Transformations with Triple Graph Grammars with Non-terminal Symbols

William da Silva, Max Bureck, Ina Schieferdecker, and Christian Hein

Fraunhofer Fokus, Berlin, Germany
william.bombardelli.da.silva@fokus.fraunhofer.de
Technische Universität Berlin, Berlin, Germany

December 5, 2018

Organization

Introduction

Background

- Model-driven software development as a technique to enhance quality of software

Background

- Model-driven software development as a technique to enhance quality of software
- Models as formal specifications of safety-critical systems

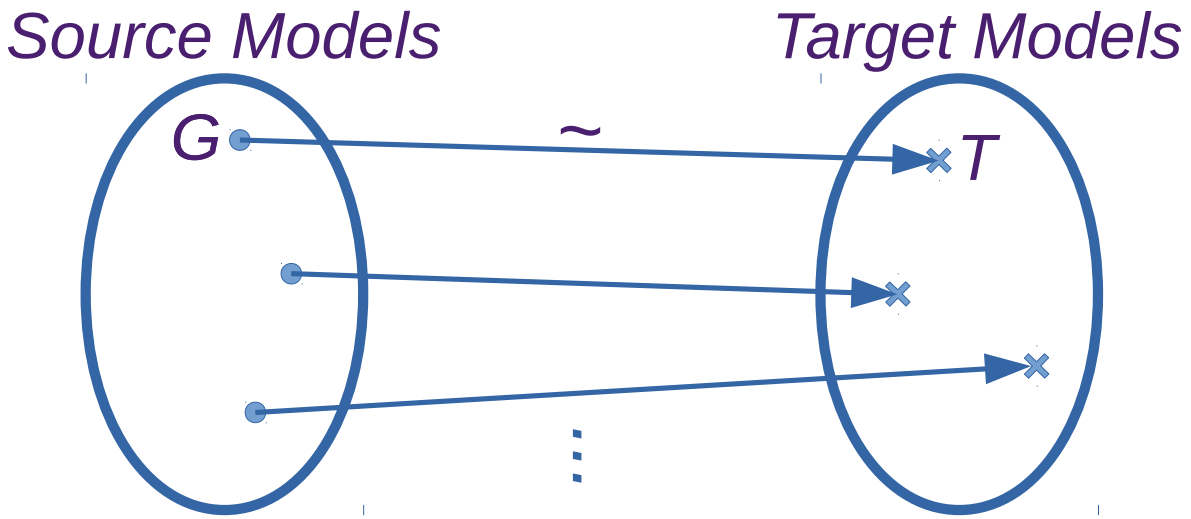
Background

- Model-driven software development as a technique to enhance quality of software
- Models as formal specifications of safety-critical systems
- Transformation between models (e.g. from a formal specification to high-level source-code and vice-versa)

Background

- Model-driven software development as a technique to enhance quality of software
- Models as formal specifications of safety-critical systems
- Transformation between models (e.g. from a formal specification to high-level source-code and vice-versa)
- **Goal:** Comprehensible and reliable transformations
  - Efficient representation of abstract concepts
  - Small size

The Model Transformation Problem



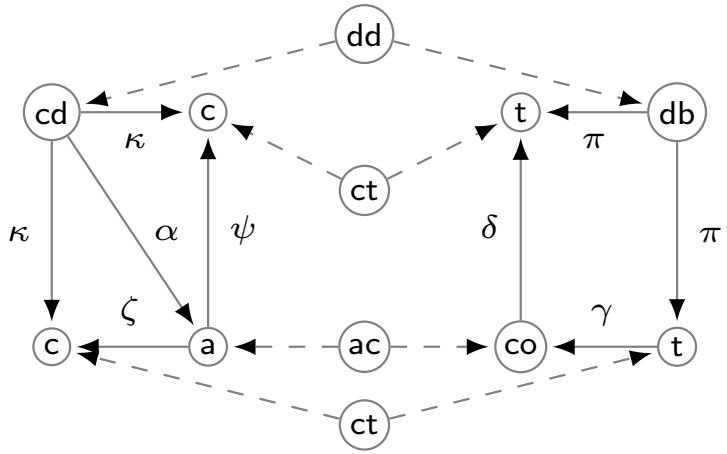
- $G \sim T$ iff G is correctly transformed into T
- $\sim$ is the *correctly-transformed relation* between source and target models
- **Batch forward transformation:**
  Given G, find a T, such that $G \sim T$

The Triple Graph Grammar Approach

- Models are graphs

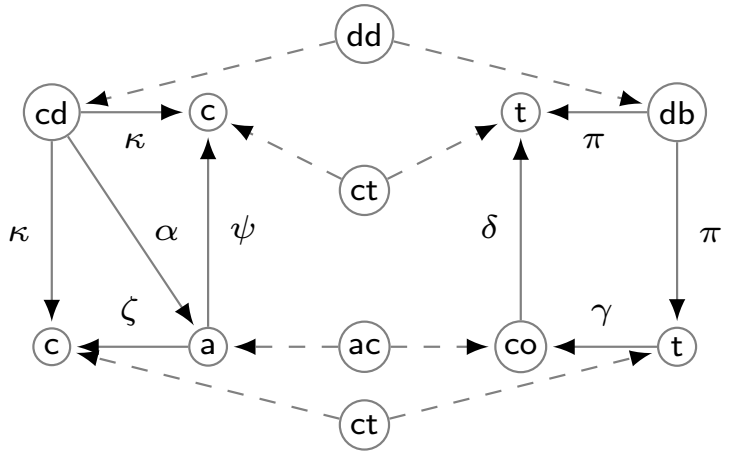
The Triple Graph Grammar Approach

- Models are graphs
- Two correctly-transformed graphs G and T are in a triple graph $G \leftarrow C \rightarrow T$



The Triple Graph Grammar Approach

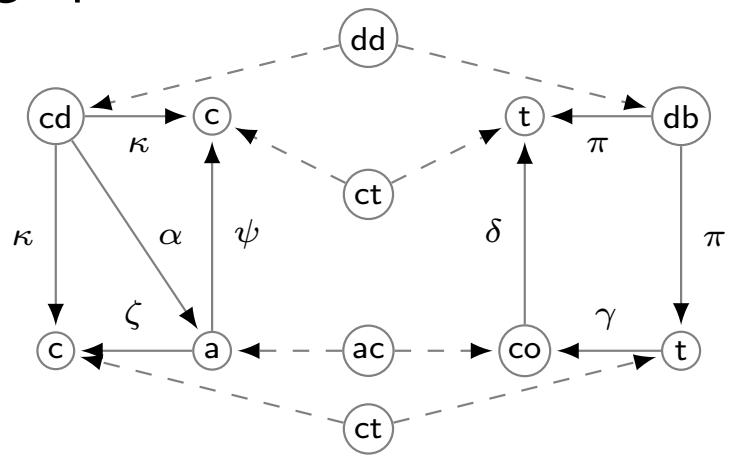
- Models are graphs
- Two correctly-transformed graphs G and T are in a triple graph $G \leftarrow C \rightarrow T$



- A triple graph grammar TGG is a generator of a set of triple graphs $L(TGG)$

The Triple Graph Grammar Approach

- Models are graphs
- Two correctly-transformed graphs G and T are in a triple graph $G \leftarrow C \rightarrow T$

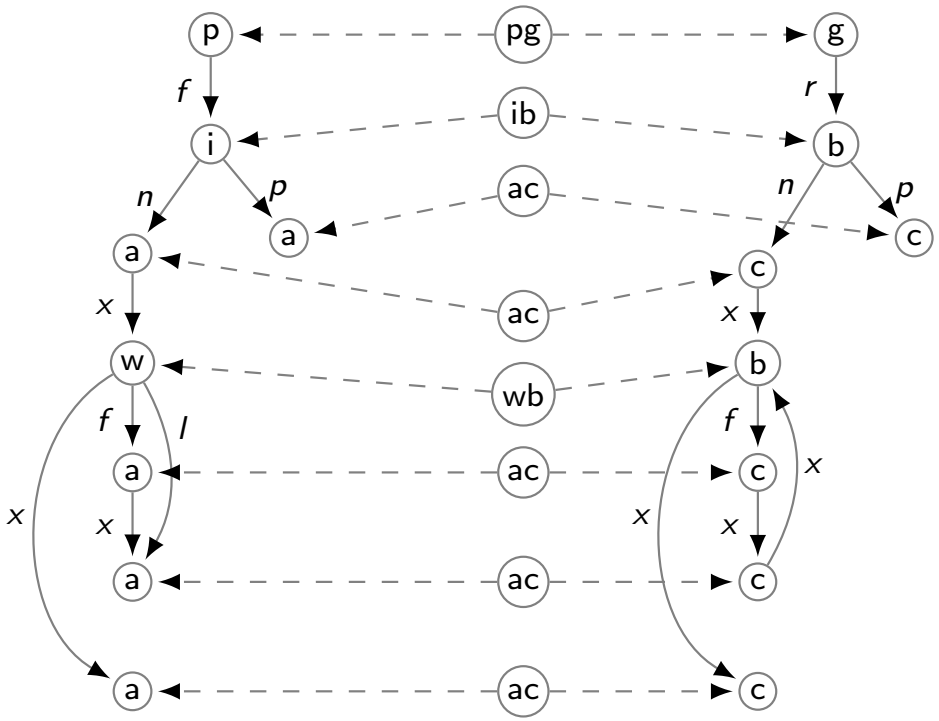


- A triple graph grammar TGG is a generator of a set of triple graphs $L(TGG)$
- The correctly-transformed relation $\sim$ between graphs is described in terms of a triple graph grammar TGG
  - $G \sim T$ iff $(G \leftarrow C \rightarrow T) \in L(TGG)$

TGG – An Example

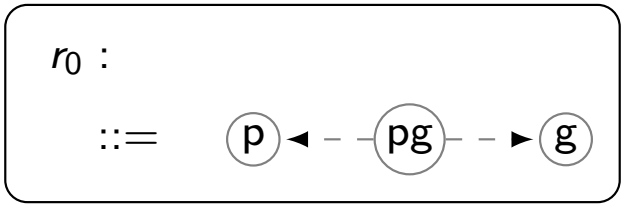
- *Pseudocode* to *Controlflow*

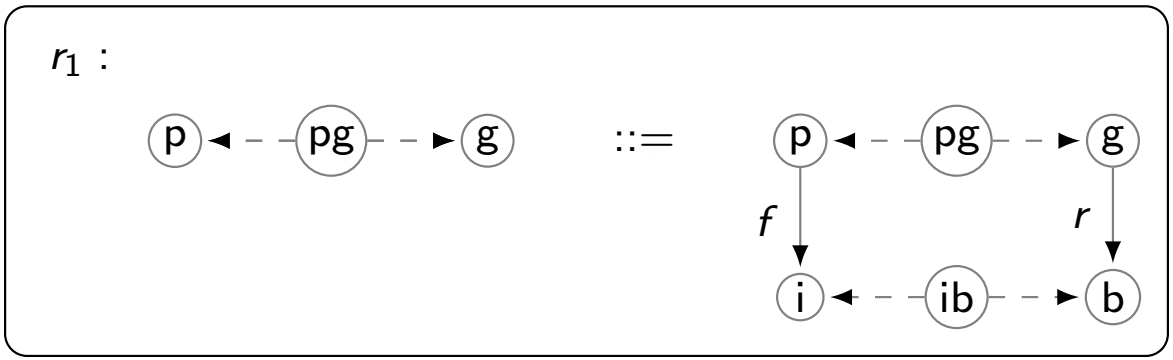
```
program main(n)
if n < 0 then
    return Nothing
else
    f ← 1
    while n > 0 do
        f ← f ∗ n
        n ← n − 1
    end while
    return Just f
end if
```

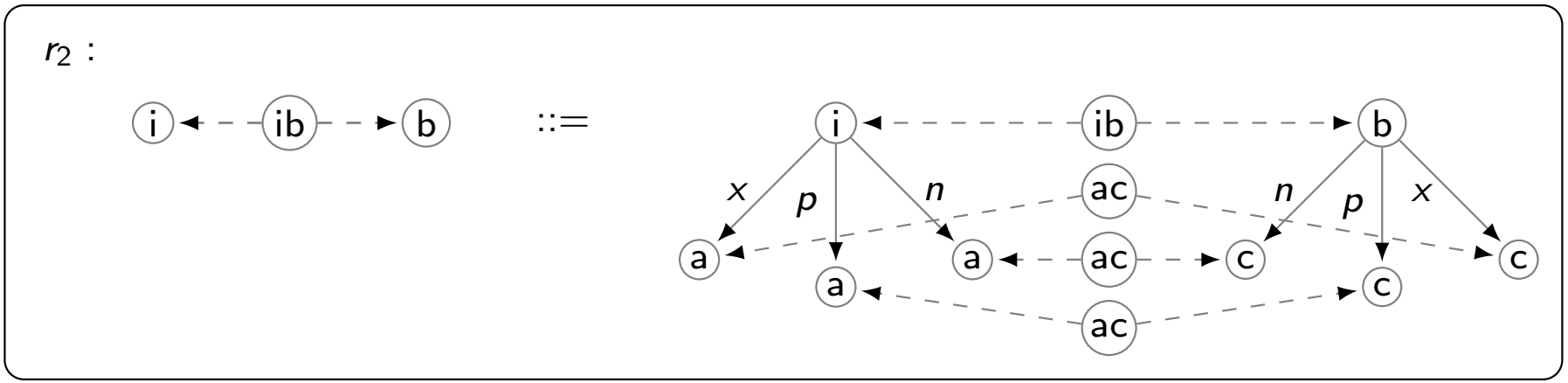


TGG – An Example

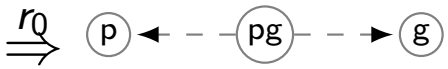
- *Pseudocode* to *Controlflow*



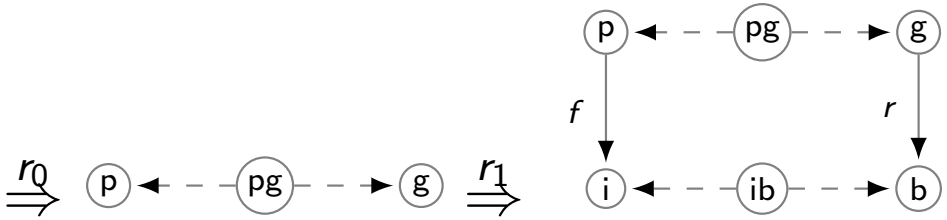




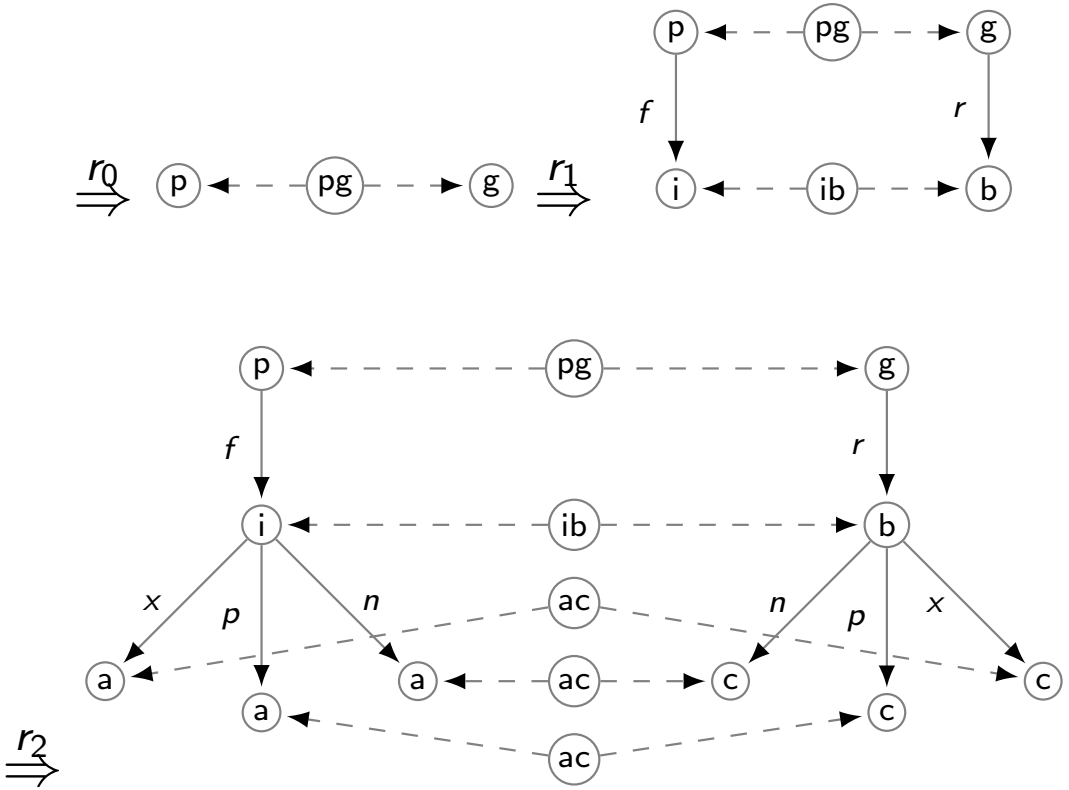
TGG – Derivation



TGG – Derivation



TGG – Derivation



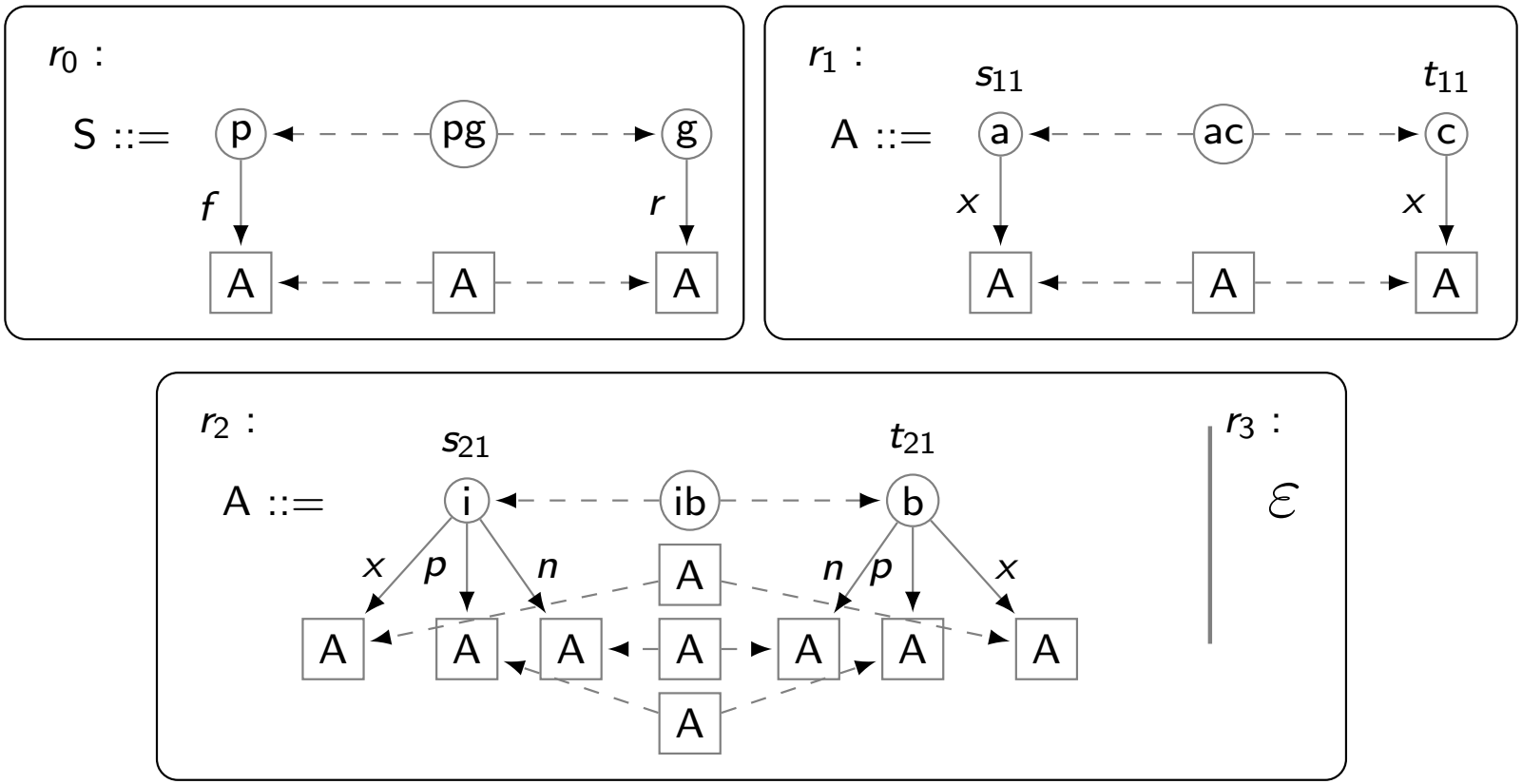
Triple Graph Grammars with Non-terminal Symbols

Our Contribution – NCE TGG

- New formalism: NCE TGG
  - *Graph Grammar with Neighborhood-controlled Embedding* (NCE) [Janssens and Rozenberg(1982)]
  - *Triple Graph Grammar* (TGG) [Schürr(1994)]
- Non-terminal symbols
- Context-free

NCE TGG – An Example

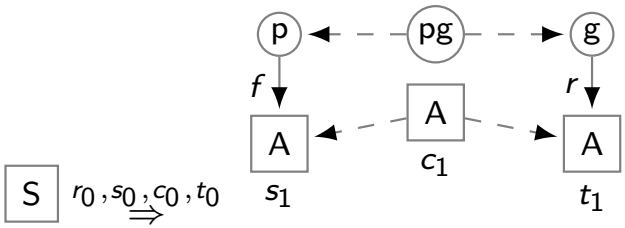
- *Pseudocode* to *Controlflow*



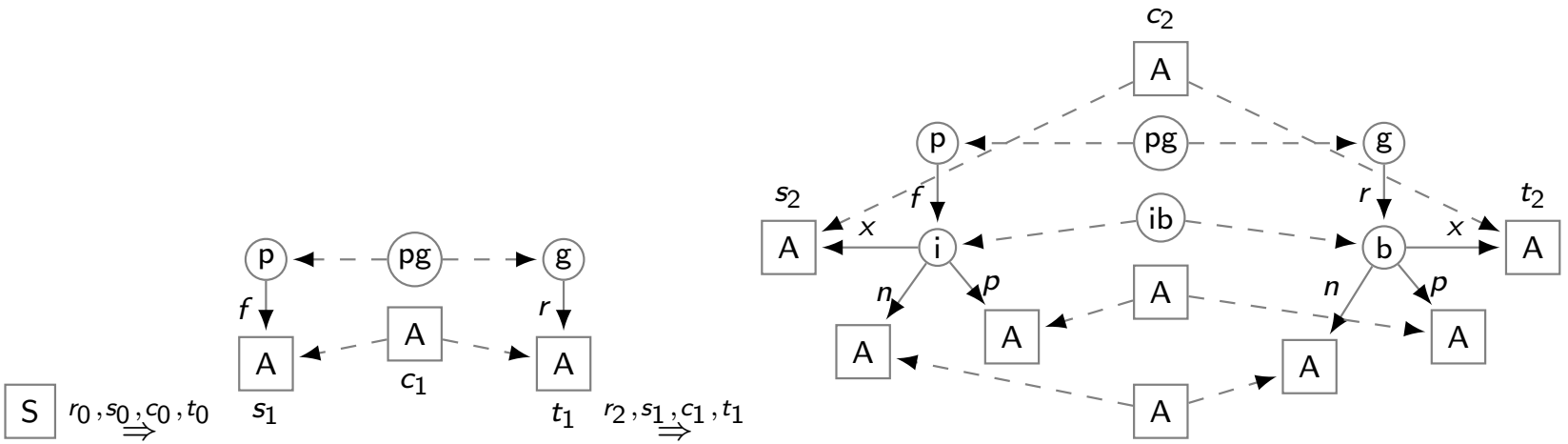
Pseudocode to Controlflow – Derivation

S

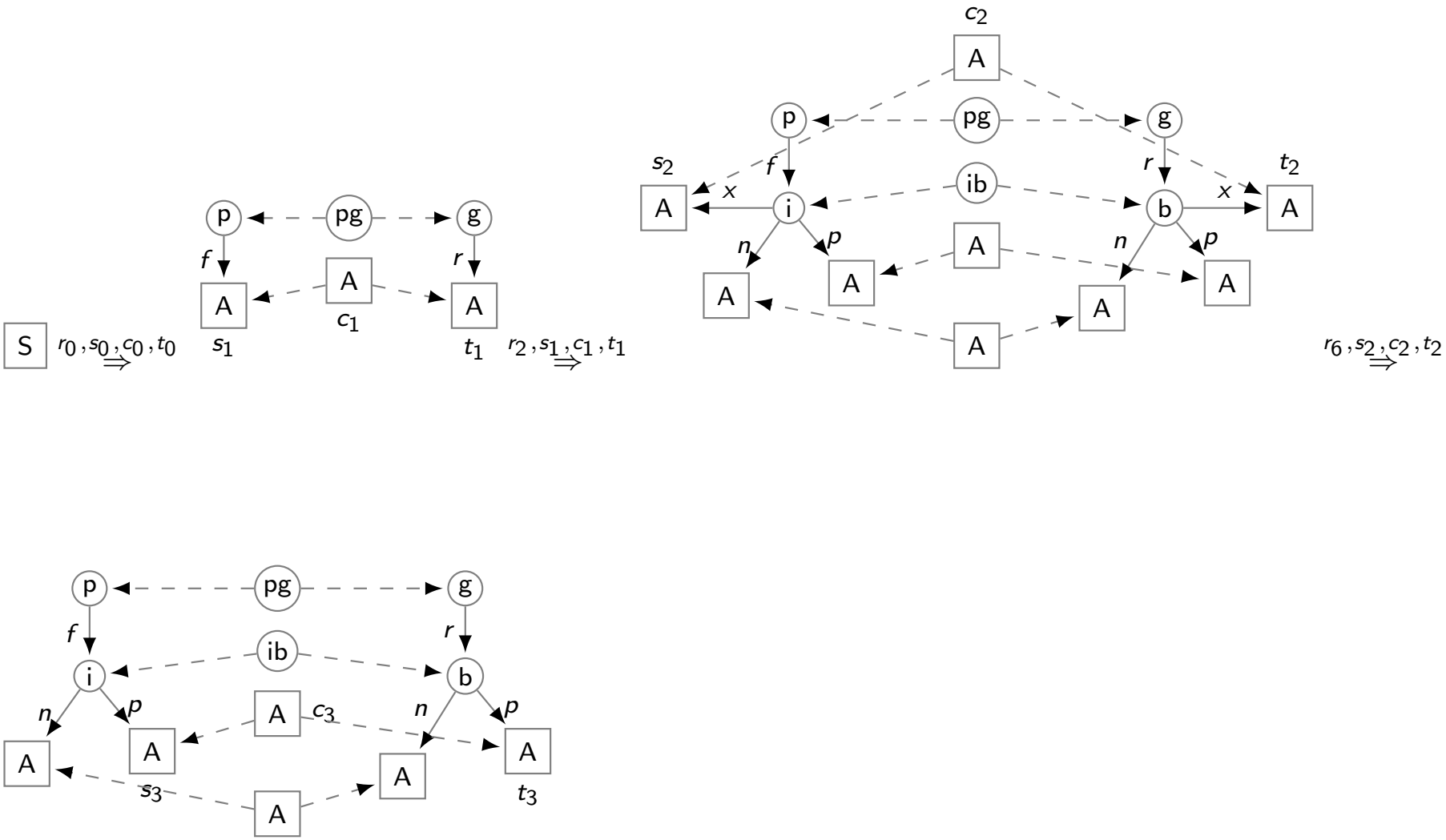
Pseudocode to Controlflow – Derivation



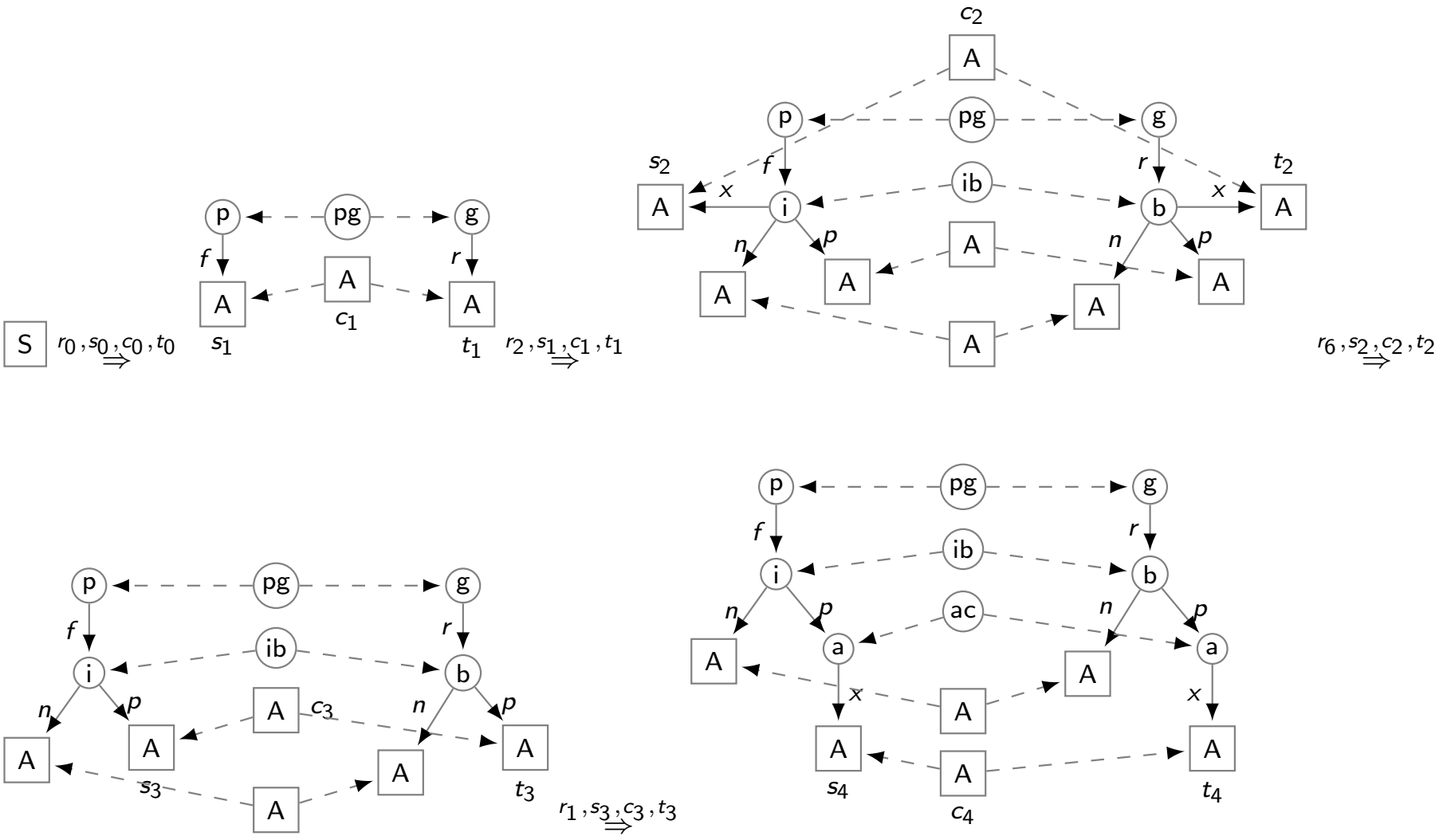
Pseudocode to Controlflow – Derivation



Pseudocode to Controlflow – Derivation

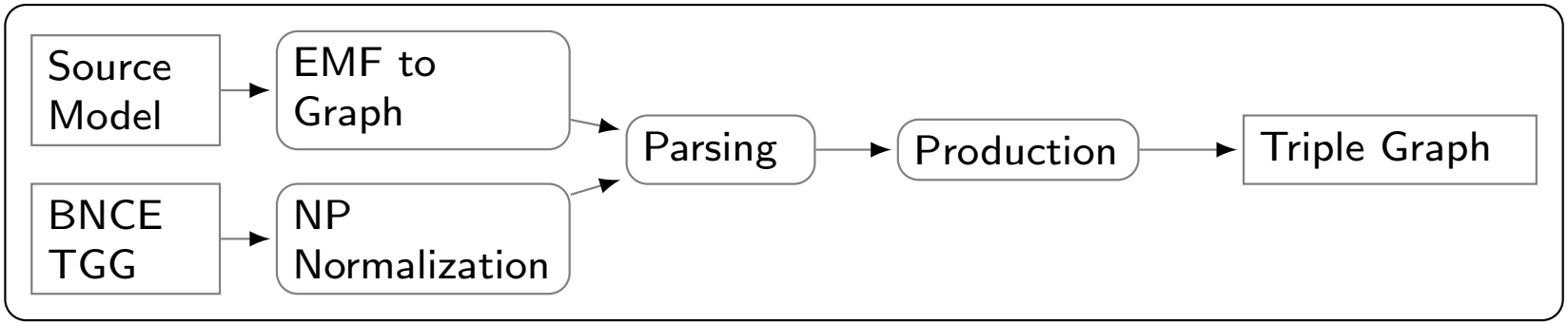


Pseudocode to Controlflow – Derivation

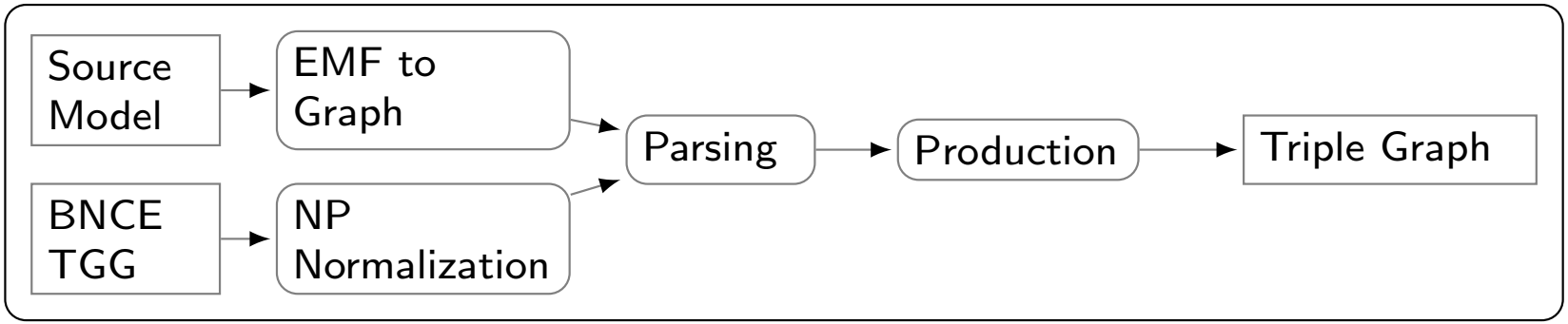


Transformation

Transformation



Transformation



- Bottom-up parser, analogous to CYK, from [Rozenberg and Welzl(1986)]
- For degree-bounded connected graphs: Polynomial worst-case time complexity, but not linear
- Performance not practicable (yet)

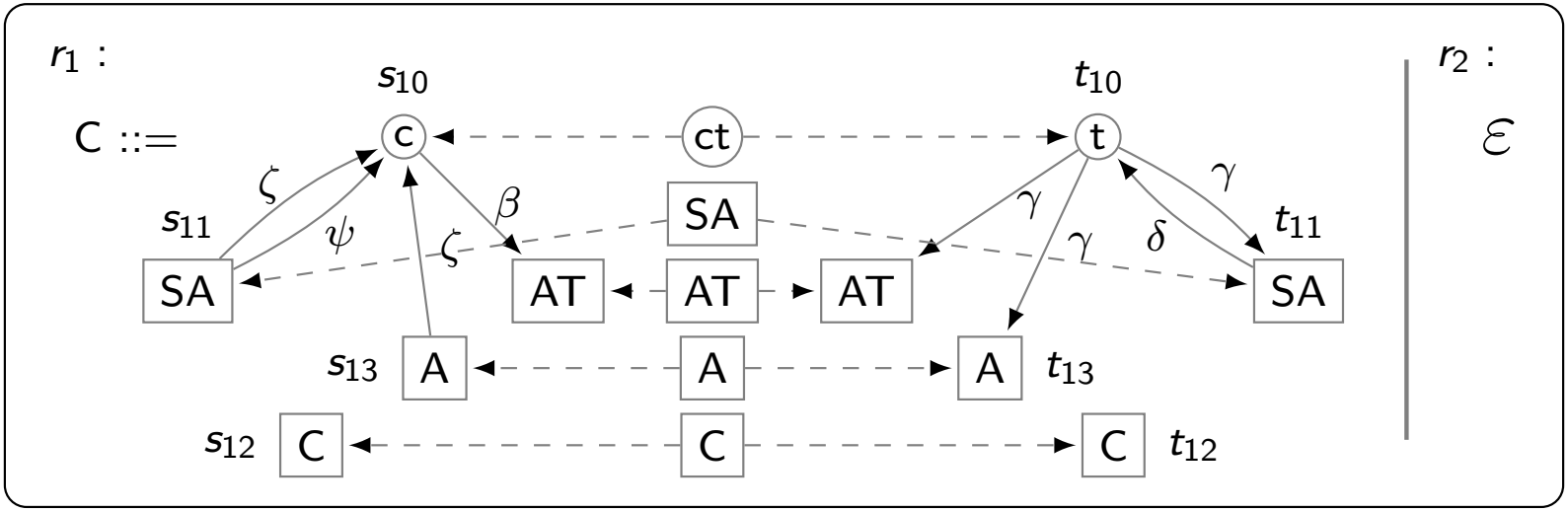
Triple Graph Grammars with Application Conditions

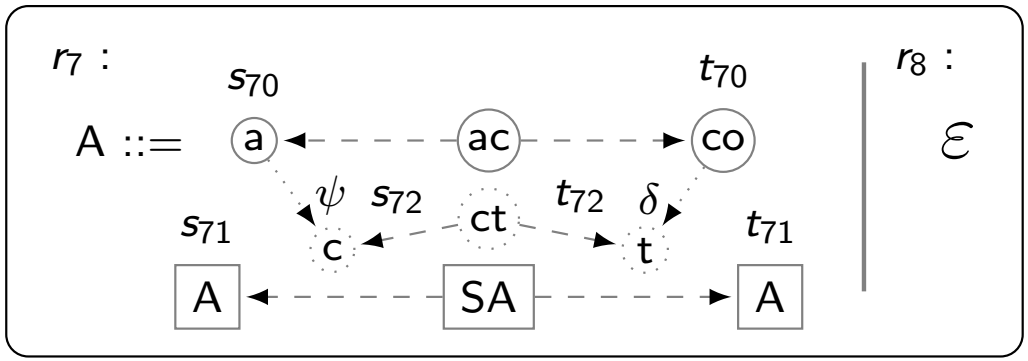
PAC NCE TGG

- New formalism: NCE TGG with Positive Application Conditions (PAC NCE TGG)
  - PAC vertices are created with *rule applications*.
  - PAC vertices are removed with *resolutions*.
  - Allows some sort of context
  - Enhances grammar's generative power

PAC NCE TGG – An example

- *Class* to *Database*





Evaluation

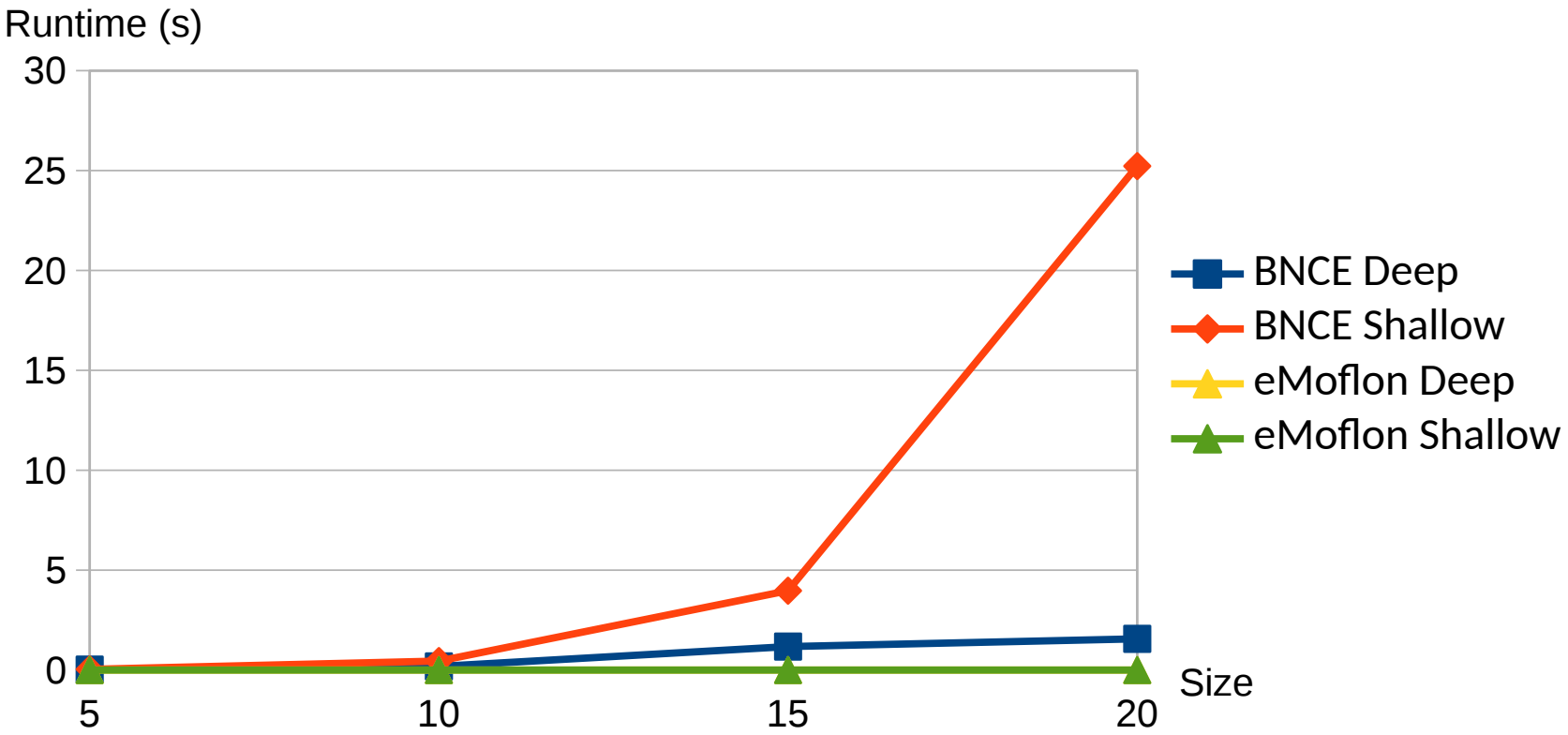
Usability Evaluation

	Standard TGG		BNCE TGG	
Transformation	Rules	Elements	Rules	Elements
Pseudocode2Controlflow	47	1085	**7**	**185**
BTree2XBTree	**4**	**50**	5	80
Star2Wheel	-	-	**6**	**89**
Class2Database	**5**	**80**	9	117
Statemachine2Petrinet	**5**	**114**	7	131
Total	61	1329	**28**	**513**
Average	15.25	332.25	**7**	**128.25**
Median	**5**	**97**	7	124

Table: Results of the usability evaluation of the BNCE TGG formalism in comparison with the standard TGG for the model transformation problem

Performance Evaluation

- *Pseudocode* to *Controlflow*



Conclusion

Conclusion

- New context-free TGG formalism
  - Outperforms standard TGG in 2 evaluated cases and in average
  - Special potential for code-generation
  - Extension with positive application conditions

Conclusion

- New context-free TGG formalism
  - Outperforms standard TGG in 2 evaluated cases and in average
  - Special potential for code-generation
  - Extension with positive application conditions
- Future Work
  - Efficient transformer: Top-down parser
  - Broader evaluation including empirical assessment with engineers and performance reports
  - Model synchronization

References

References

Dirk Janssens and Grzegorz Rozenberg.

Graph grammars with neighbourhood-controlled embedding.
Theoretical Computer Science, 21(1):55–74, 1982.

Grzegorz Rozenberg and Emo Welzl.

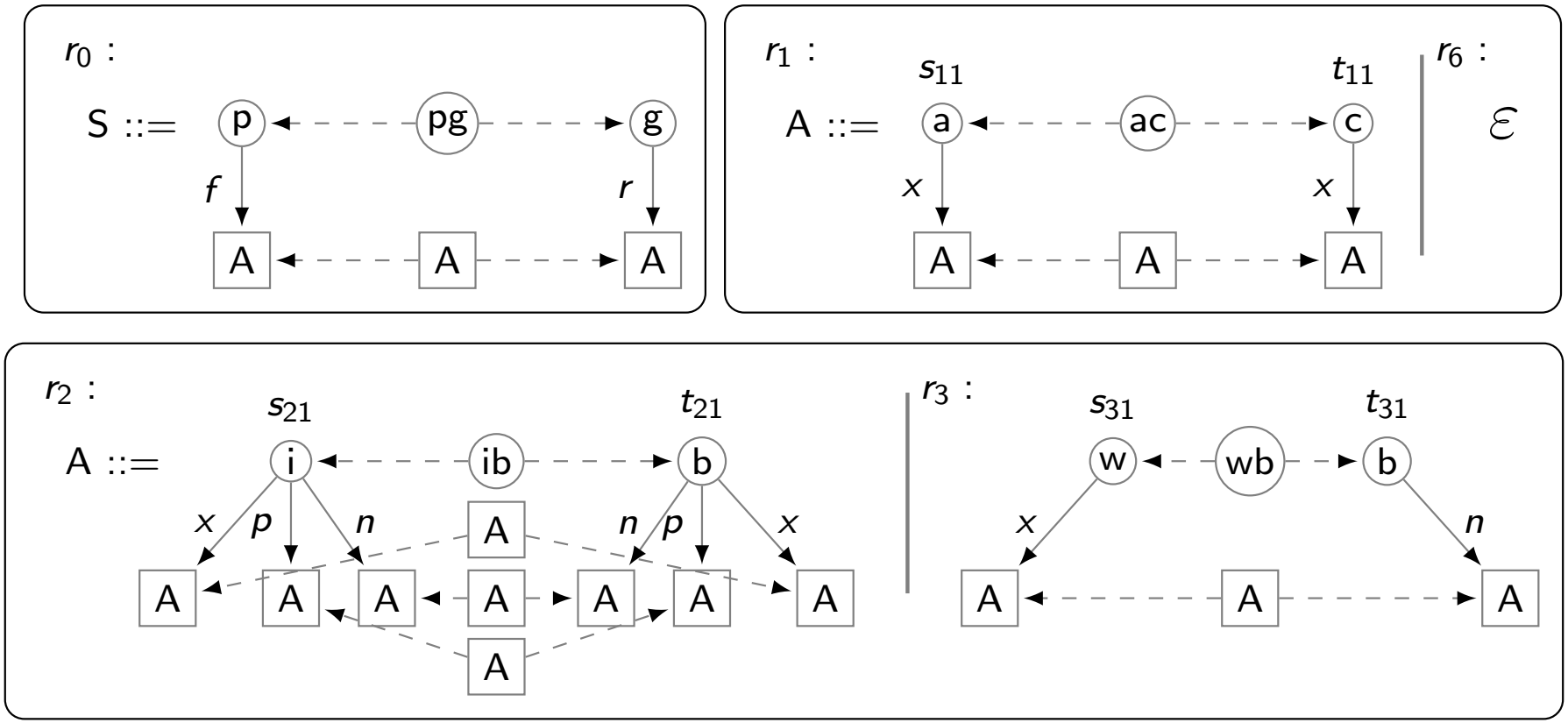
Boundary NLC graph grammarsbasic definitions, normal forms, and complexity.
Information and Control, 69(1-3):136–167, 1986.

Andy Schürr.

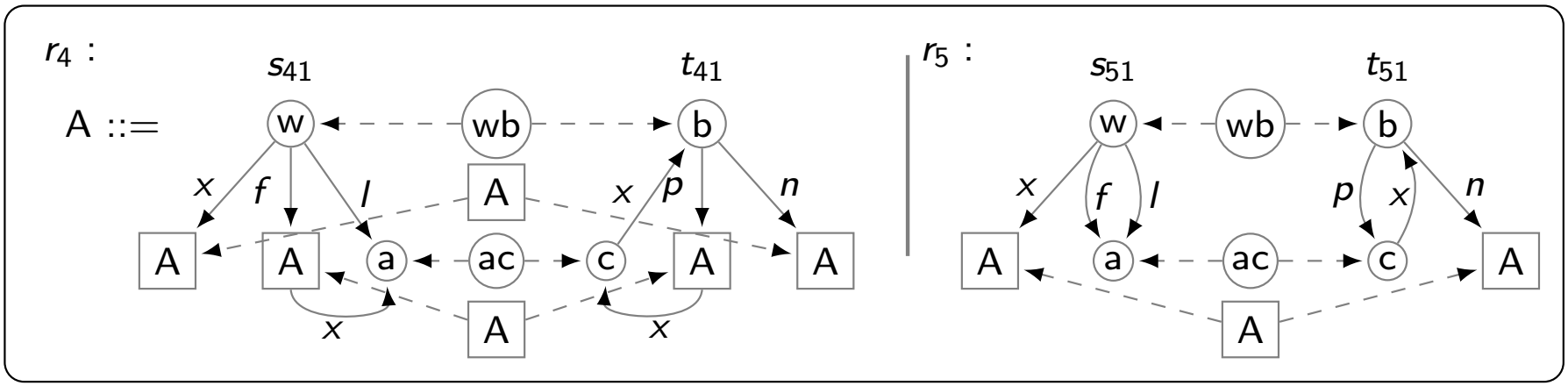
Specification of graph translators with triple graph grammars.
In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 151–163. Springer, 1994.

Appendix

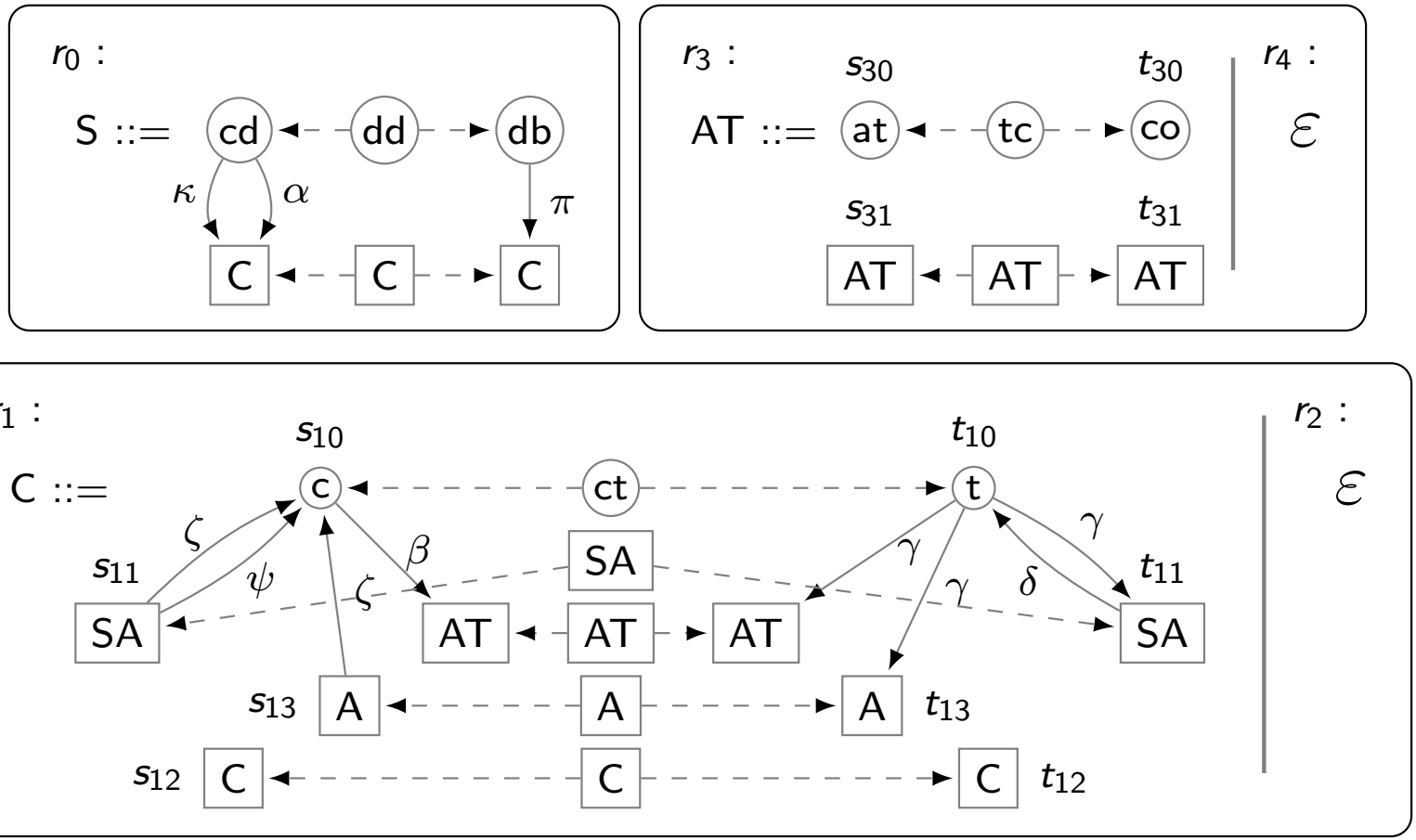
Pseudocode to Controlflow – Full



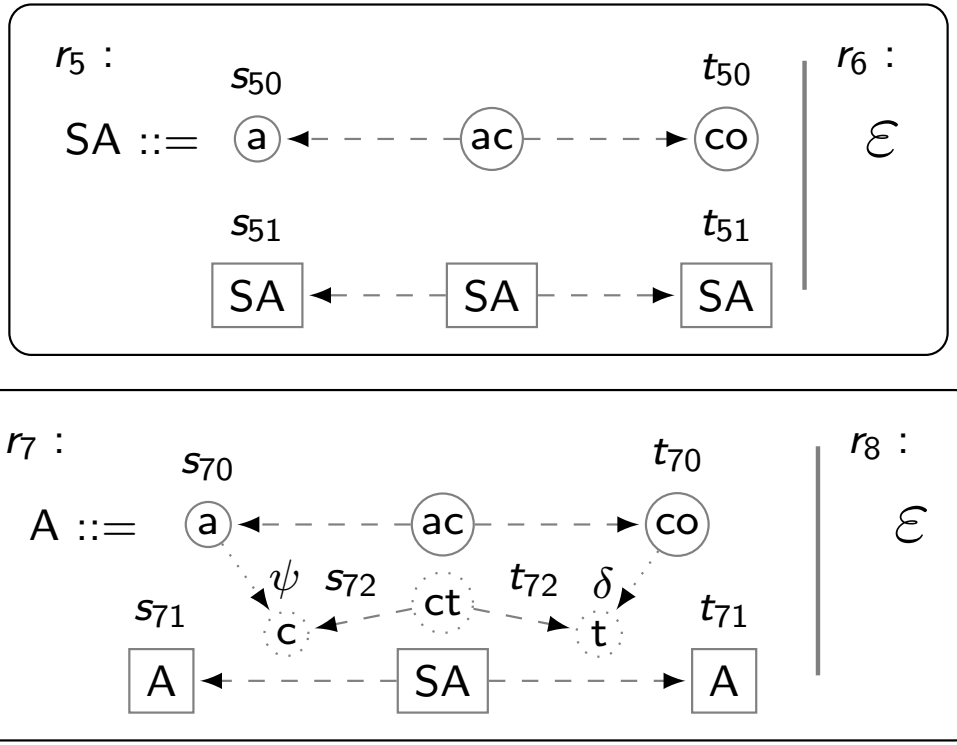
Pseudocode to Controlflow – Full



Class to Database – Full

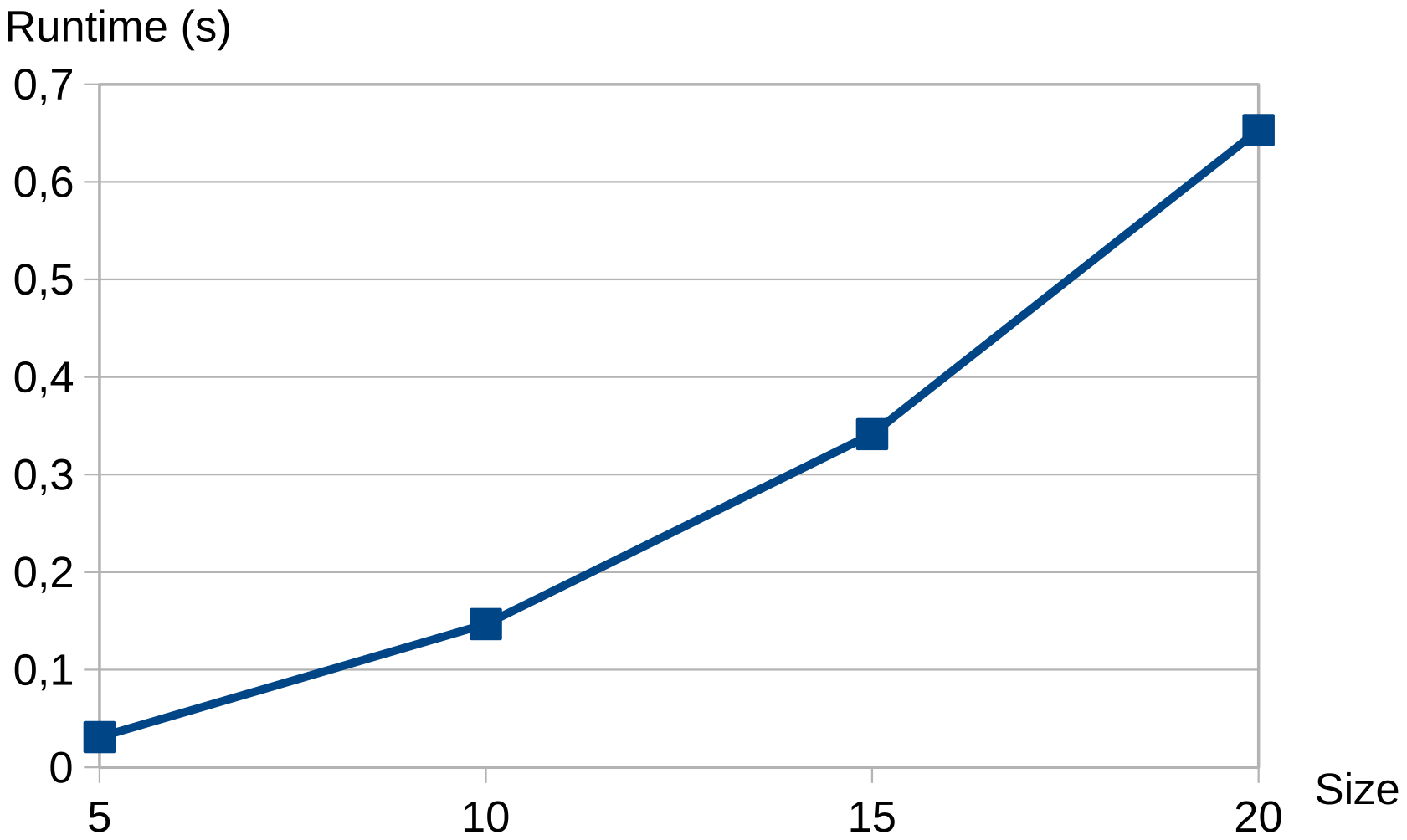


Class to Database – Full



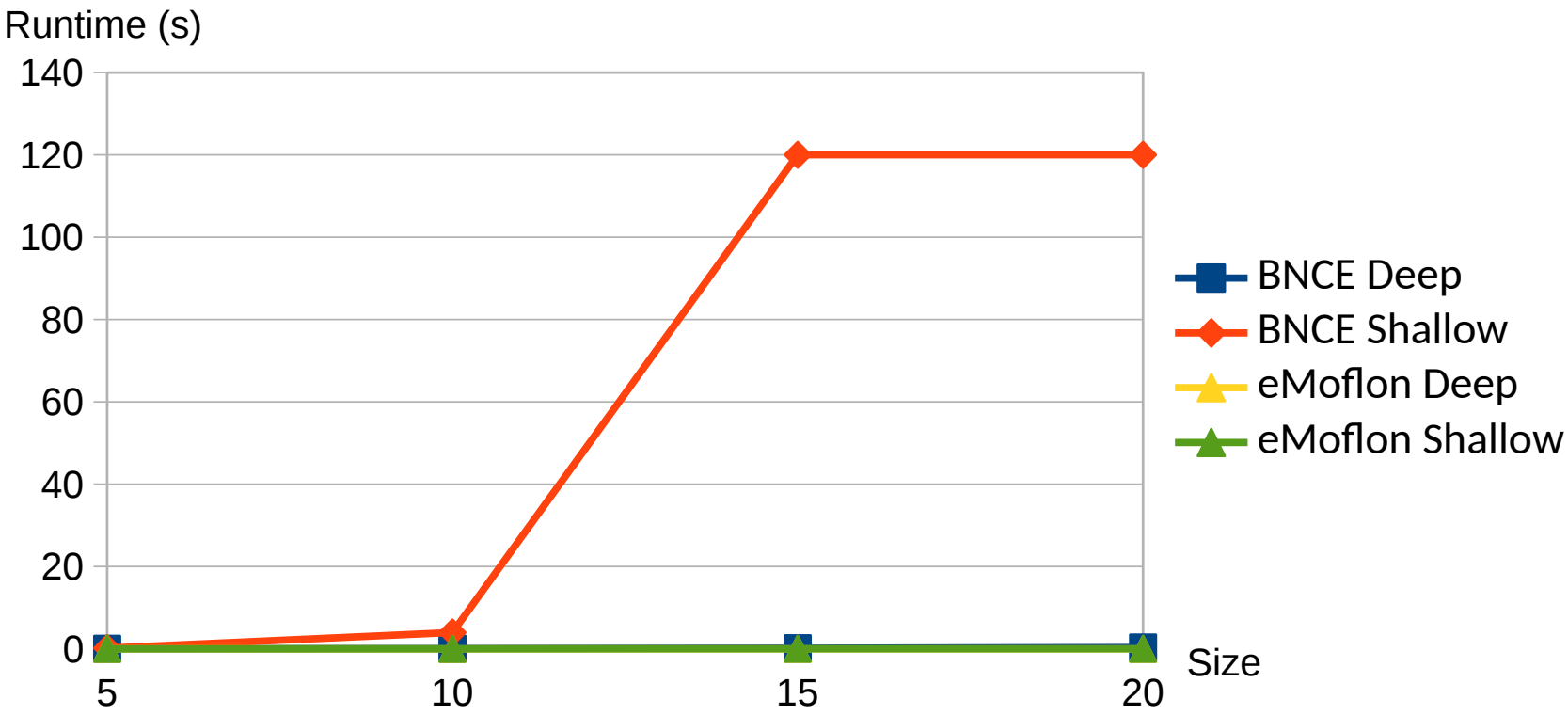
Performance Evaluation

- *Star* to *Wheel*



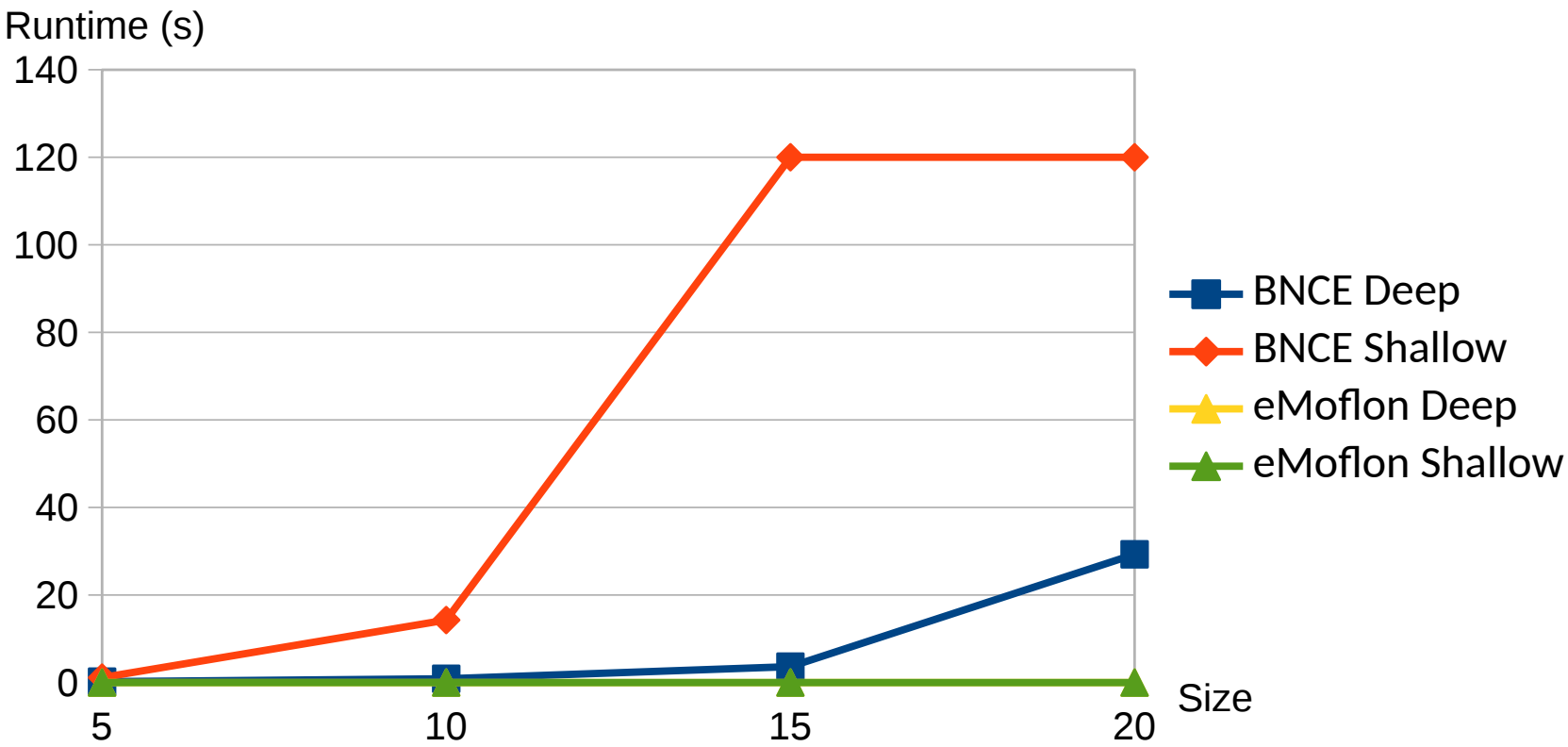
Performance Evaluation

- *BTree* to *XBTree*



Performance Evaluation

- *StateMachine* to *PetriNet*



Performance Evaluation

- *Class* to *Database*