Towards Simpler Theorem-Proving of Graph Grammars with Negative Application Conditions

Guilherme Azzi and Leila Ribeiro

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

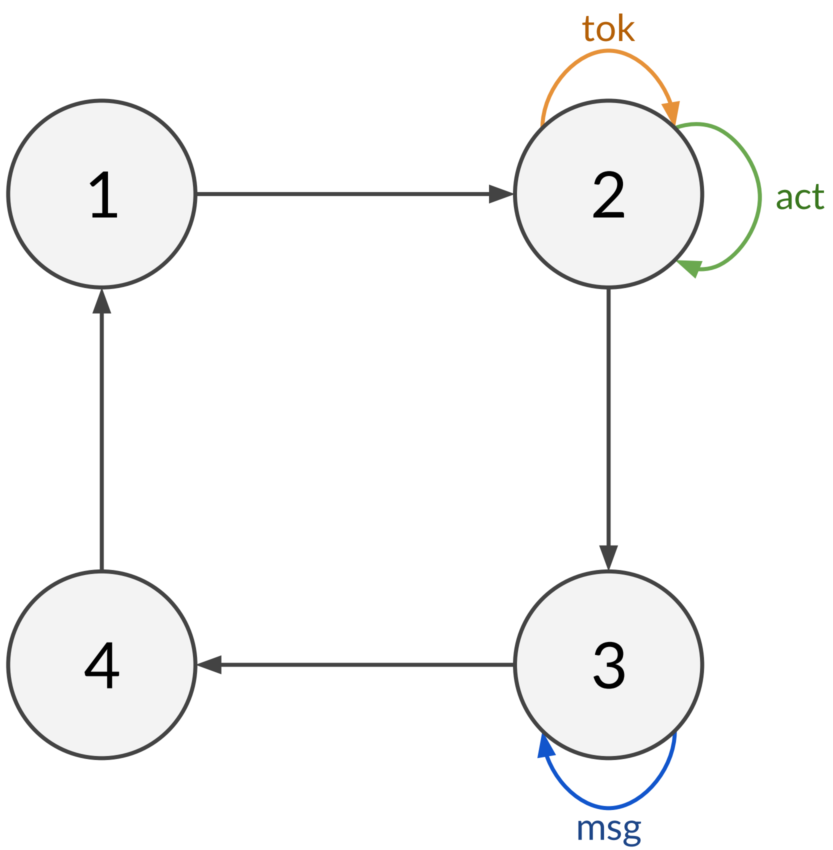
INSTITUTO DE INFORMÁTICA UFRGS

Graph Grammars

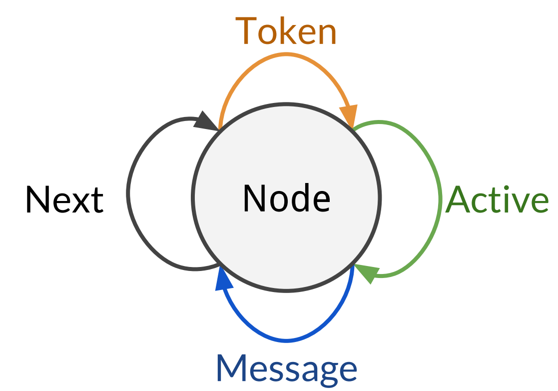
- Graph-based **computational model**
- Similar to **Chomsky-grammars**
- **Typed graphs** as states
- **Transformation rules** as behaviour
- Well-suited for **concurrency**, **model transformation**

Typed Graphs as States

Example: token ring protocol

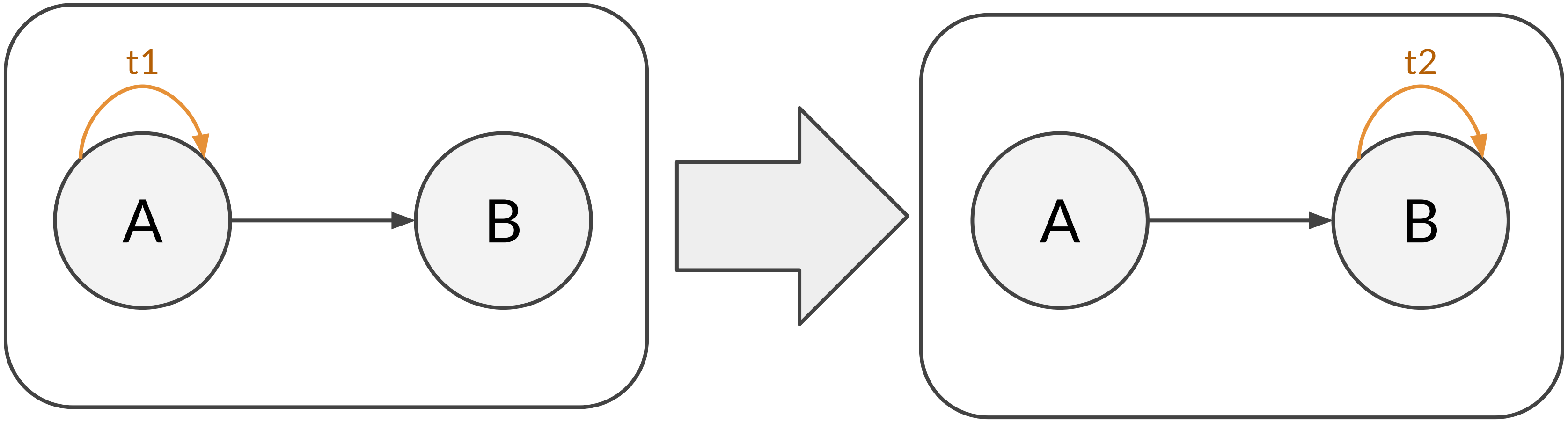


Types:

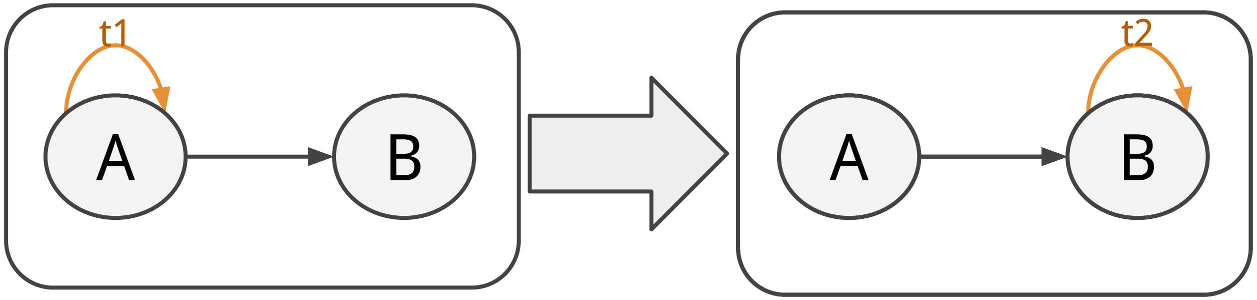


Transformation Rules as Behavior

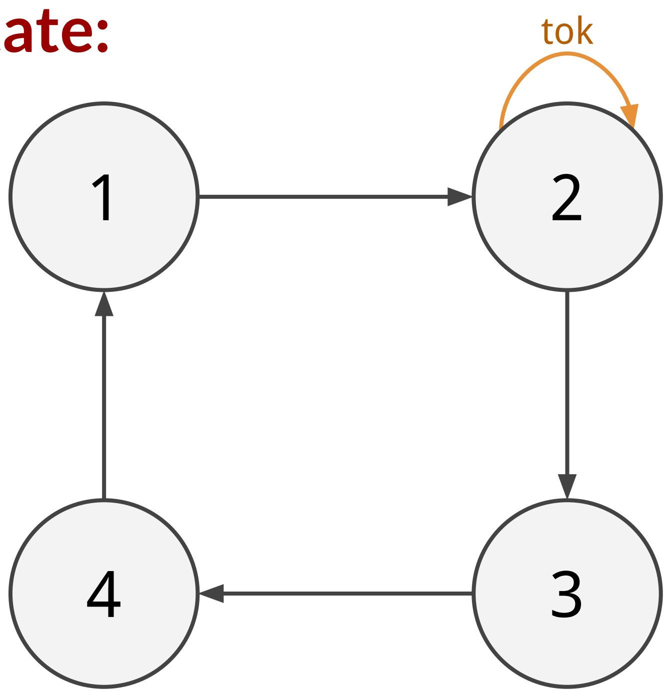
Example: pass the token along



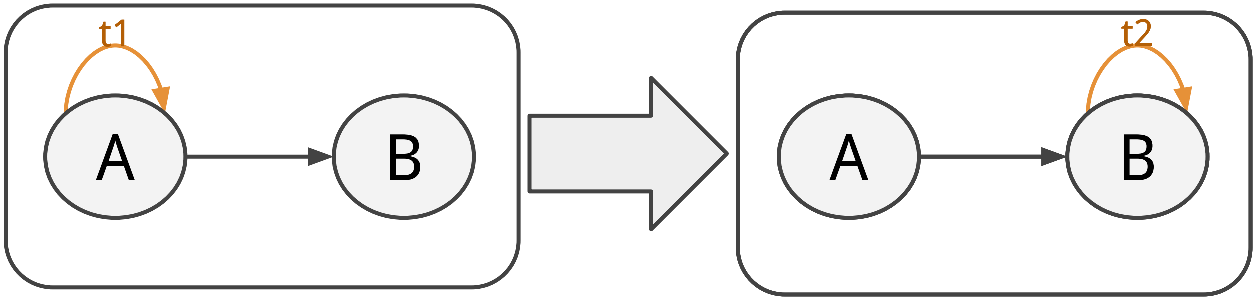
Graph Transformation



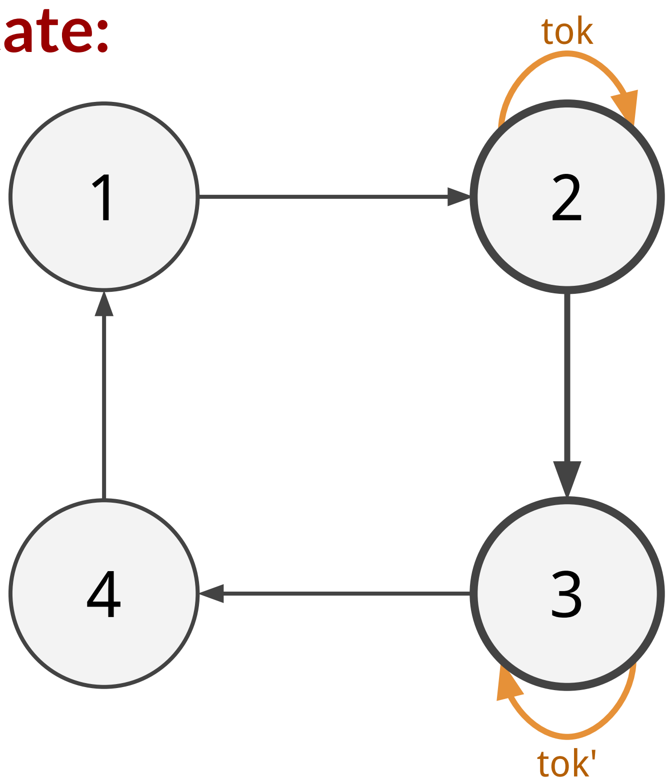
State:



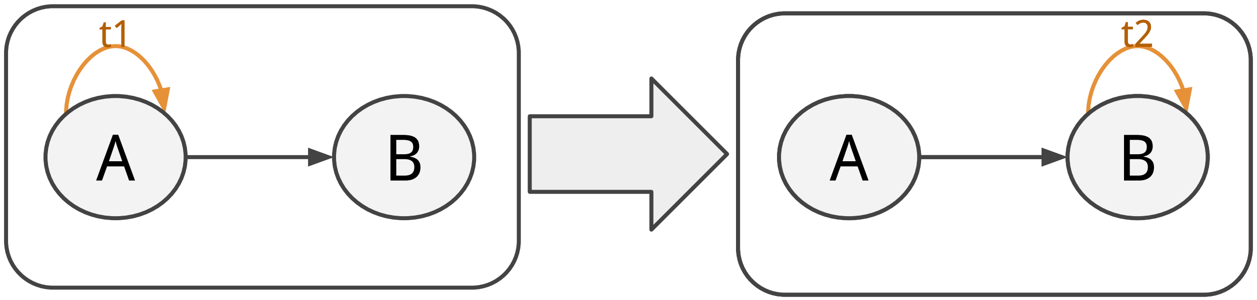
Graph Transformation



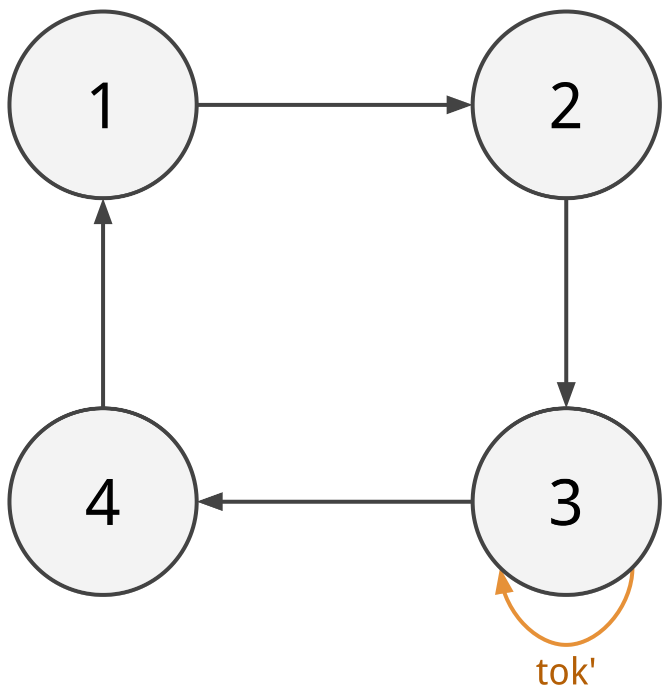
State:



Graph Transformation

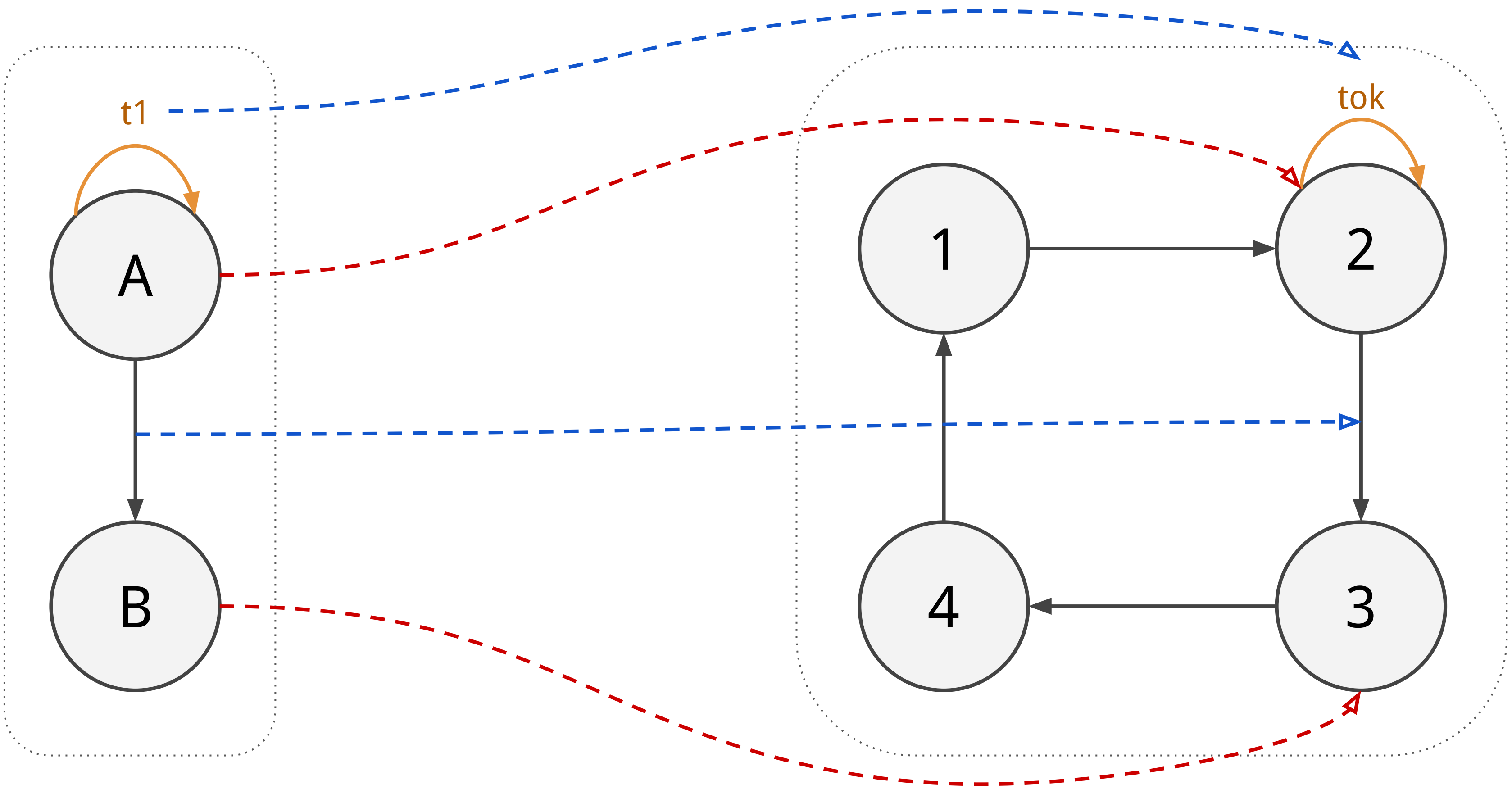


State:

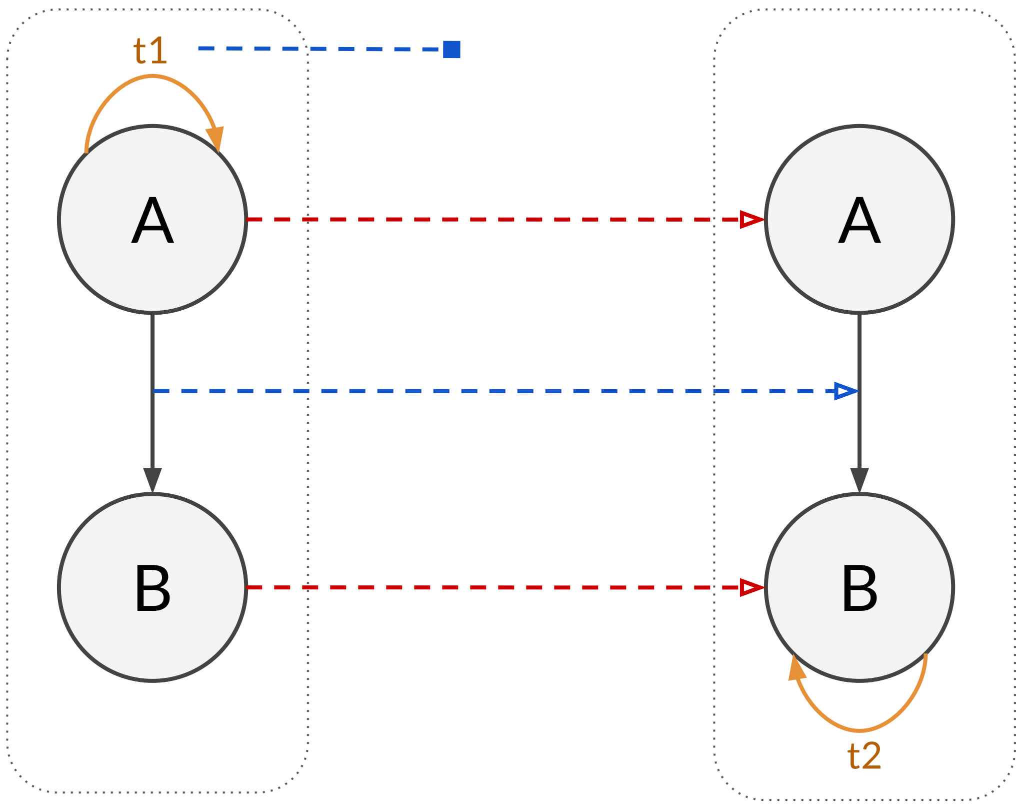


.inf INSTITUTO DE INFORMÁTICA UFRGS

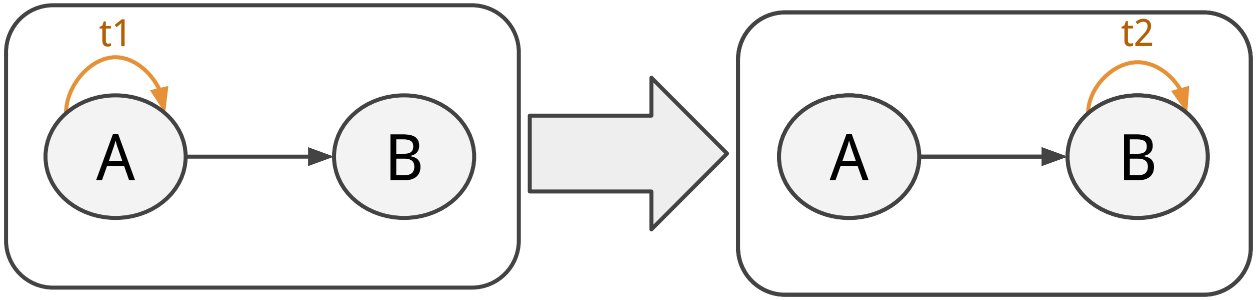
Graph Morphisms as Matches



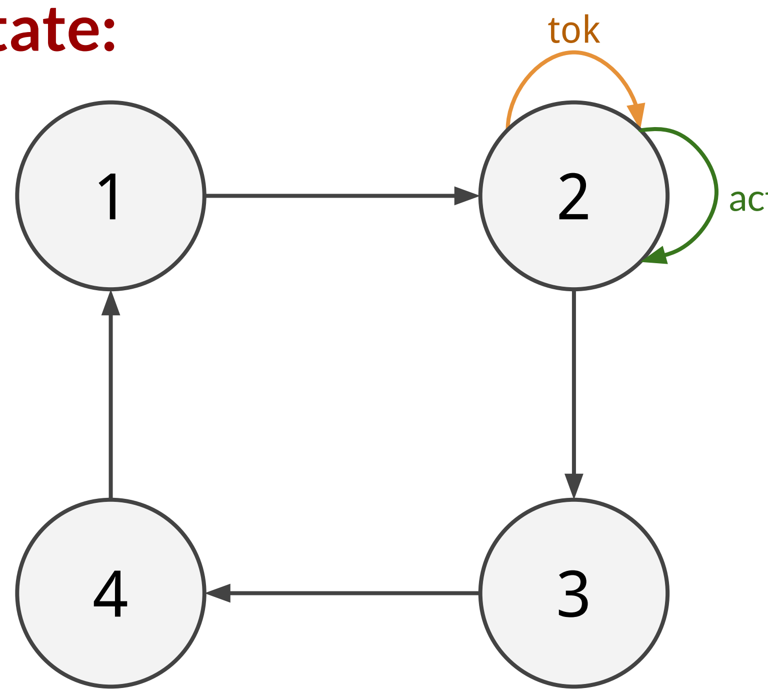
Partial Morphisms as Rules



Forbidden Patterns



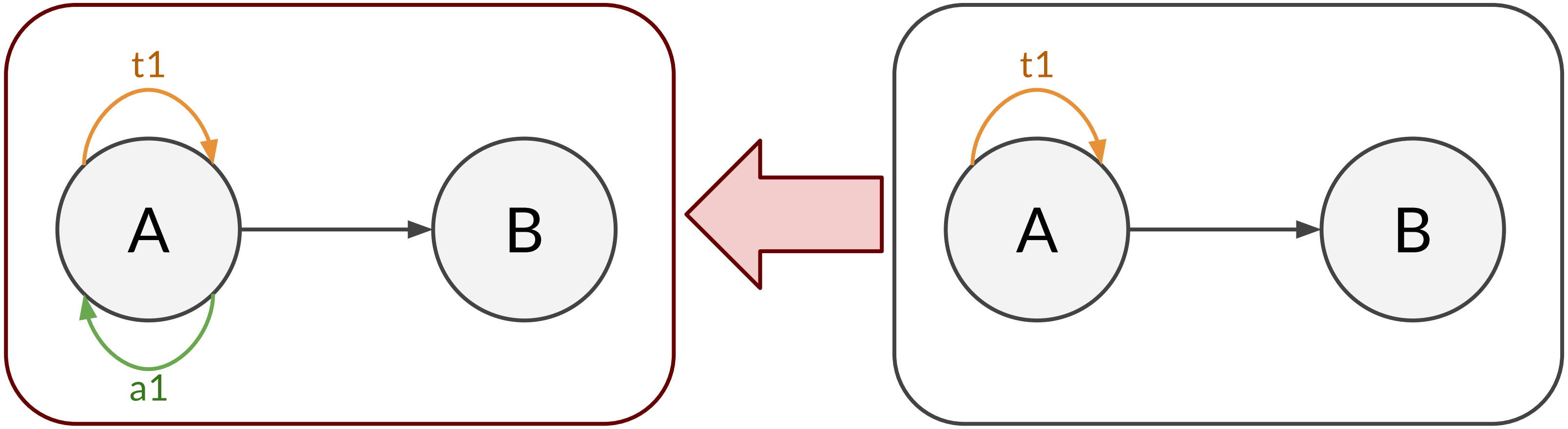
State:



- The token is **given away** while the **node is active**
- Active node should **always hold the token**
- This transformation should be **forbidden**!

Negative Application Conditions (NACs)

- **Negative Application Condition**: forbidden pattern



Graph Grammar

A Graph Grammar is a set of **transformation rules** with NACs along with an **initial graph**.

Verification of Graph Grammars

- Graph grammar as **model** of a system
- Model should have some **desired properties**
  - Always **single token**
  - At most **one active node**, and it **holds the token**
  - Edges of **Token and Active** types **have source = target**
  - **Finite** amount of nodes and edges

Verification of Graph Grammars

- Ensure the grammar has the **desired properties**
- **Theorem Proving**: very general guarantees, but high effort
- **Goal:** reduce the effort necessary for theorem-proving

Theorem Proving with Event-B

- Framework for **modeling and verifying** systems
- Based on **first-order logic** with **set theory**
- Specification of **state and transitions**

Event-B Model

Context	Machine
• Abstract sets	
• Constants	• Variables
• Axioms	• Invariants
	• Events

Theorem Proving with Event-B

- Model is correct if the **events preserve the invariants**
- **Tool support** with Rodin (Eclipse-based)
- Generates **proof obligations** to guarantee invariants
- Assisted proofs

Encoding Graph Grammars into Event-B

- **Type graph** is encoded as (V_T, E_T, src_T, tgt_T)
- **Typed Graphs** are encoded as $(V_G, E_G, src_G, tgt_G, tV_G, tE_G)$
- **Graph morphisms** encoded as (f_V, f_E)
- **Rules** encoded as their graphs and morphisms

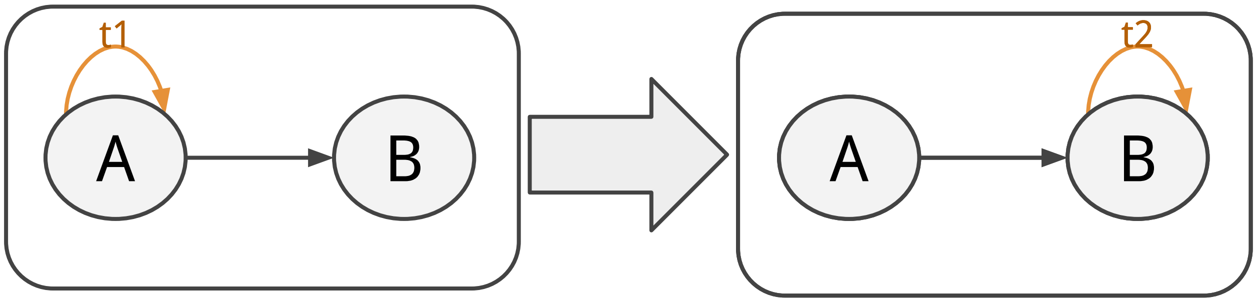
Encoding Rule Application with Event-B

- Rule application encoded as **event**
- **Parameters:** the match (m_V, m_E)
- **Guards:** match is a morphism, NACs are satisfied
- **Actions:**
  - Remove and create deleted nodes/edges
  - Update functions that encode type, source and target

Encoding NACs

- Initial formulation:
  - There is **no morphism** from the NAC graph to the instance graph **that agrees with the match**
- Element-based formulation:
  - There are **no non-matched nodes/edges** in the instance graph **with the forbidden types, sources and targets**
- Both are **equivalent** (Cavalheiro et al.)
- Second one is **more convenient** for proofs

Simplified Proof



- Proposition: rule preserves invariant that there is a **single active node**
- Proof outline:
  - **Assume invariants and guards hold** in original state
  - Since there is no morphism from NAC to state, **A isn't active**
  - Since there is only one node with a token, **only A holds a token**
  - Since all active nodes must hold a token, **no node is active**
  - The **rule doesn't activate** any nodes
  - Thus, **no node is active** after the transformation

Alternative Encoding of Rules

- **Same strategy** used for rules
- Original encoding: there **exists a morphism** from the LHS to the instance graph
- Alternative encoding: there **exist nodes/edges** in the instance graph **with expected types, sources and targets**

Comparison of Encodings

- **Token ring protocol** was formalized and verified
  - In original encoding, by Cavalheiro et al.
  - In alternative encoding
- **All 8 invariants** were proven
- Number and difficulty of **proof obligations** compared
  - **Automatic**
  - **Easy**: clicking on symbols and using provers
  - **Manual**: complex guidance, requires insight

Results

- Translation POs show that the encoding is “consistent”
- Application POs show that the invariants are respected

Encoding	Translation POs				Application POs			
	POs	Auto	Easy	Manual	POs	Auto	Easy	Manual
Original	64	60	4	0	56	14	18	24
Modified	91	87	0	4	44	15	5	24
Difference	+27	+27	-4	+4	-12	+1	-13	0

- **Little impact** on the difficulty
- Downside: **more premises** during proofs
- *Possible* upside: **closer to intuition**

Major Difficulties

- **Preservation of types** is nontrivial to prove
- **Locality** of changes is hard to explore
- **Recurring patterns** are hard to abstract

Research Directions

- **Formal definition** of translation
- Proof of **correctness**
- **Attributed graphs**
- Library of **lemmas or proof tactics**
- Encoding in **other theorem prover**
- **Separation Logic**

.inf INSTITUTO DE INFORMÁTICA UFRGS

Conclusions

- Alternative encoding **doesn’t lead to simpler proofs**
- *Possibly* closer to intuition
- Some **major roadblocks identified**

Acknowledgements

CAPES

CNPq
Conselho Nacional de Desenvolvimento Científico e Tecnológico

VERITES
CERTIFIED

Thank you!

Questions?

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

.inf INSTITUTO DE INFORMÁTICA UFRGS

Encoding the Type Graph

```
sets
  VertT
  EdgeT
```

Sets of node types and edge types

```
constants
  Node
  Nxt Tok Msg Act
  sourceT
  targetT
```

Node/edge types

Define the types of sources/targets

```
axioms
  VertT = {Node}
  EdgeT = {Nxt, Tok, Msg, Act}
  sourceT : EdgeT → NodeT
  sourceT = {Nxt↦Node, Tok↦Node, Msg↦Node, Act↦Node}
  // ...
```

Defining the State

```
variables
  VertG
  EdgeG
  sourceG targetG
  tG_E tG_V
```

Sets of nodes/edges

Source/target of the edges

Types of nodes/edges

```
invariants
  VertG ∈ ℙ(ℕ)
  EdgeG ∈ ℙ(ℕ)
  sourceG : EdgeG → VertG
  targetG : EdgeG → VertG
  tG_V : VertG → VertT
  tG_E : EdgeG → EdgeT
  // ...
```

Encoding the Rules

```
sets
  VertL1 EdgeL1
  VertR1 EdgeR1
```

Left- and right-hand sides

```
constants
  N11 N12
  Tok11 Nxt11
```

Nodes/edges of the LHS

```
  sourceL1 targetL1 tL1_V tL1_E
```

Types/source/target of the LHS

```
  N13 N14
  Tok12 Nxt12 Msg11
  sourceR1 targetR1 tR1_V tR1_E
```

RHS

```
axioms
  // ...
```

Encoding the Application

```
event
  any
    mV mE
    newAct11 newMsg11
  where
    // ...
  then
    EdgeG := EdgeG U {newAct11, newMsg11}
    sourceG := {newAct11 mV(N11)
```