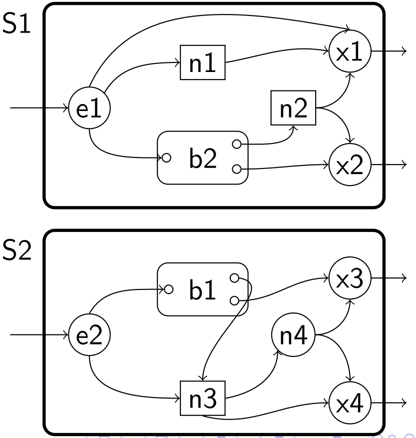
Recursive Game Graphs

We chose to use **recursive game graphs** as model.
An RGG is composed of **game modules**, one of which is the start module.



We chose to use **recursive game graphs** as model.
An RGG is composed of **game modules**, one of which is the start module.

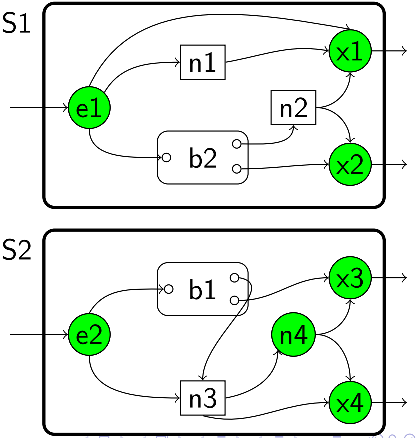
- Vertices are partitioned into two sets depending on player who controls the next move. The player are and



Recursive Game Graphs

We chose to use **recursive game graphs** as model.
An RGG is composed of **game modules**, one of which is the start module.

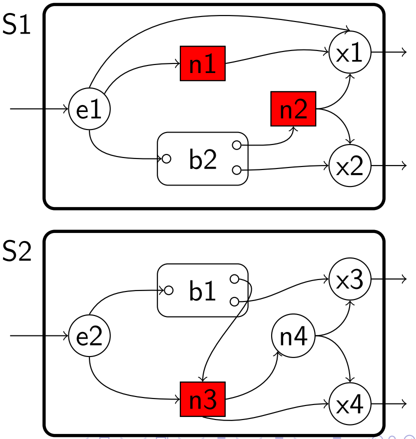
- Vertices are partitioned into two sets depending on player who controls the next move. The player are **GreenPlayer** and



Recursive Game Graphs

We chose to use **recursive game graphs** as model.
An RGG is composed of **game modules**, one of which is the start module.

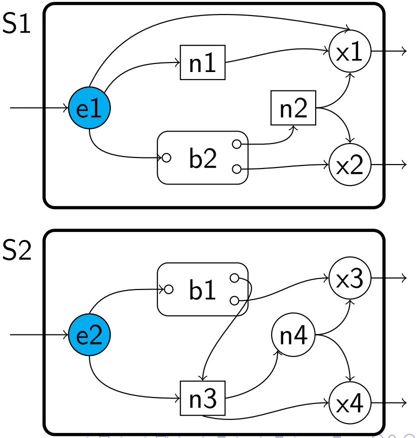
- Vertices are partitioned into two sets depending on player who controls the next move. The player are **GreenPlayer** and **RedPlayer**



Recursive Game Graphs

We chose to use **recursive game graphs** as model.
An RGG is composed of **game modules**, one of which is the start module.

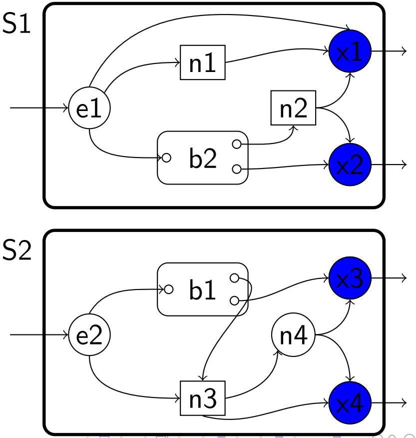
- Vertices are partitioned into two sets depending on player who controls the next move. The player are **GreenPlayer** and **RedPlayer**
- Each module has **entry** nodes



Recursive Game Graphs

We chose to use **recursive game graphs** as model.
An RGG is composed of **game modules**, one of which is the start module.

- Vertices are partitioned into two sets depending on player who controls the next move. The player are **GreenPlayer** and **RedPlayer**
- Each module has **entry** nodes and **exit** nodes.

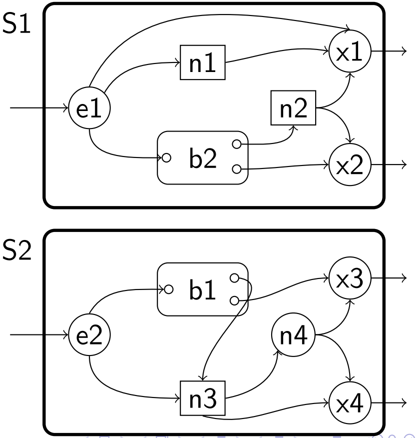


Recursive Game Graphs

We chose to use **recursive game graphs** as model.
An RGG is composed of **game modules**, one of which is the start module.

- Vertices are partitioned into two sets depending on player who controls the next move. The player are **GreenPlayer** and **RedPlayer**
- Each module has **entry** nodes and **exit** nodes.

The **boxes** map the other game modules.

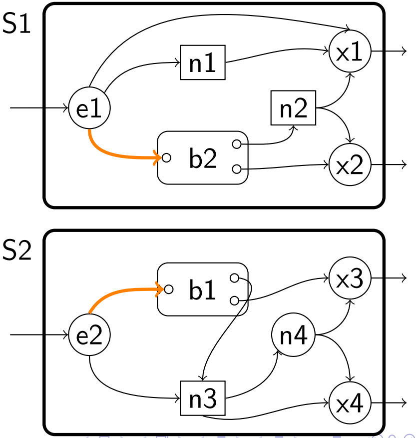


Recursive Game Graphs

We chose to use **recursive game graphs** as model.
An RGG is composed of **game modules**, one of which is the start module.

- Vertices are partitioned into two sets depending on player who controls the next move. The player are **GreenPlayer** and **RedPlayer**
- Each module has **entry** nodes and **exit** nodes.

The **boxes** map the other game modules.
- An edge that goes from a node to box is a **call**

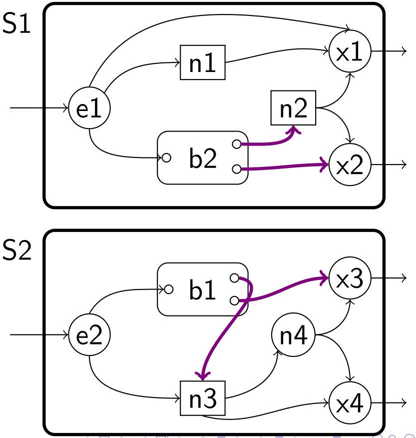


Recursive Game Graphs

We chose to use **recursive game graphs** as model.
An RGG is composed of **game modules**, one of which is the start module.

- Vertices are partitioned into two sets depending on player who controls the next move. The player are **GreenPlayer** and **RedPlayer**
- Each module has **entry** nodes and **exit** nodes.

The **boxes** map the other game modules.
- An edge that goes from a node to box is a call
- An edge that goes from a box to a node is a **return**



Recursive Game Graphs

A *play* of an RGG is a sequence of vertices starting from an entry node of the start module and from a node to its successor there is a permitted transition

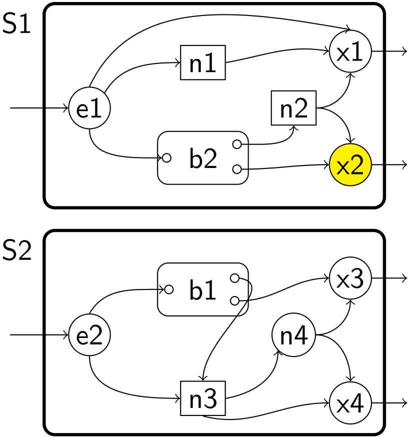
- A *winning set* over an RGG is a language on the alphabet of the RGG
- A winning set can be expressed by a winning condition (automata, formulas, etc...)
- A play is said to be *winning* if its labeling is a string that belongs to the winning set

Strategies

- A *strategy* for a player is a function that encodes how the player must play the game. It associates a move to every run that ends in a node controlled by that player
- A strategy is *winning* for a player if all plays according with that strategy are winning, whatever are the moves of the other player
- On RGGs can be defined a specific type of strategy, called **modular**
- A strategy is said modular iff it can only refer to the *local memory* of the module to choose the next move

Strategies

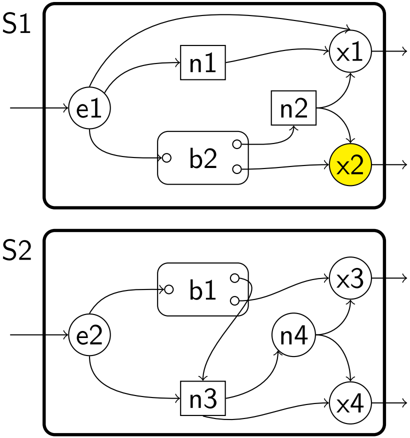
We consider a reachability condition on x2



Strategies

We consider a reachability condition on x2

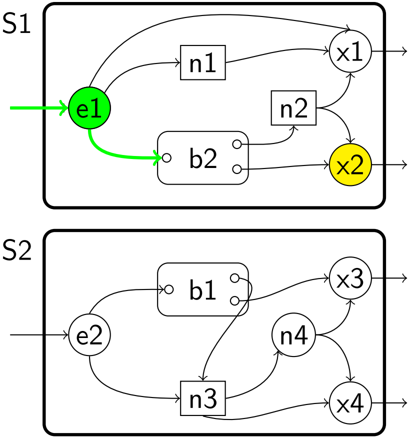
- The following strategy is winning:
  - from n4 go to x4
  - from e2 call b1
  - from e1, if the previous move was a call, go to x1, else go to b2



Strategies

We consider a reachability condition on x2

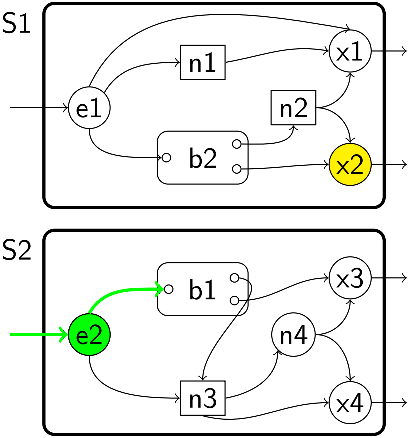
- The following strategy is winning:
  - from n4 go to x4
  - from e2 call b1
  - from e1, if the previous move was a call, go to x1, else go to b2



Strategies

We consider a reachability condition on x2

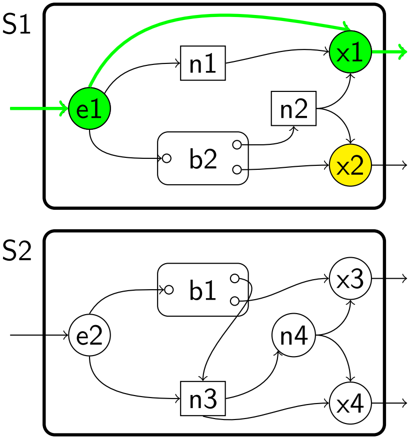
- The following strategy is winning:
  - from n4 go to x4
  - from e2 call b1
  - from e1, if the previous move was a call, go to x1, else go to b2



Strategies

We consider a reachability condition on x2

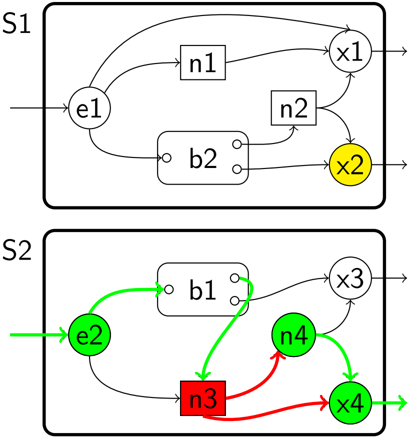
- The following strategy is winning:
  - from n4 go to x4
  - from e2 call b1
  - from e1, if the previous move was a call, go to x1, else go to b2



Strategies

We consider a reachability condition on x2

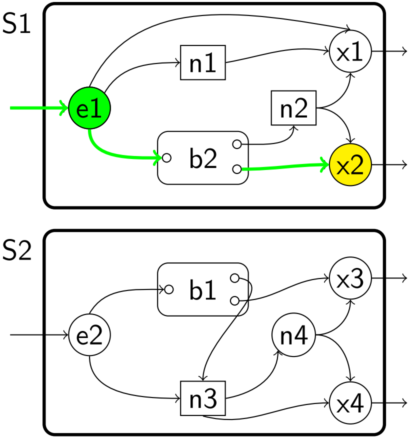
- The following strategy is winning:
  - from n4 go to x4
  - from e2 call b1
  - from e1, if the previous move was a call, go to x1, else go to b2



Strategies

We consider a reachability condition on x2

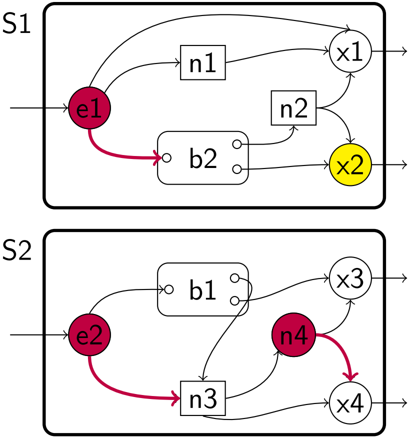
- The following strategy is winning:
  - from n4 go to x4
  - from e2 call b1
  - from e1, if the previous move was a call, go to x1, else go to b2



Strategies

We consider a reachability condition on x2

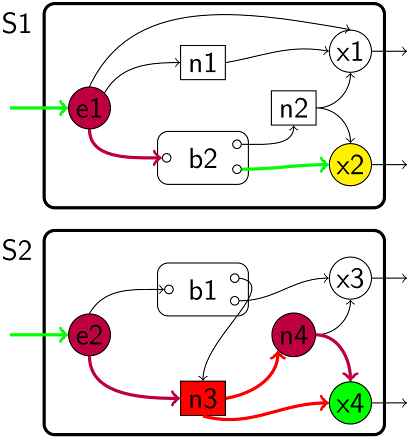
- The strategy seen before is not modular. because the third rule must know informations of the previous module to choose the internal move to e1 or the call to b2
- The strategy **modular** highlighted in this example is winning



Strategies

We consider a reachability condition on x2

- The strategy seen before is not modular. because the third rule must know informations of the previous module to choose the internal move to e1 or the call to b2
- The strategy **modular** highlighted in this example is winning



Reasons

Studying games on graphs allows to analize behaviours of *open systems*

- In open system settings, the execution depends on the interaction between a **controller** and an **external enviroment**
- Design a controller that supplies inputs to the system so that the product of controller and system satisfies correctness specification corresponds to computing *winning strategies* in two-player games

Reasons

Studying games on graphs allows to analize behaviours of *open systems*

- In open system settings, the execution depends on the interaction between a **controller** and an **external enviroment**

- Design a controller that supplies inputs to the system so that the product of controller and system satisfies correctness specification corresponds to computing *winning strategies* in two-player games

If a winning modular strategy can be found, it means that we can design for every module a controller that guarantees the correctness of the system **whatever is the context in which the module is invoked**

Problem

It is not always possibile to have a winning strategy for the protagonist on a RGG, and not all RGGs that have a winning strategy have also a winning modular strategy for the protagonist

Problem

It is not always possibile to have a winning strategy for the protagonist on a RGG, and not all RGGs that have a winning strategy have also a winning modular strategy for the protagonist

Problem

Given a RGG G and a winning condition, is there a modular winning strategy for the GreenPlayer?

Our contribution

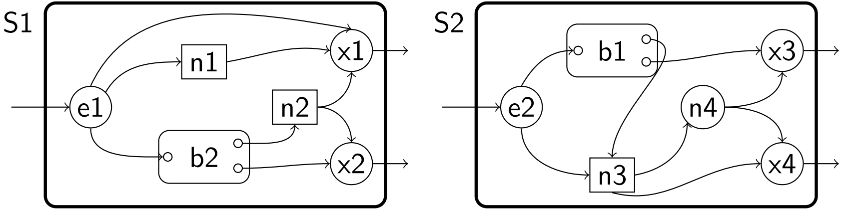
In this work we analize and resolve the problem for a new class of games that we have defined Modular CARET Games

- CaRet is a temporal logic designed for recursive machines
- Allows to express also stack inspection properties and specifications of partial and total correctness with respect to pre and post conditions
- The syntax is:

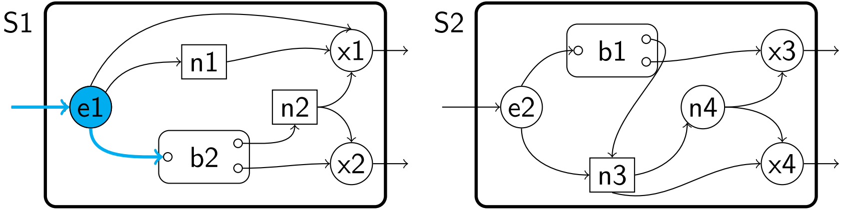
$$\varphi := p \mid \varphi \vee \varphi \mid \neg\, \varphi \mid \bigcirc^{g} \varphi \mid \bigcirc^{a} \varphi \mid \bigcirc^{-} \varphi \mid \varphi\, \mathcal{U}^{g} \varphi \mid \varphi\, \mathcal{U}^{a} \varphi \mid \varphi\, \mathcal{U}^{-} \varphi$$

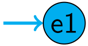
- Temporal operators are defined in three different **modalities:**
  - **Global successor**
  - **Abstract successor**
  - **Caller**

Differences between modalities

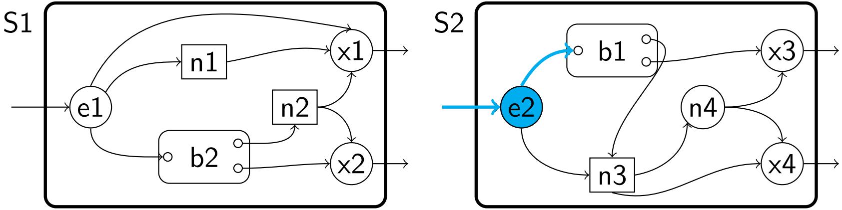


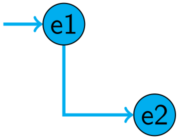
Differences between modalities



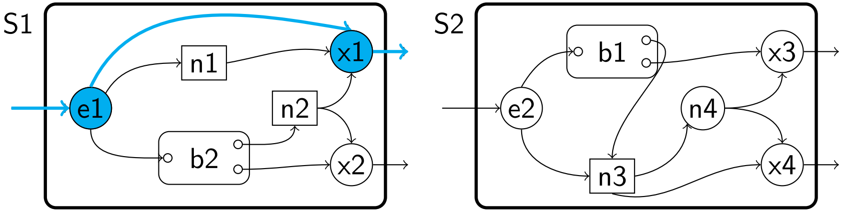


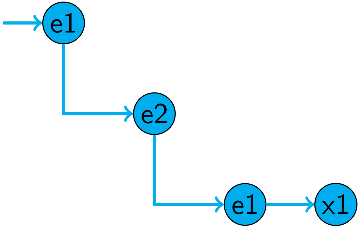
Differences between modalities



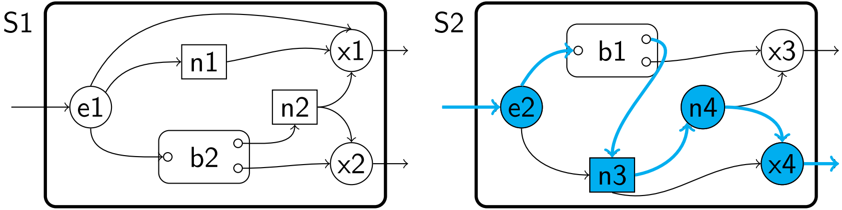


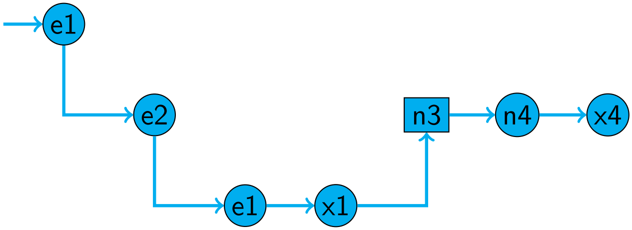
Differences between modalities



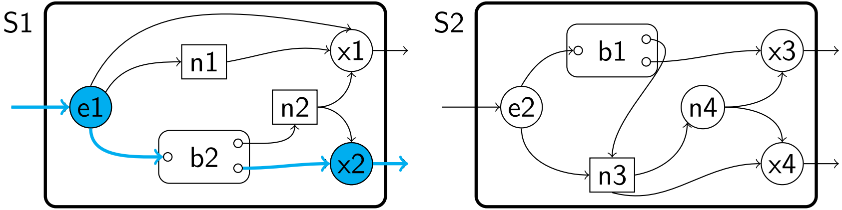


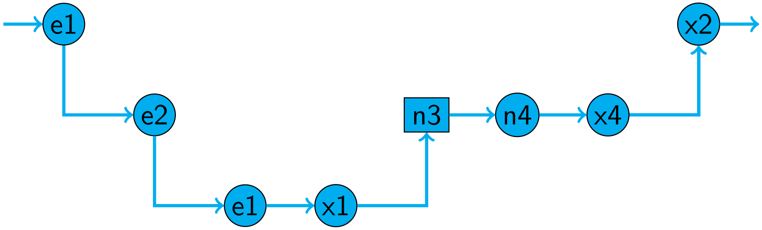
Differences between modalities



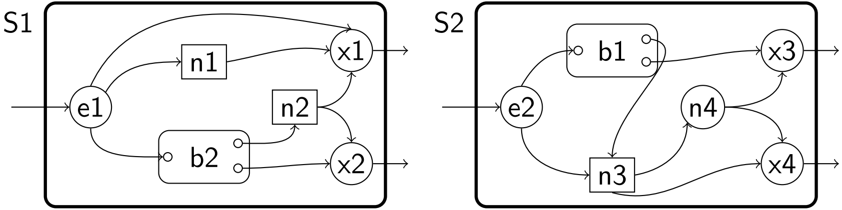


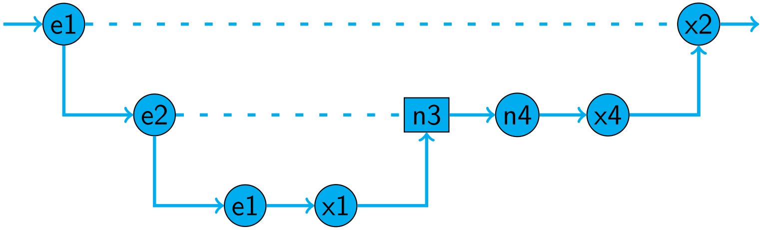
Differences between modalities



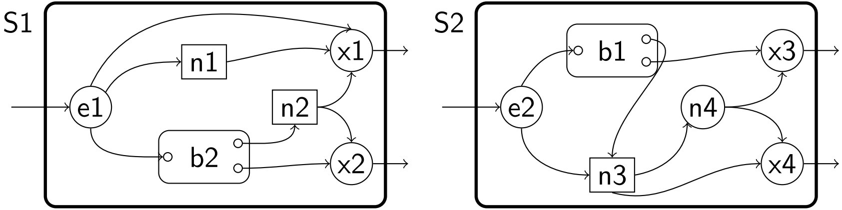


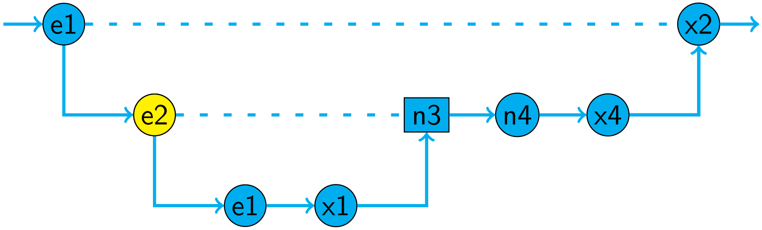
Differences between modalities



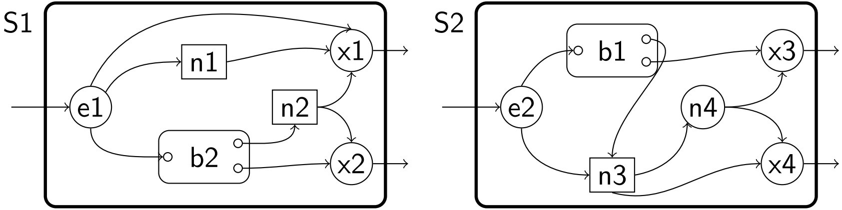


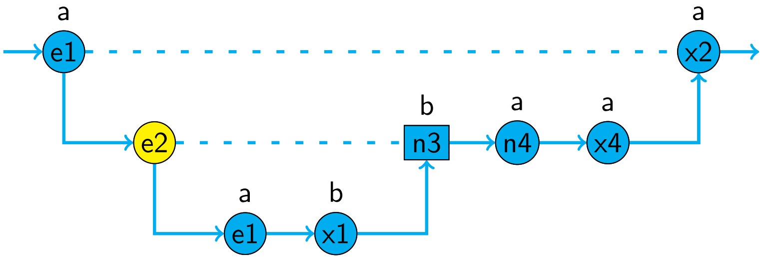
Differences between modalities



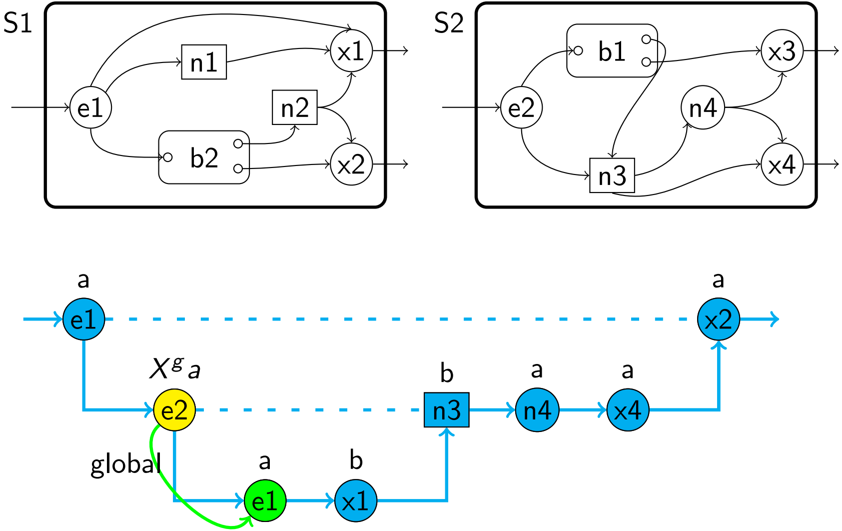


Differences between modalities

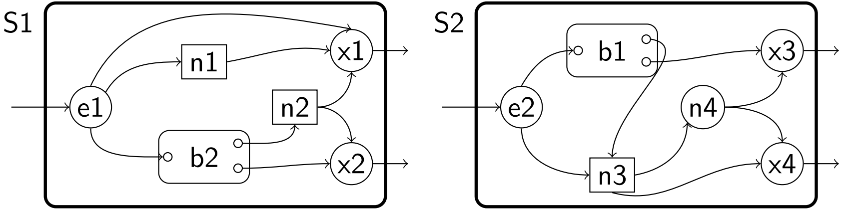


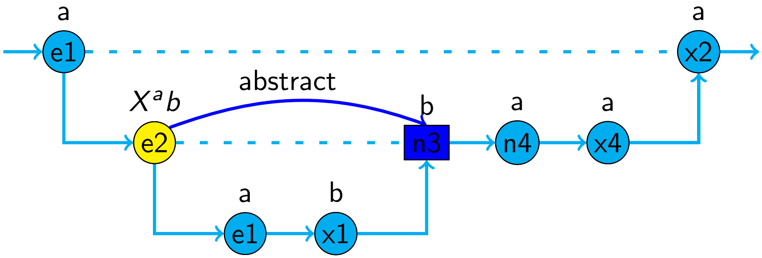


Differences between modalities

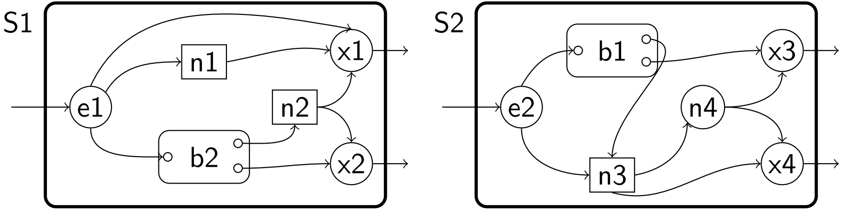


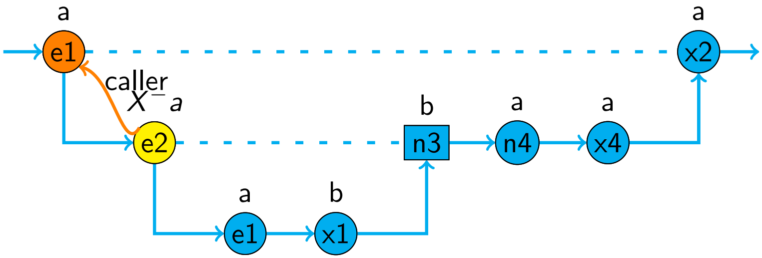
Differences between modalities





Differences between modalities





Our result

- We show that deciding the existence of a winning modular strategy in a CaRet game is 2ExpTime-Complete
- The lower bound derives from the fact that LTL-games for not recursive graph are already 2ExpTime-Hard
- We'll show only that the upper bound of the proposed construction is 2ExpTime

Our Solution

Idea

Reduce the decision problem to a problem of emptiness of nondeterministic automata with parity condition

Let $\langle G, \varphi \rangle$ be a modular CaRet game

- Costructing an equivalent game $\langle G', color \rangle$ with a parity winning condition such that $|G'| = O(2^{|\varphi|})$
- Costructing from $\langle G', color \rangle$ a two-way alternating parity tree automaton A_{win} that accepts a strategy iff corresponds to a winning modular strategiey
- Convert A_{win} in a one-way nondeterministic automaton, take its intersection with the tree automaton that accepts strategy trees and check the emptiness of this resulting automaton

Conclusions

- Deciding existence of modular winning strategies is solved in [Alur, La Torre, Madhusudan, 2003] for:
  - Reachability and safety winning condition *(finite plays)*
  - External winning condition given as deterministic/universal Büchi/coBüchi or LTL formulas *(infinite plays)*
- As far as we know, studies that analyze the problem with non-regular winning condition are not known
- Our work extends the previous results to this class of winning conditions, because CaRet formulas can express also not regular properties

Conclusions

- The implementability of the proposed technique in tools, however, is countered by the high complexity, exponential in the size of the graph and doubly exponential in the size of the formula
- Typically, the analysis of real systems are done for small formulas
- Regarding the structural complexity, deciding the existence of modular strategies for LTL games is NP-complete
- In this paper we have not analyzed the structural complexity and the question remains open