Advanced Topics in Software Engineering (02265)

Ekkart Kindler

DTU Compute
Department of Applied Mathematics and Computer Science

V. Transformations

Overview

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

1. Model to Text Transformation (M2T)
   (JET → last week)

2. Model to Model Tranformation (M2M)
   (TGG → today)

3. Other approaches
   (QVT, ...)

4. Overview and classification

→ Is there a fundamental difference between M2T and M2M?

→ Differences between different M2M technologies!

2.1. Motivation

Up to now:

- Use of modelling notations (M1)!
- Development of modelling notation (M2)!
- Generate code (or other text) from models!

How do we

- Transform one model into another?
- Transform changes in the target model back to the source model?
- Keep different models consistent (synchronization)?
- Identify changes and transfer them to other models?
  (Version control, Identify design decisions, ...)

Is there a difference to a transformation to text?

2.2. Background

- Grammars, a reminder!
  (and two different purposes)

- Graph grammars
  (same thing just with graphs)

Example: Grammar

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

E	→	T | E + T	Expression
T	→	F | T * F	Term
F	→	I | N | (E)	Factor
I	→	a | … | z | Ia | … | Iz	Identifier
N	→	0 | … | 9 | N0 | … | N9	Number

Example: Gramm

Left-hand side (of a rule)

Terminal symbols: +, *, (,), a, … z, 0, …, 9

Non-terminal symbols: E, T, F, I, N

E	→	T | E + T	
T	→	F | T * F	Term
F	→	I | N | (E)	Factor
I	→	a | … | z | Ia | … | Iz	Identifier
N	→	0 | … | 9 | N0 | … | N9	Number

Right-hand side (of a rule)

Meta-symbols: →, |

Meta-meta-symbol: …

Example: Parsing

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

x + y * (x + 1)

E → E + T → T + T → F + T → I + T → x + T →
x + T * F → x + F * F → x + I * F →
x + y * F → x + y * (E) → x + y * (E + T) →
x + y * (T + T) → x + y * (F + T) →
x + y * (I + T) → x + y * (x + T) →
x + y * (x + F) → x + y * (x + N) →
x + y * (x + 1)

Grammars

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- A grammar consists of
  - rules and
  - one axiom (a non-terminal)
- A rule says how, within a character sequence, a sub-sequence can be replaced by another sequence

  Technically, $E \rightarrow T \mid E + T$ represents two rules!

- Grammars are often restricted to character sequences of non-terminals on the left-hand side
- In context-free grammars (*cfg*), the left-hand side consists of exactly one non-terminal symbol

Grammars: Use 1

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- can be used to define the legal syntax of a programming language or some other textual language
- can be used to build parsers and for building the syntax tree

  → Formal languages
  → Compiler construction
  → Parsing theory

  → Xtext

- In this sense, ”grammars are meta-models” or ”meta-models are grammars”

Another example: ”Grammar”

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

$\#< \;\rightarrow\; \#>$

$a< \;\rightarrow\; <a$

$>a \;\rightarrow\; a>$

$>\# \;\rightarrow\; <\#$

What does that “grammar” do?

Another example:

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

#>aaaaaa# → #a>aaaaa# → #aa>aaaa# → #aaa>aaa# →
#aaaa>aa# → #aaaaa>a# → #aaaaaa># → #aaaaaa<# →
#aaaaa<a# → #aaaa<aa# → #aaa<aaa# → #aa<aaaa# →
#a<aaaaa# → #<aaaaaa# →
#>aaaaaa# → ...

”Grammar”: Use 2

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- No axiom (just a ”start configuration”)

In this context, the “grammar” is typically called “rewriting system”.

- No distinction between terminals and non-terminals (conceptually, all symbols can be considered to be terminals; technically, all symbols can be considered to be non-terminals)

- The purpose is not parsing a string; it is about ”defining behaviour”; the string is just the current state (→ Markov algorithms)

Traditionally, the “algorithms” would be required to terminate; but, if they don’t, it just defines infinite behaviour (reactive systems).

Background

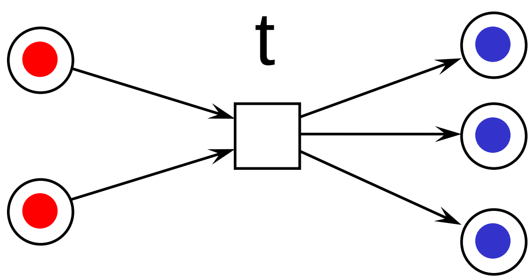
DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Grammars, a reminder!
  (and two different purposes)

- Graph grammars
  (same thing just with graphs)

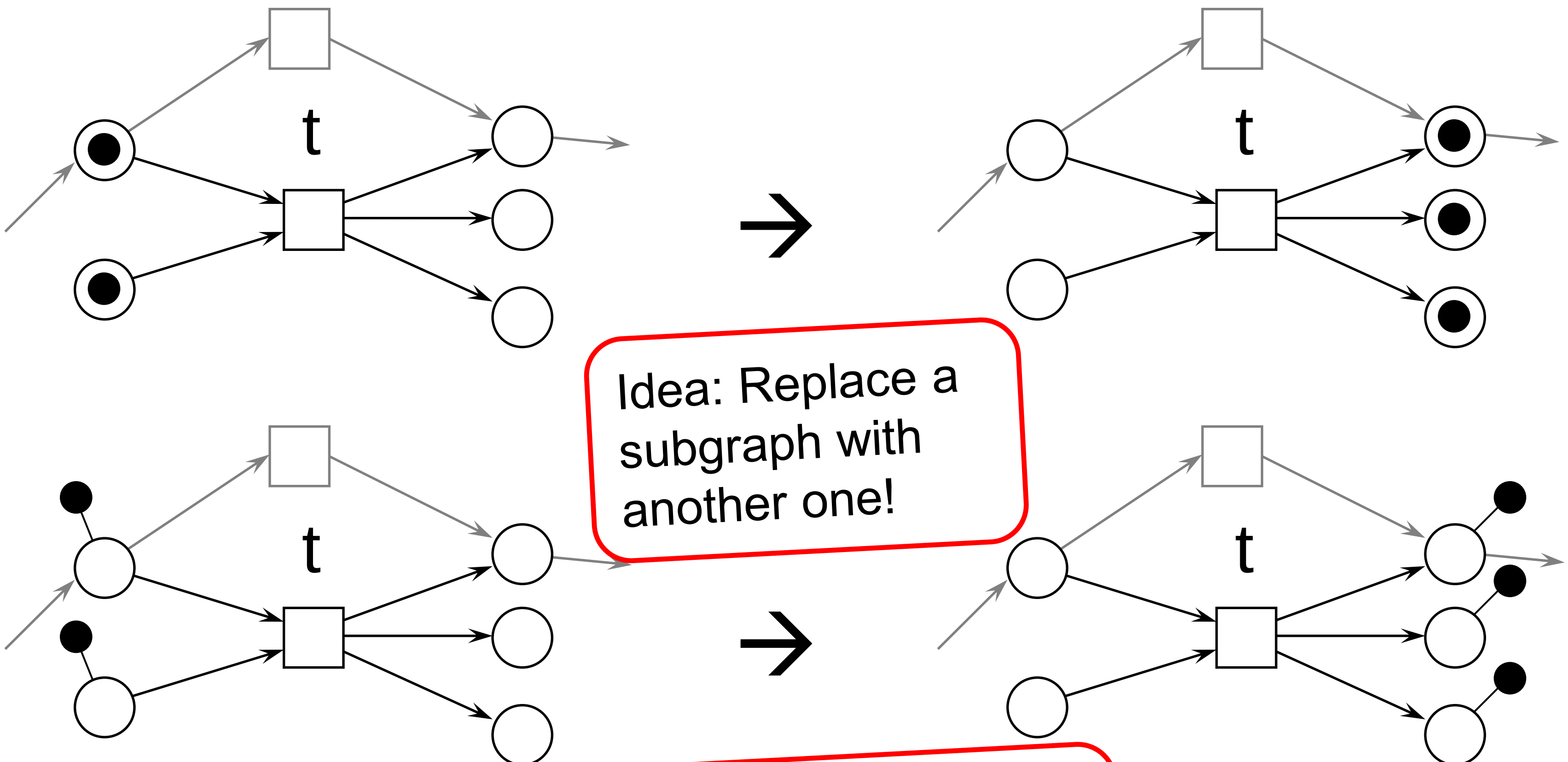
Example

Reminder: Firing rule of Petri nets



DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Firining rule of a Petri net transition



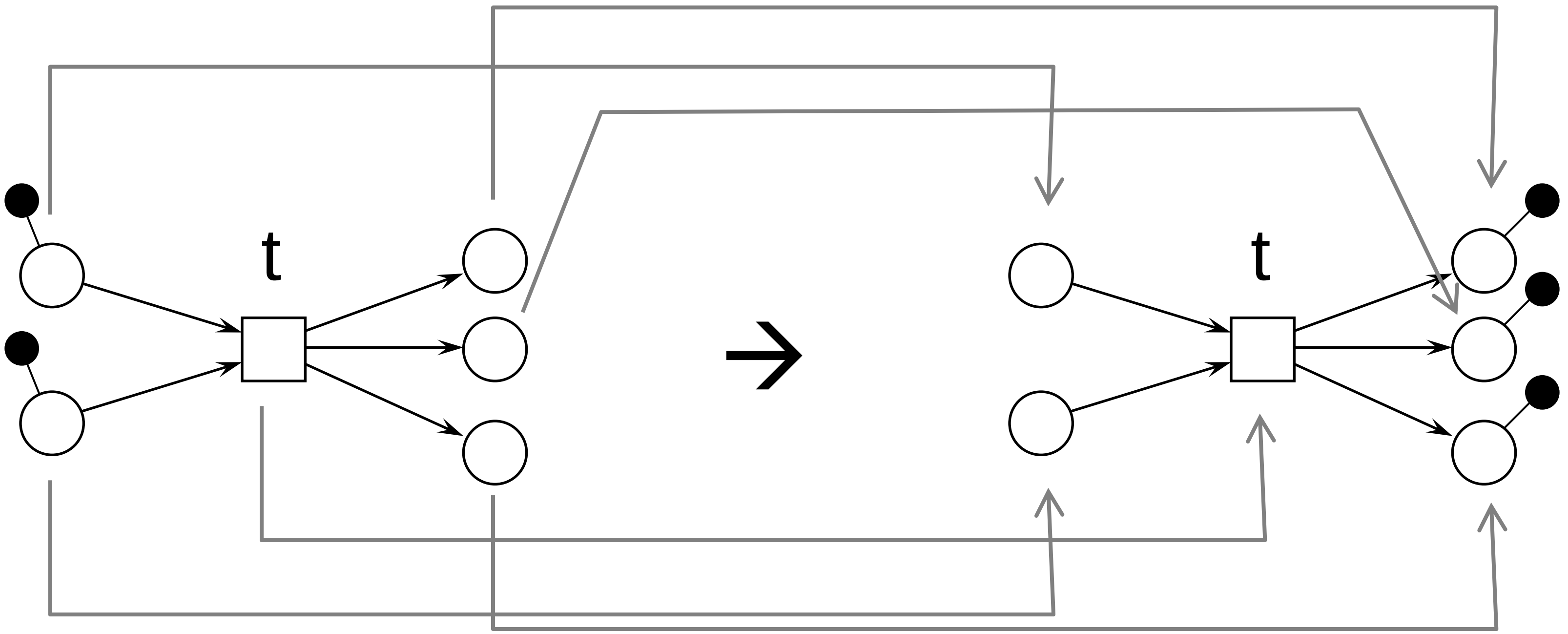
Idea: Replace a subgraph with another one!

What is the same on the left-hand and the right-hand side?

Example

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

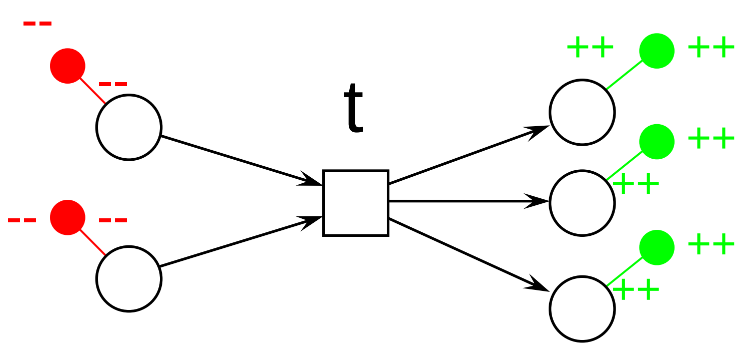
Firining rule of a Petri net transition as a graph grammar rule



Could be indicated by a mapping (also between the arcs).
For humans “mostly obvious” ☺

Firing of Petri nets

Different representation: single graph, indicating in colours (and labels) what does not change, what is deleted (--) and what is added (++):

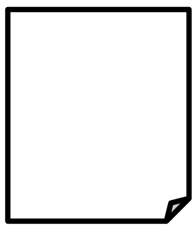


Exactly the same information as on pervious slide: just more concise

This is “Use 2” of graph grammars (defining evolving behaviour).

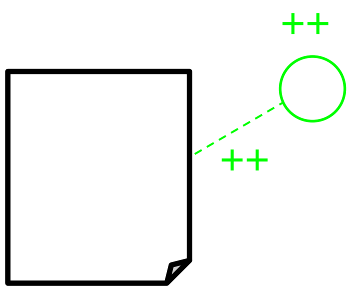
Use 1:

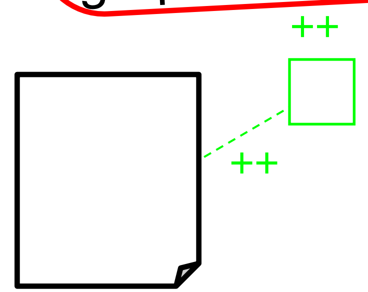
Defining the syntax of Petri nets:

Axiom: 

Note: In the tool that we are using, all nodes of the axiom will be "green" (++) nodes.

Which interpretation makes more sense, depends on whether you want to consider the axiom as a rule or as a graph.

Rule 1: 

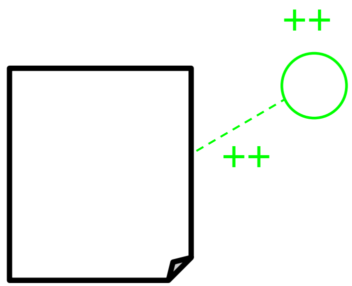
Rule 2: 

Use 1:

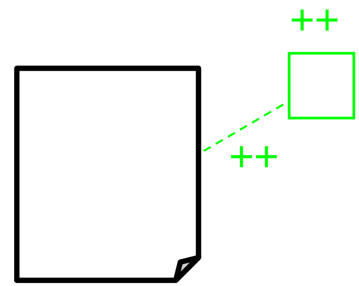
This graph grammar defines the syntax of Petri nets. It can be used to generate or parse a syntactically correct Petri net.

Defining the syntax of Petri nets:

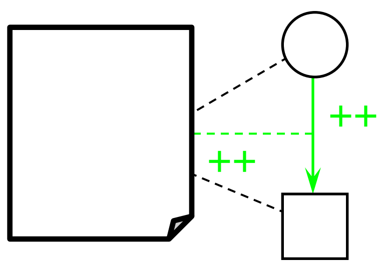
Rule 1:



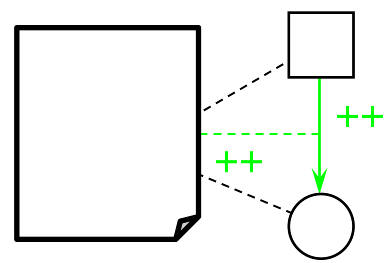
Rule 2:



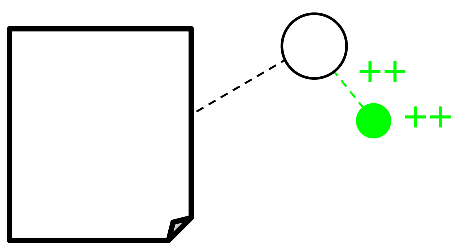
Rule 3:



Rule 4:



Rule 5:



Note that this is not the (main) purpose of GGs here; the example should just illustrate the “Use 1” of GGs.

2.3. Triple Graph Grammars (TGGs)

- Using graph grammars for defining the relation between models (in a special way),
- for transforming them accordingly, and
- keeping the resulting models consistent.

Outline

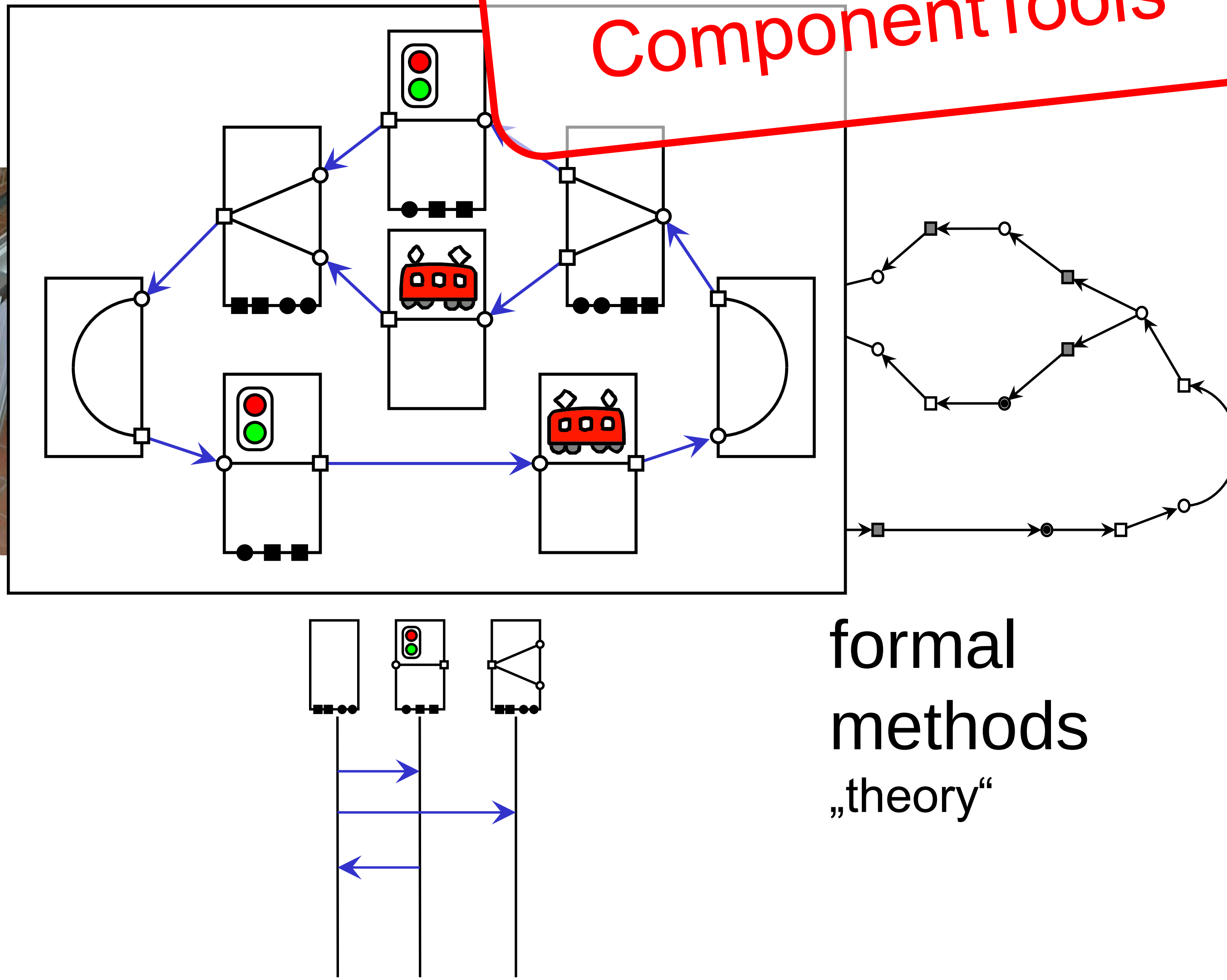
DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Example
- Semantics
- Strength
- Problems and Weaknesses
- Extensions and Open Issues

An Example

Borrowed from ComponentTools

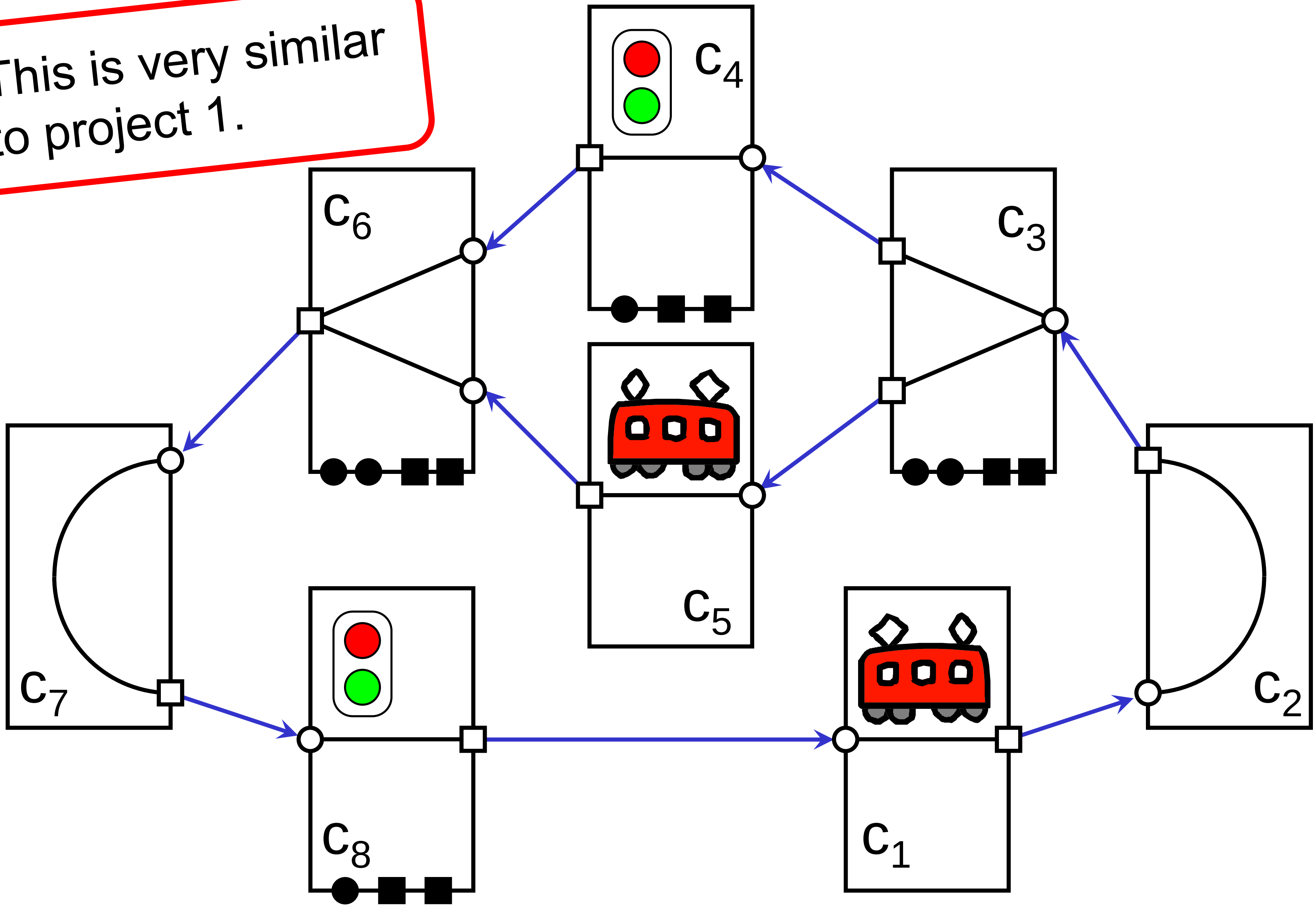
engineer
„practice“



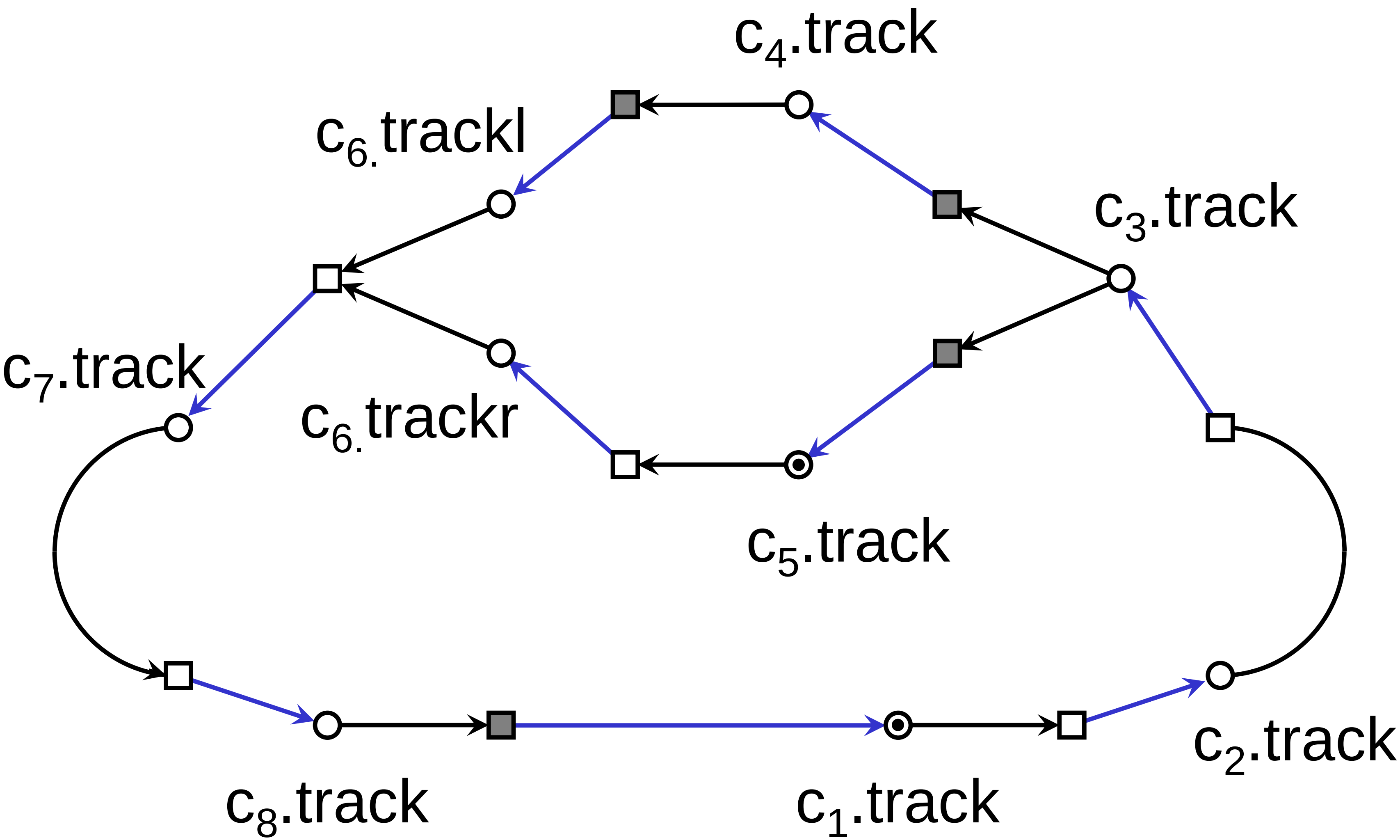
formal
methods
„theory“

Project Plan

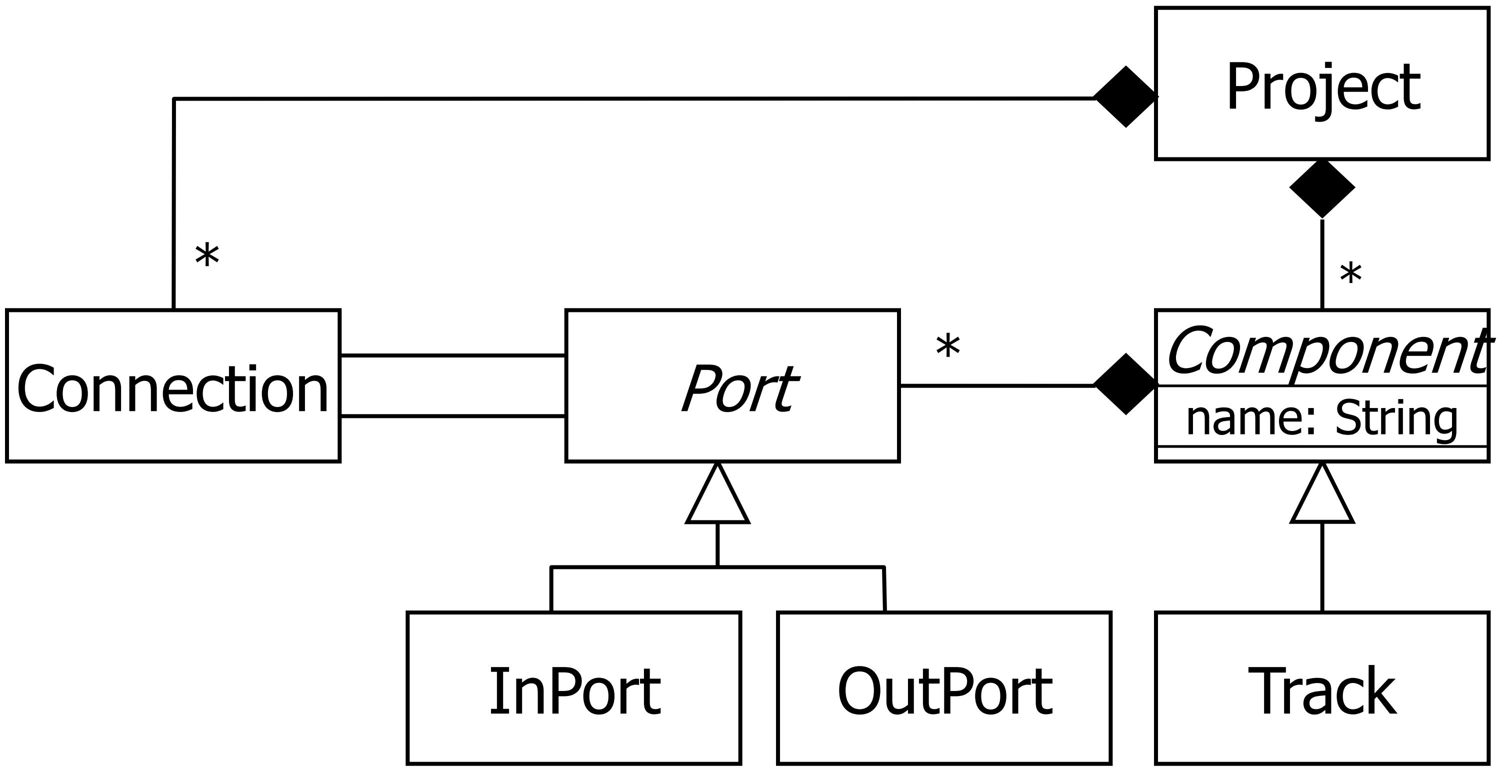
This is very similar to project 1.



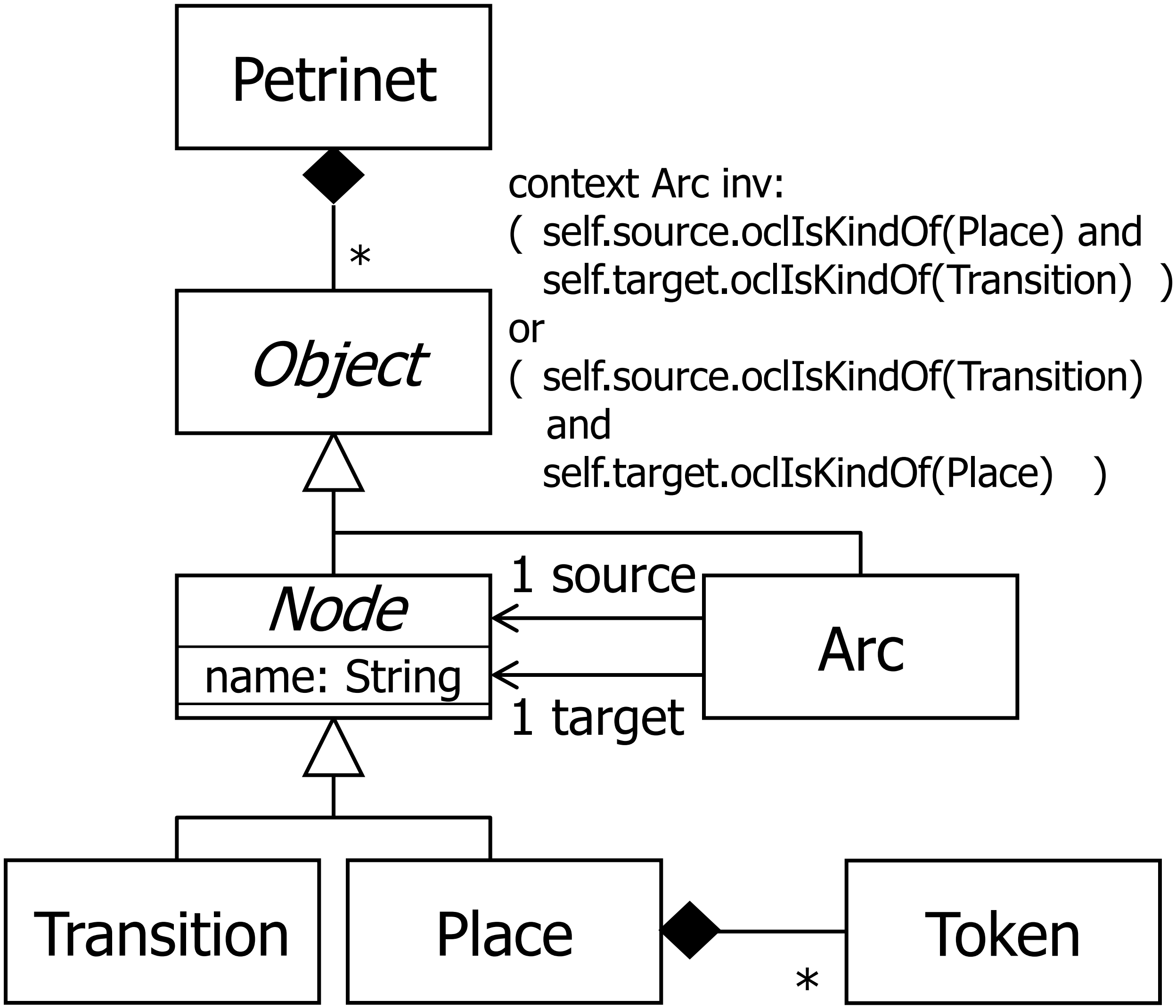
Corresponding Petri Net



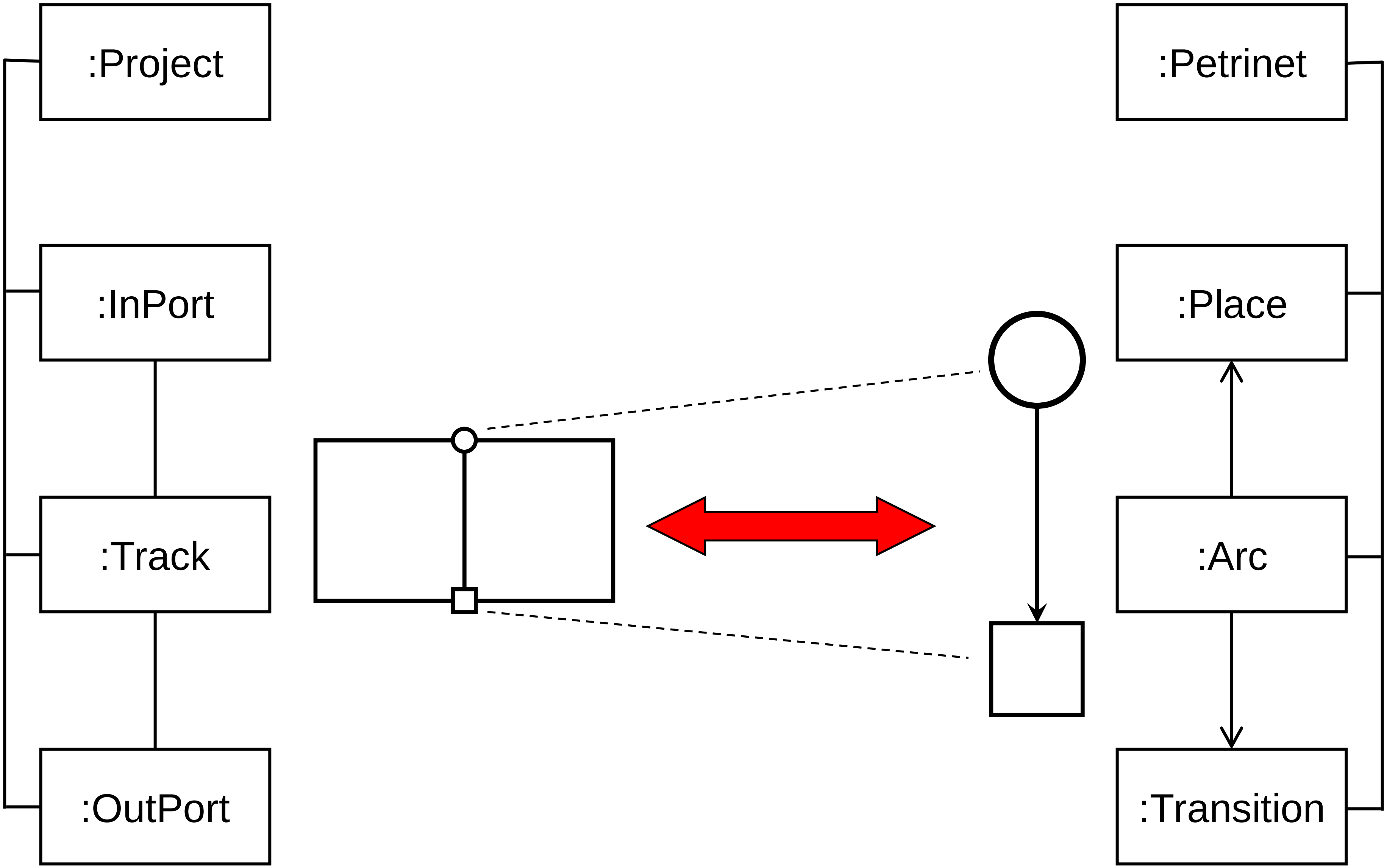
Project: Meta-model



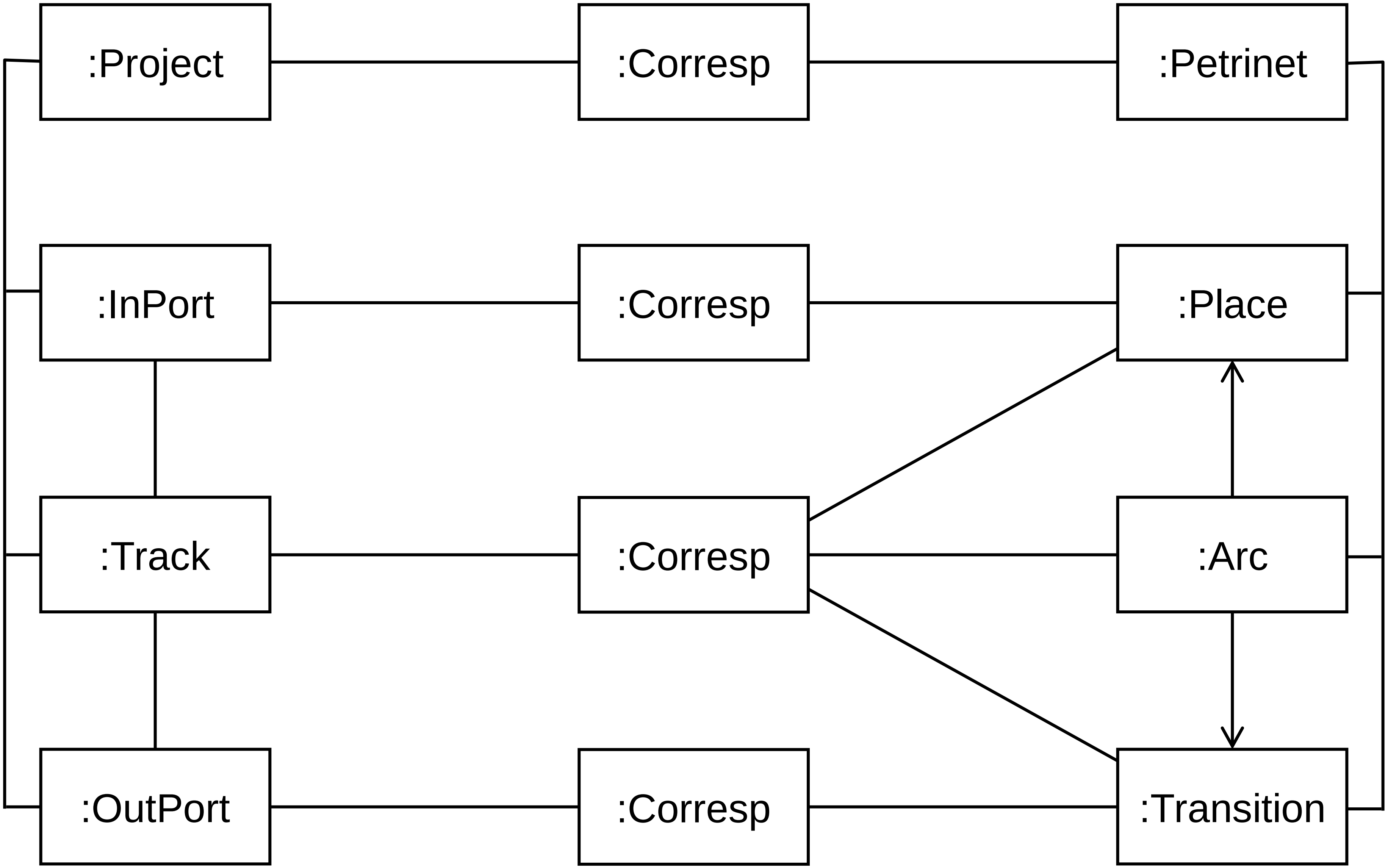
Petri nets: Meta-model



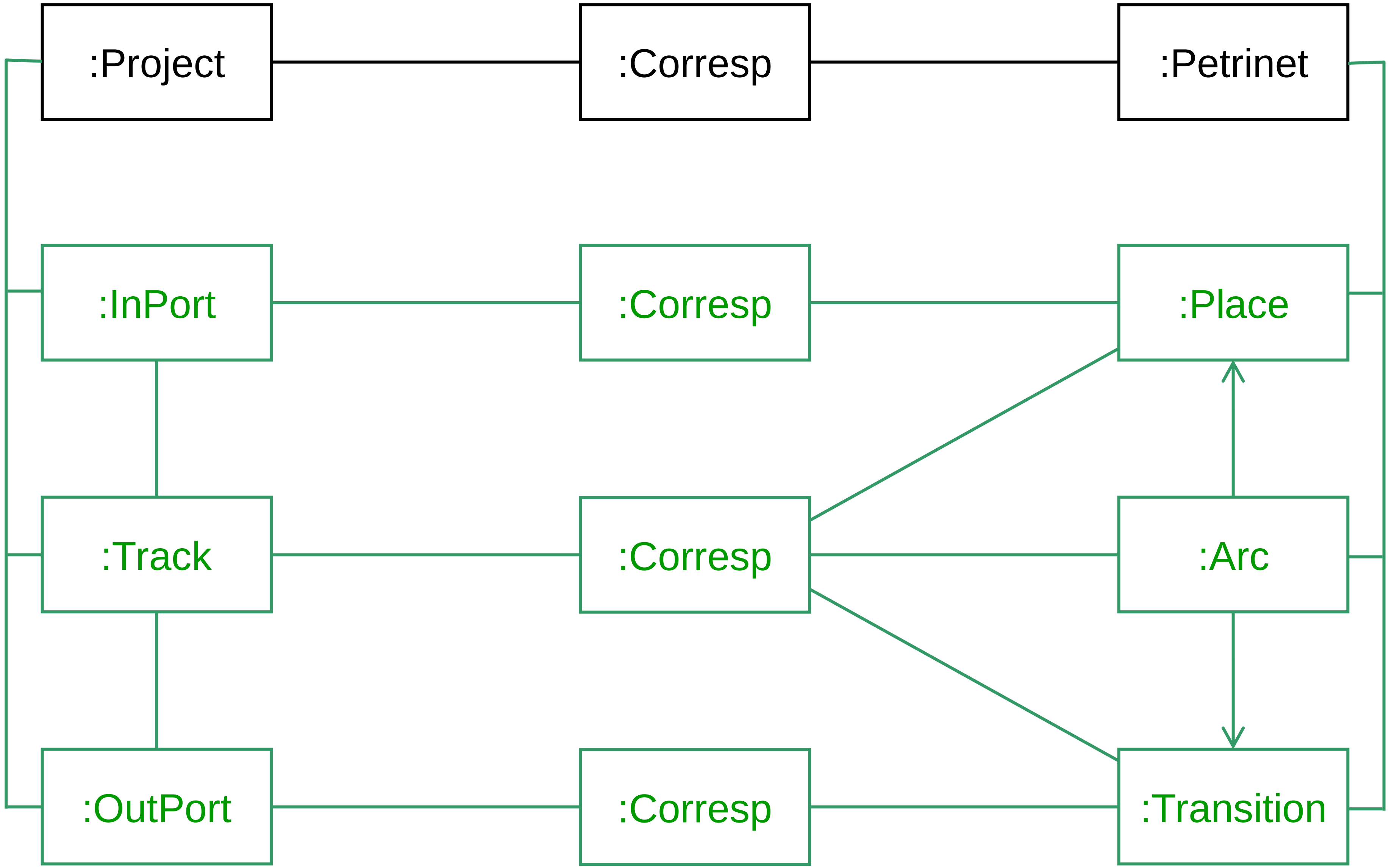
Transformations



Transformations

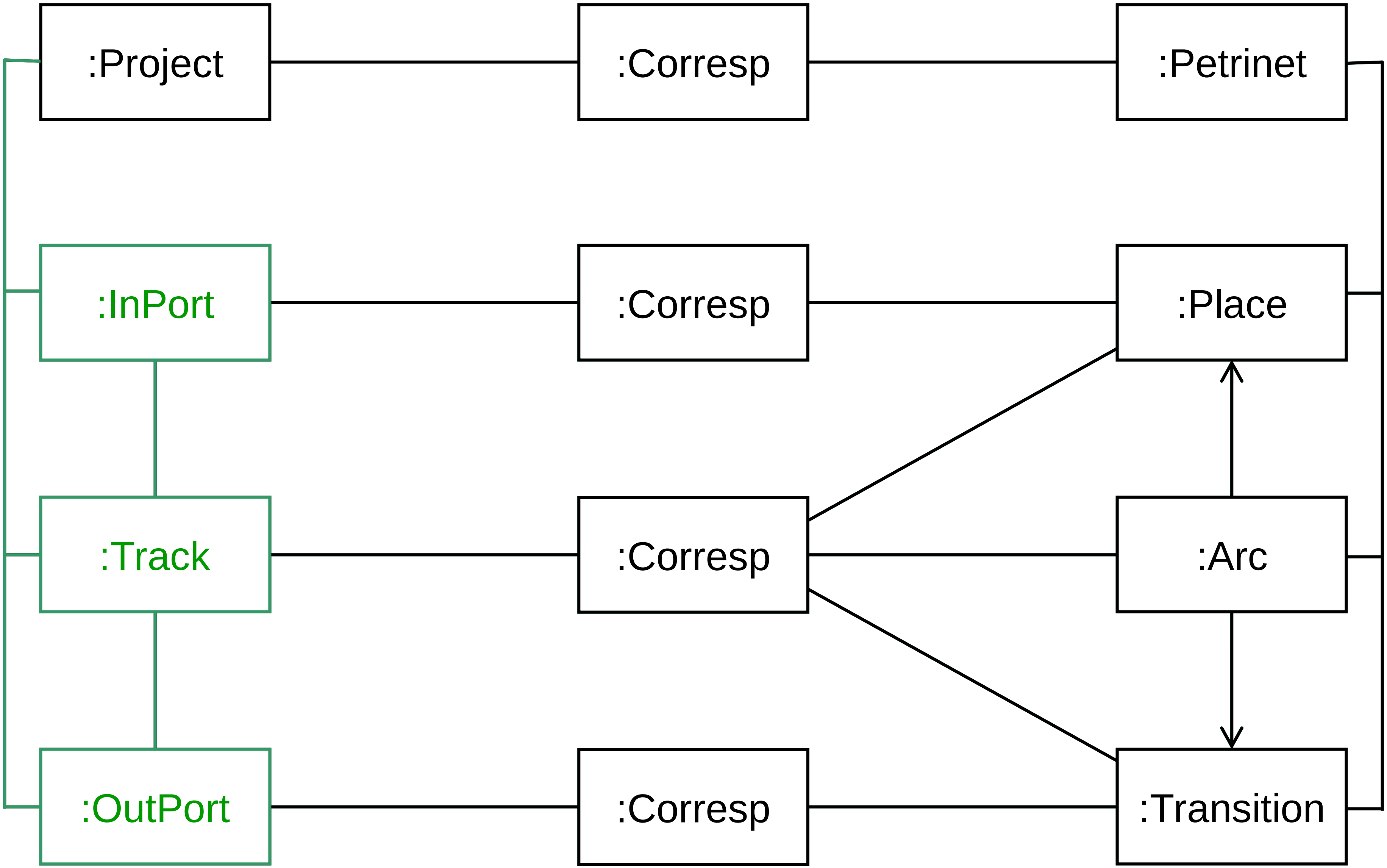


Triple Graph Grammar Rule



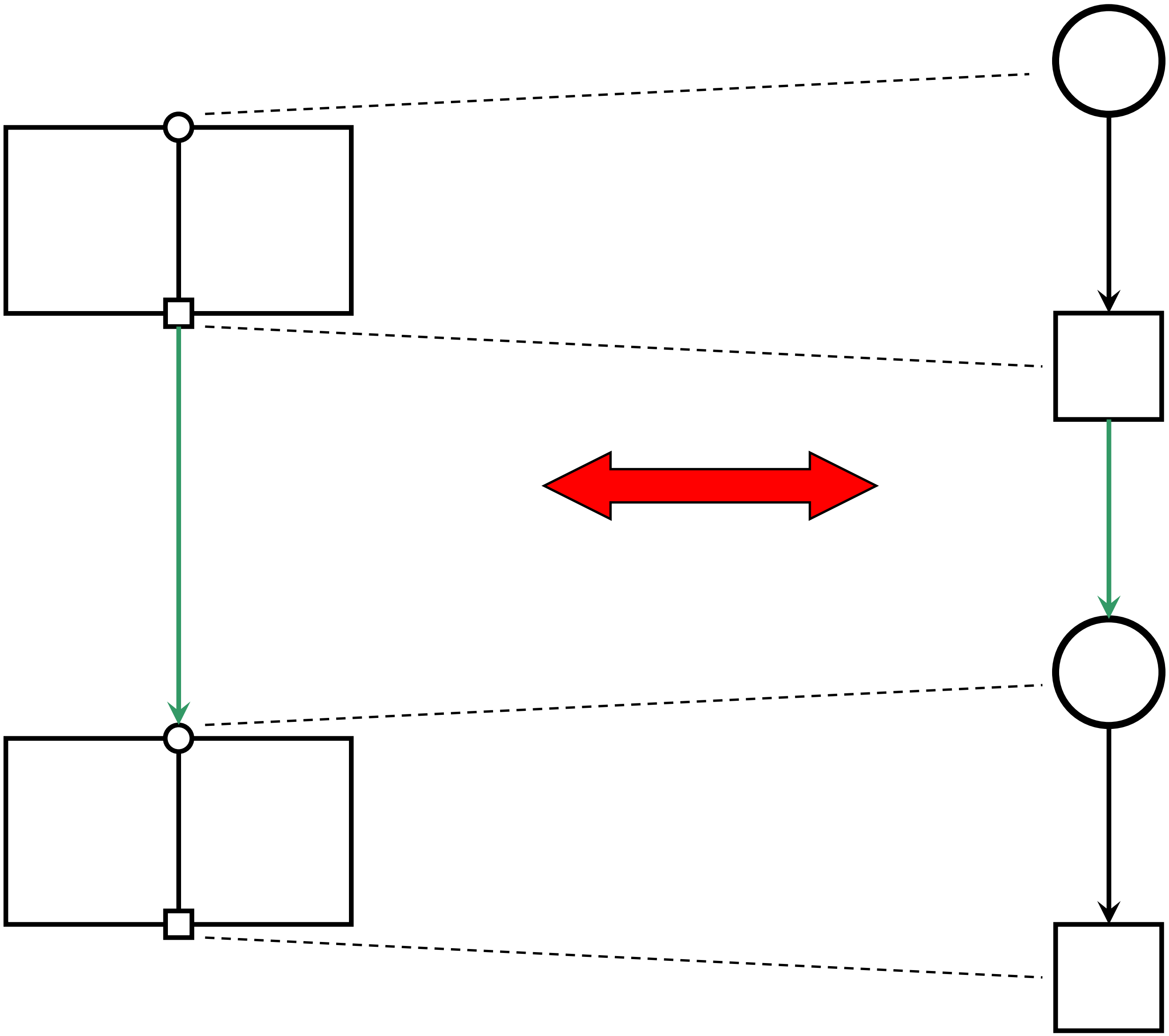
TGG-Rule Application

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



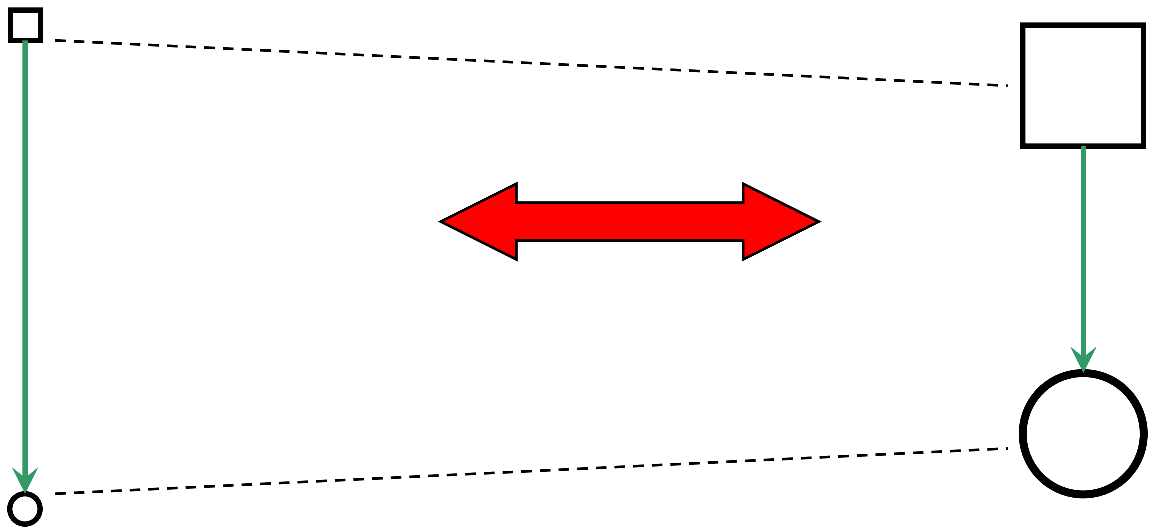
Transf. of connection

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



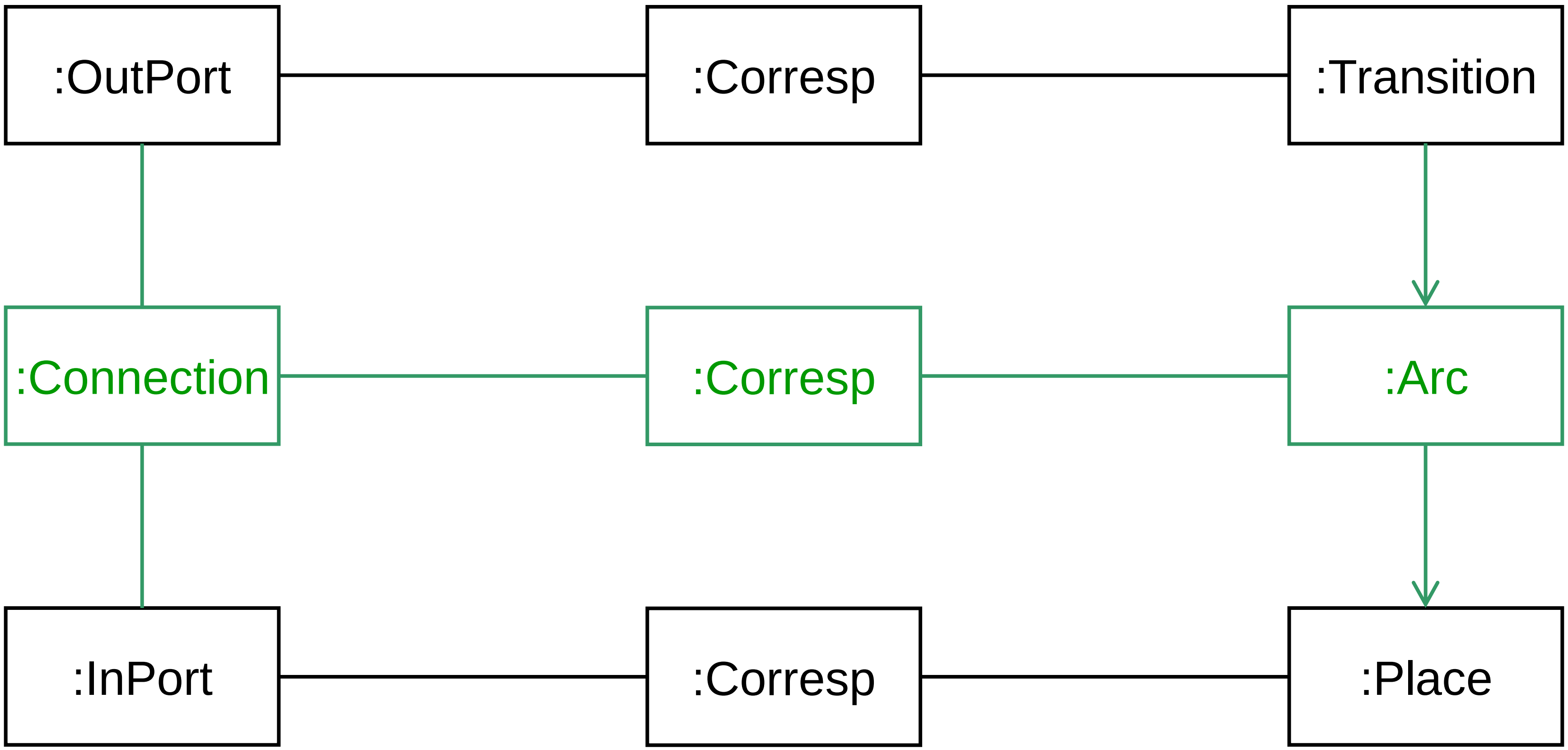
Transf. of connection

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



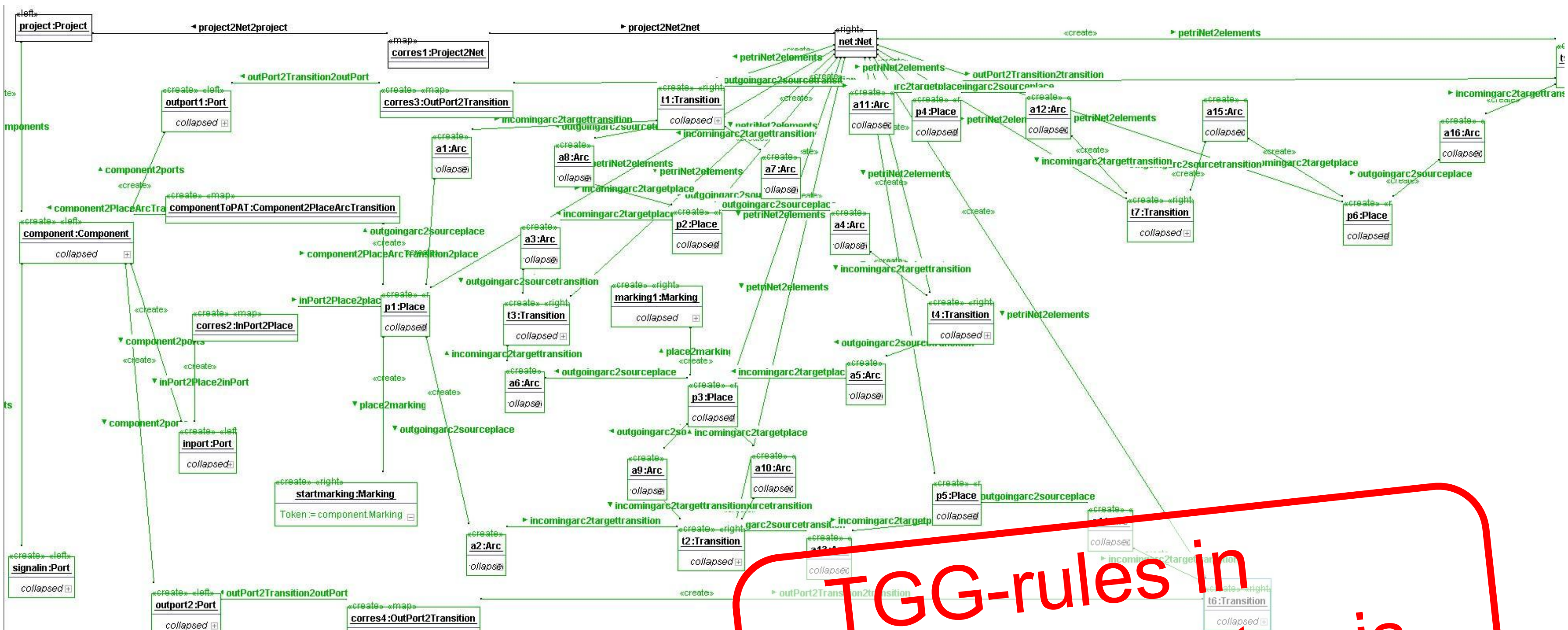
TGG-rule: Connection

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



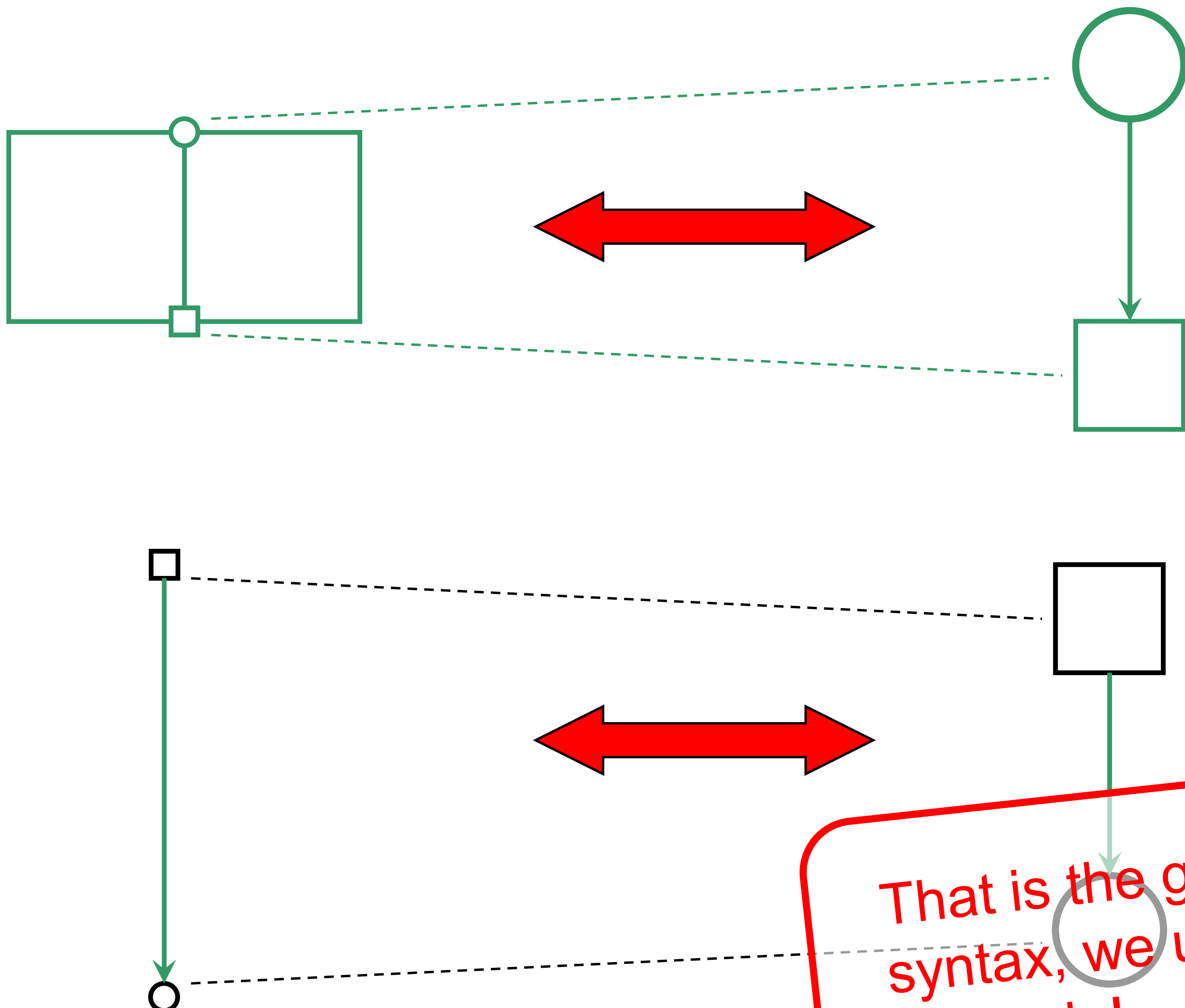
A rule in practice

A real example from component tools.



TGG-rules in abstract syntax is very verbose!

TGG-Rules in “graphical syntax”

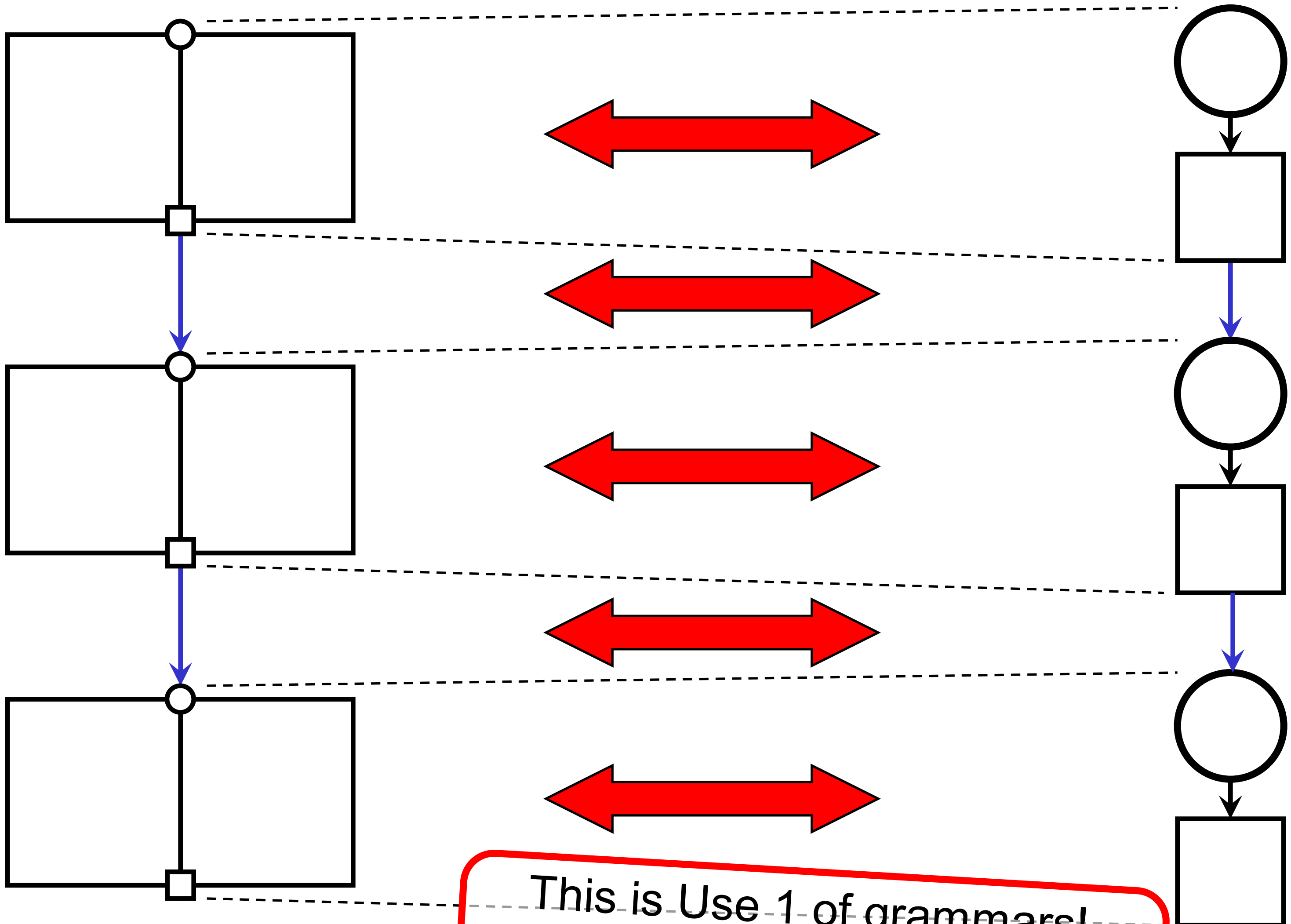


That is the graphical syntax, we used in our example!

Outline

- Example
- Semantics
- Strength
- Problems and Weaknesses
- Extensions and Open Issues

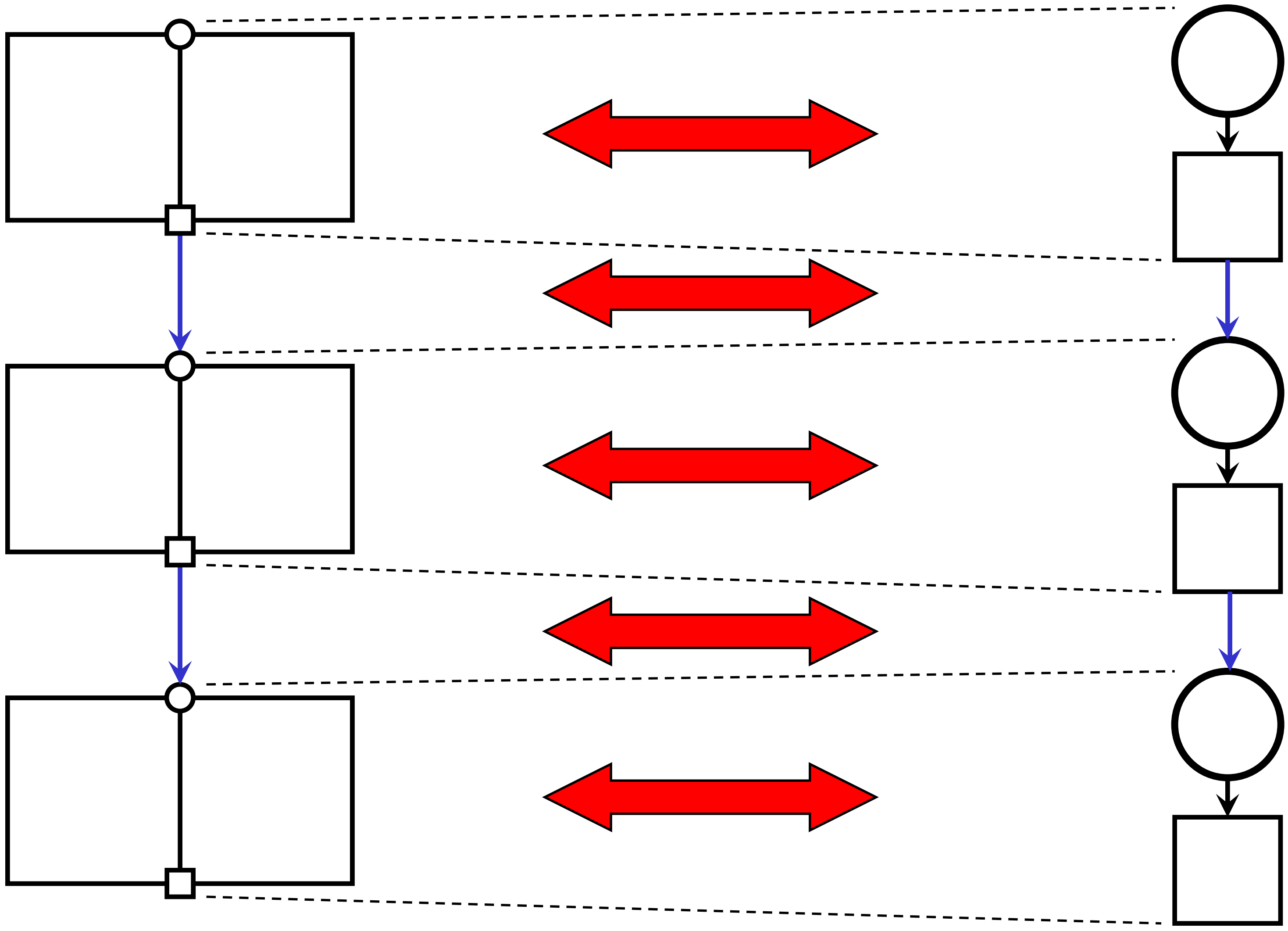
Semantics of TGG-Rules



This is Use 1 of grammars!
Just for two/three models in parallel.

“Model-driven Execution”

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



Outline

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Example
- Semantics
- Strength
- Problems and Weaknesses
- Extensions and Open Issues

Strength of TGGs

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Rules are declarative and local
- Semantics works both ways

Under some reasonable constraints, which are yet to be identified.

- Yet, the transformations are operational (compiler / interpreter approach)
- Transformations are operational in both directions!

Corollary of locality

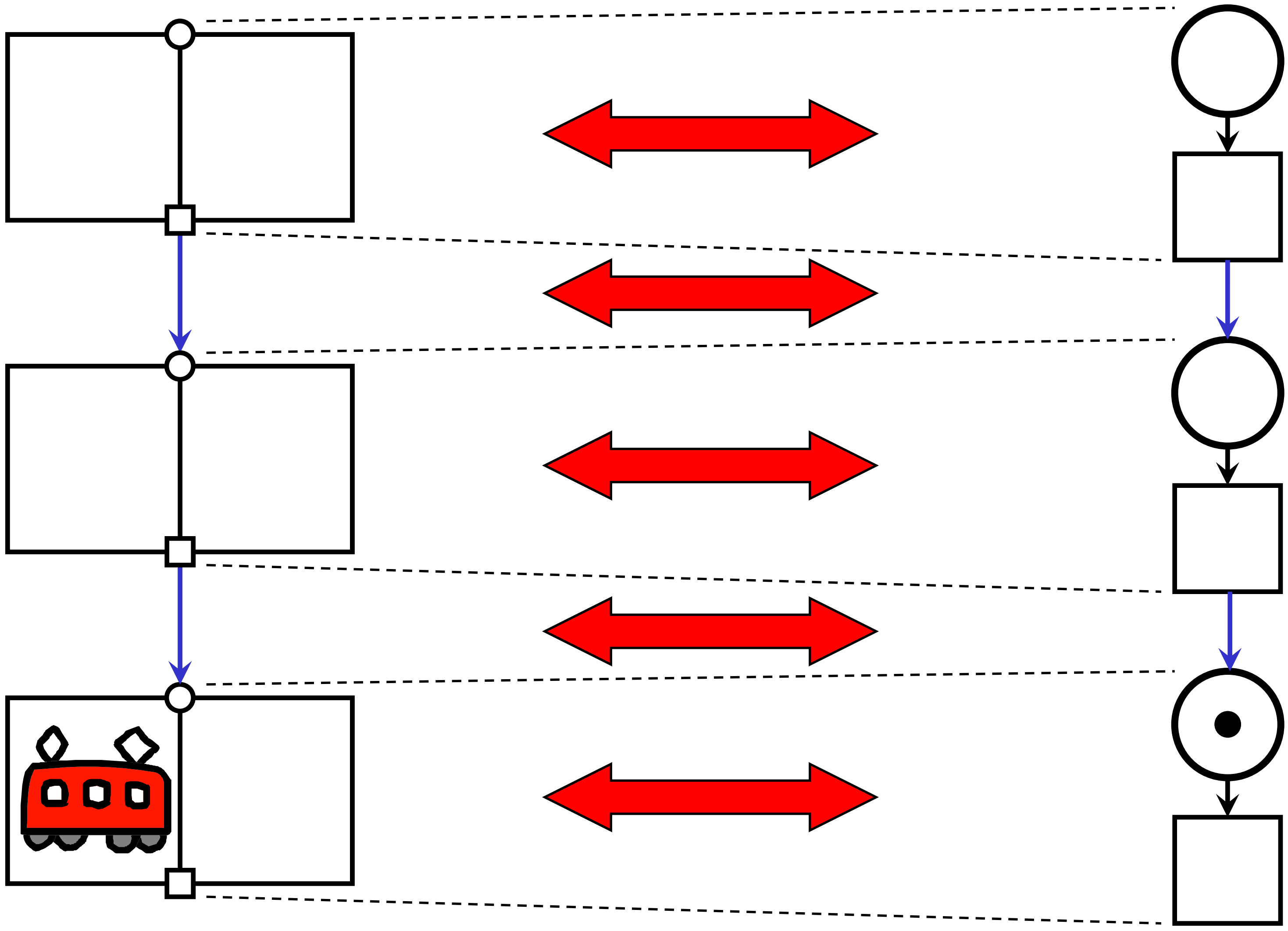
DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Transformations can (in principle) be verified for semantical correctness

- Approach works incrementally!

Incremental application

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



TGGs are good for

- Defining transformations between models that are structurally similar

- Executing these transformations (in models of reasonable size)

TGGs are not so good for

- defining transformations between models that are very different in structure

TGGs should be combined with other transformation technologies such as templates! How?

- defining the legal syntax for the models on each sides of the transformation

Use UML and OCL for defining the legal "syntax" of source and target (meta-modelling).

TGGs are not so good for

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- formulating the rules of real-world examples in abstract syntax

But, this is only a matter of better tool support!
(Could be a nice MSc-project!)

- sometimes there are many large but very similar rules

We need mechanisms for reusing and structuring rules (TGG++):
- “inheritance”
- combination and composition of rules (“where” / “when” → QVT)

Outline

- Example
- Semantics
- Strength
- Problems and Weaknesses
- Extensions and Open Issues

Extensions

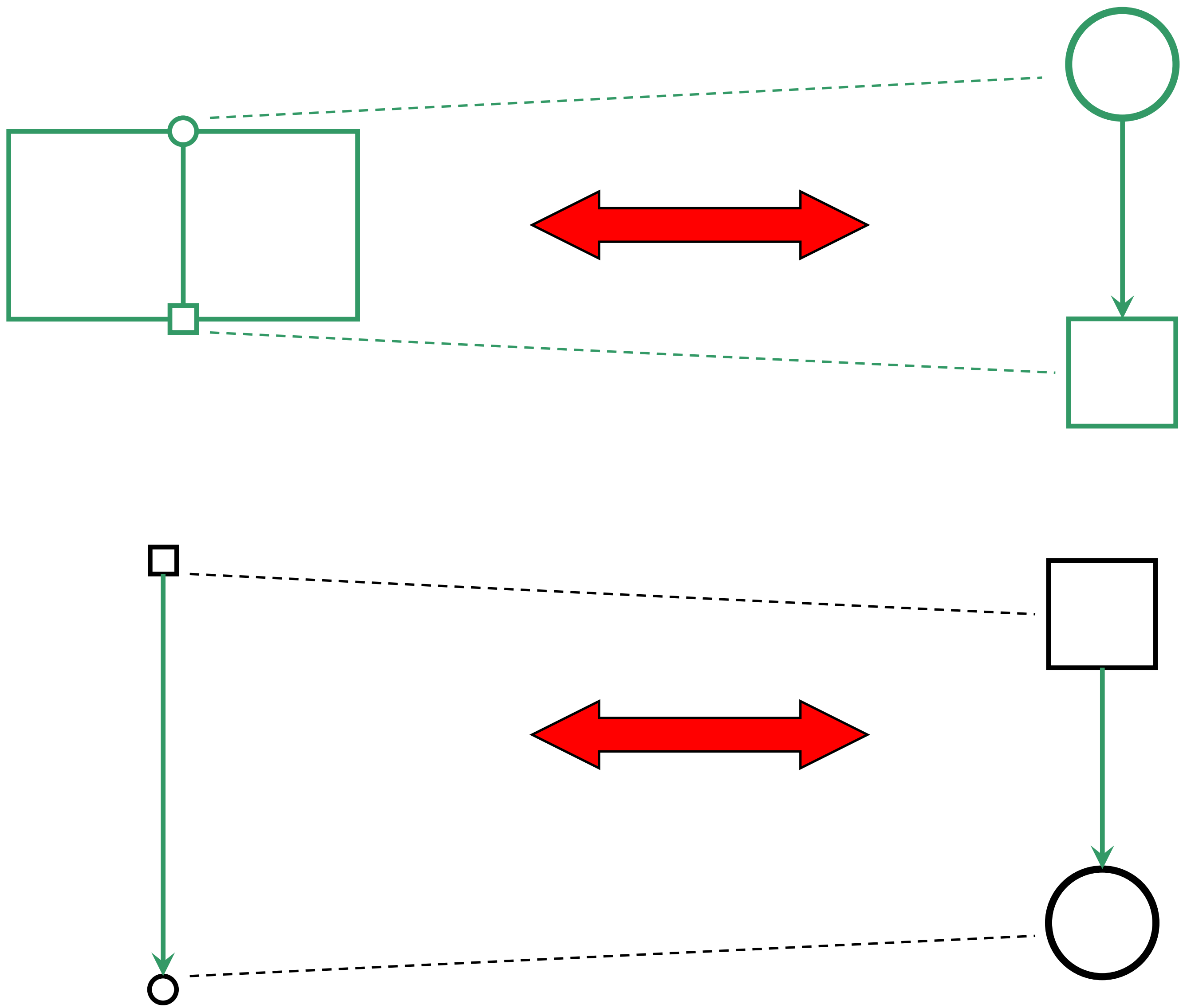
DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- TGG++
  - Inheritance of rules
  - where-clause
  - other “abbreviations”

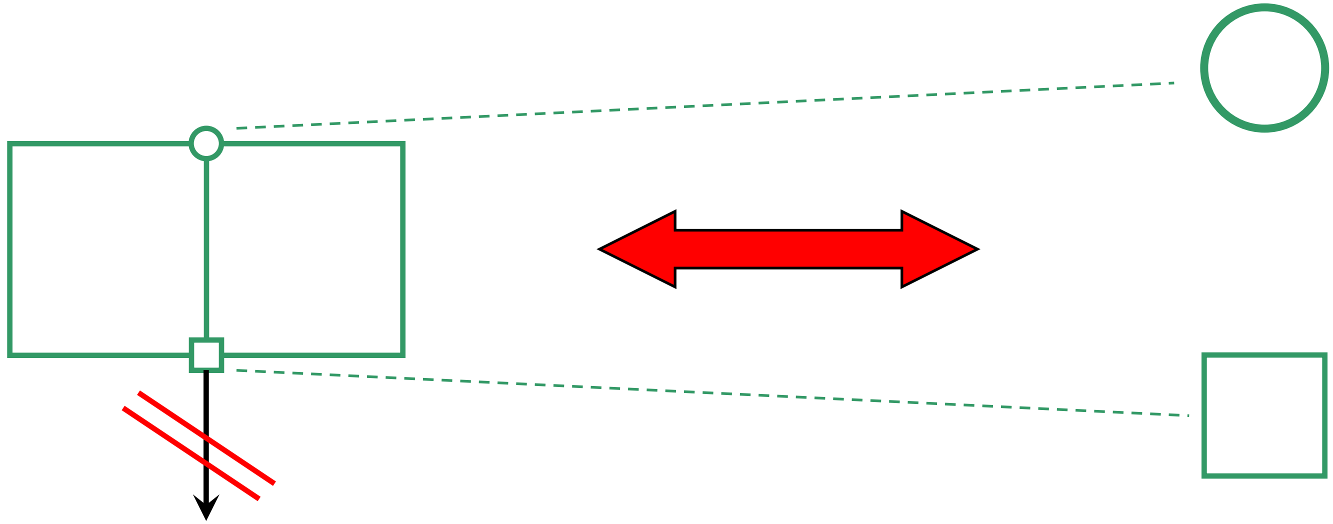
- Negation

TGG-Rules in “graphical syntax”

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

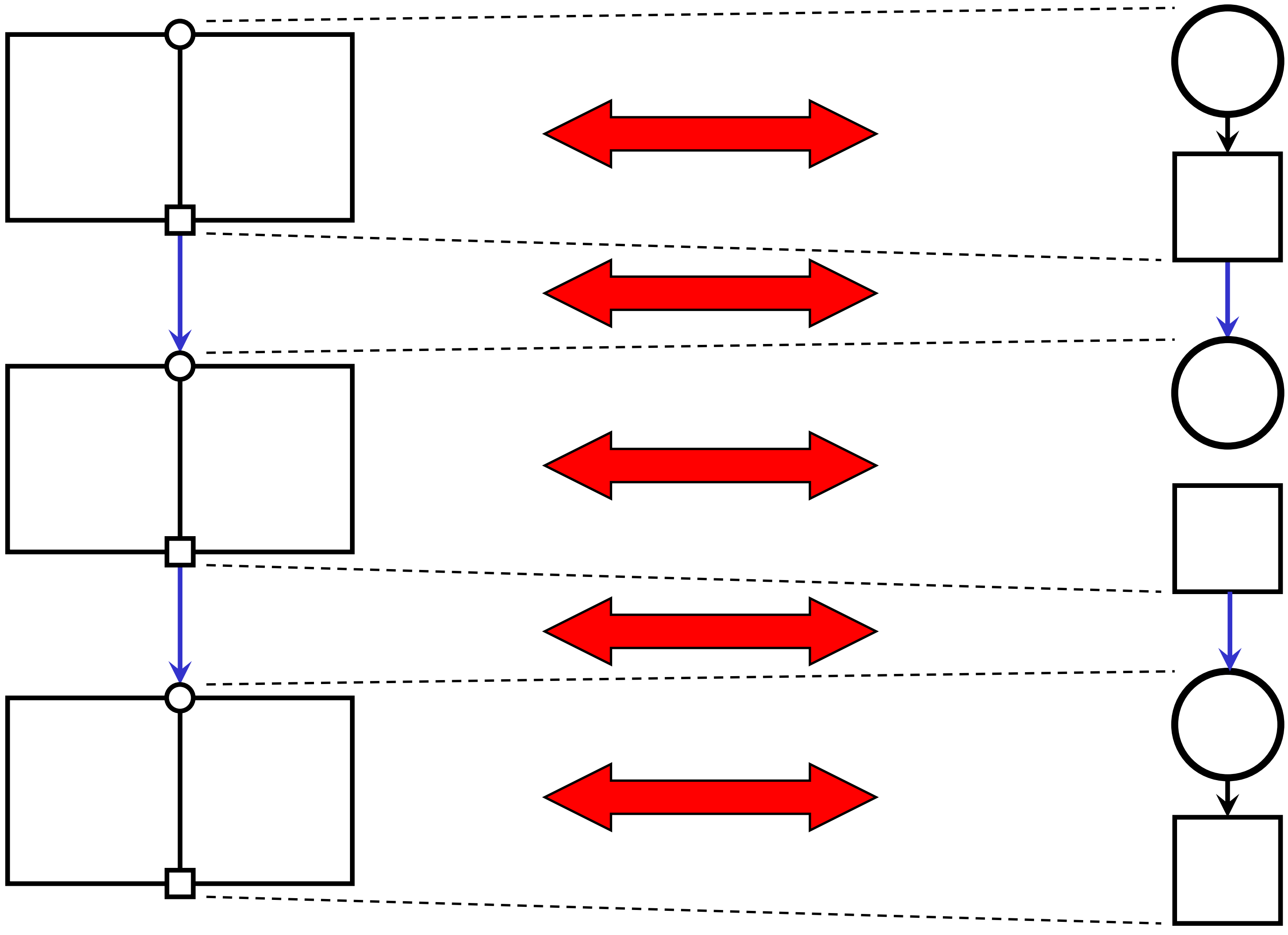


DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

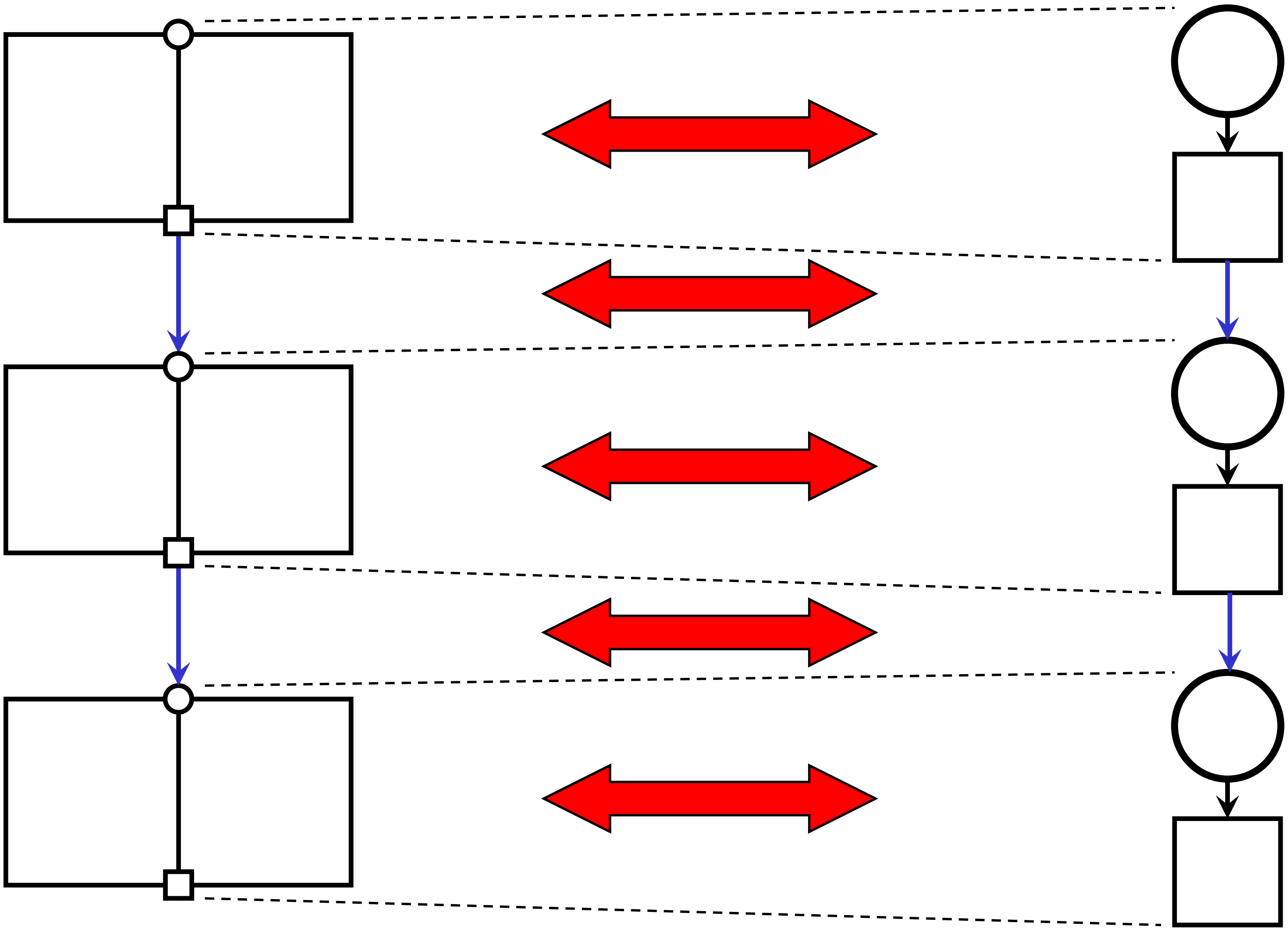


Generation Semantics

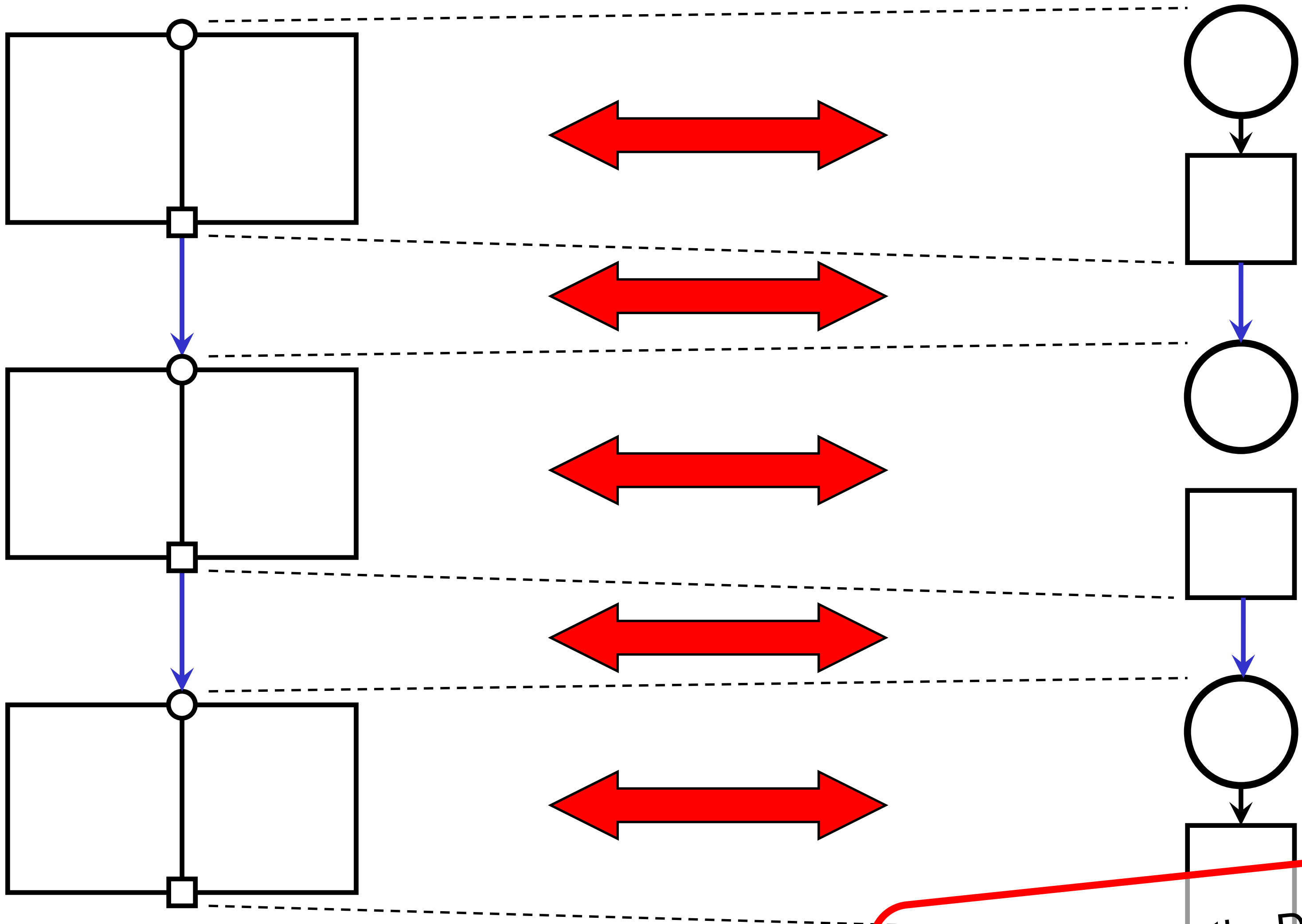
DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



“Model-driven Semantics”



DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



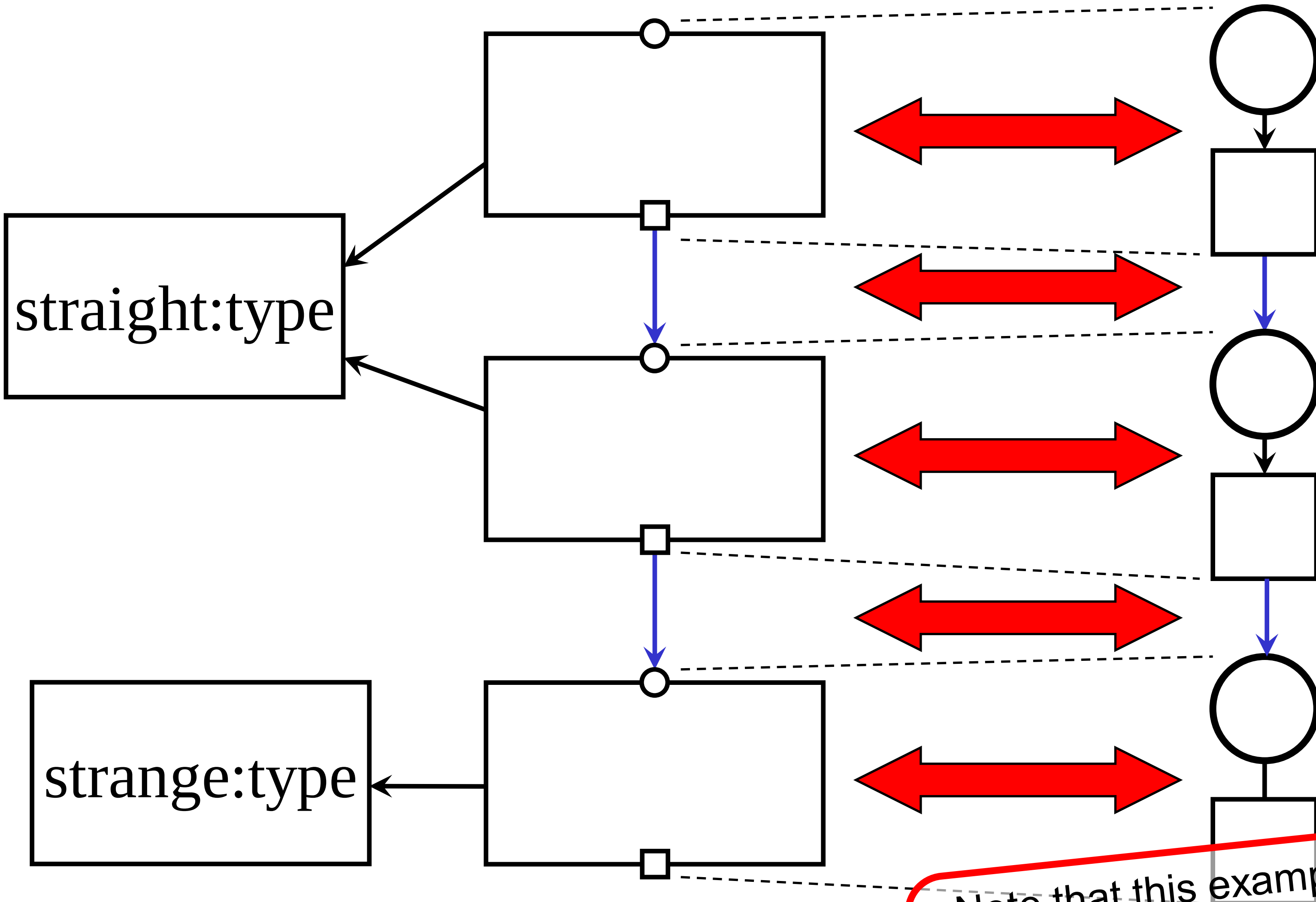
More problematic: Deletion of model elements. But, in principle doable.

Extensions

- TGG++
  - inheritance of rules
  - where-clause
  - other “abbreviations”

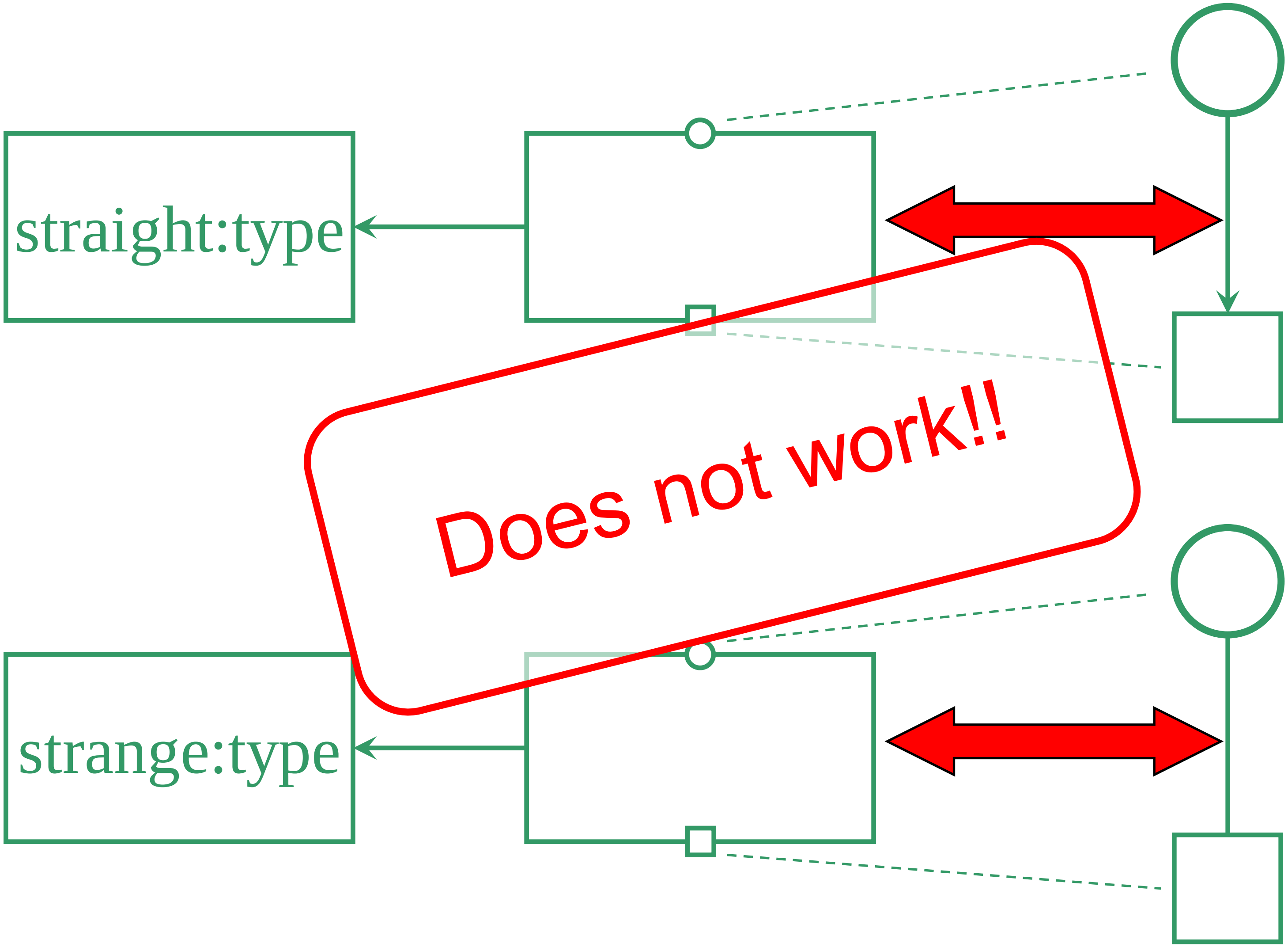
- Negation
  - grammar-style semantics
    (not what we want?)
  - model-driven semantics
    (incrementality lost or incompatible)

Re-usable nodes

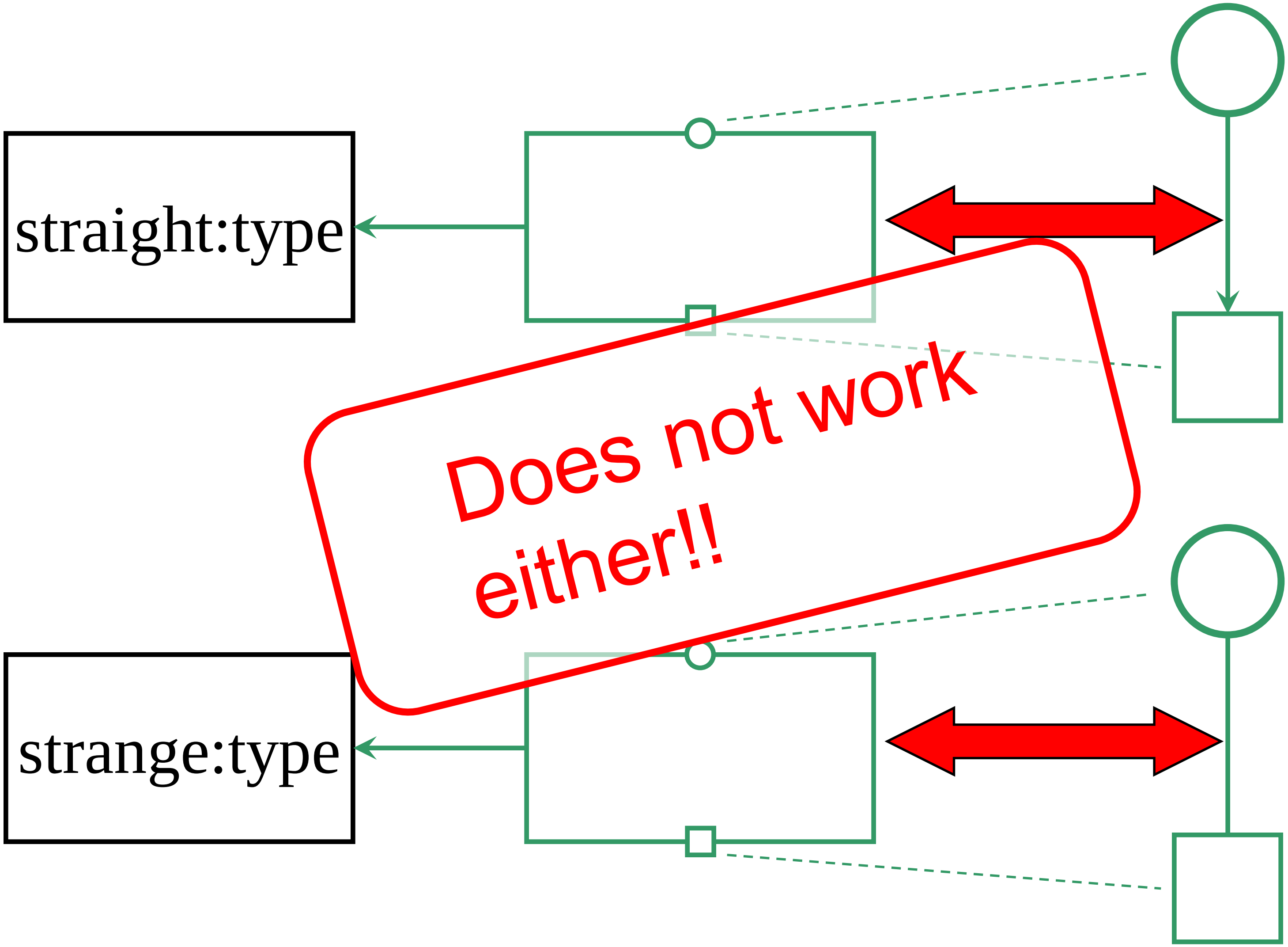


Note that this example uses a changed meta-model (components refer to their type).

Rules?

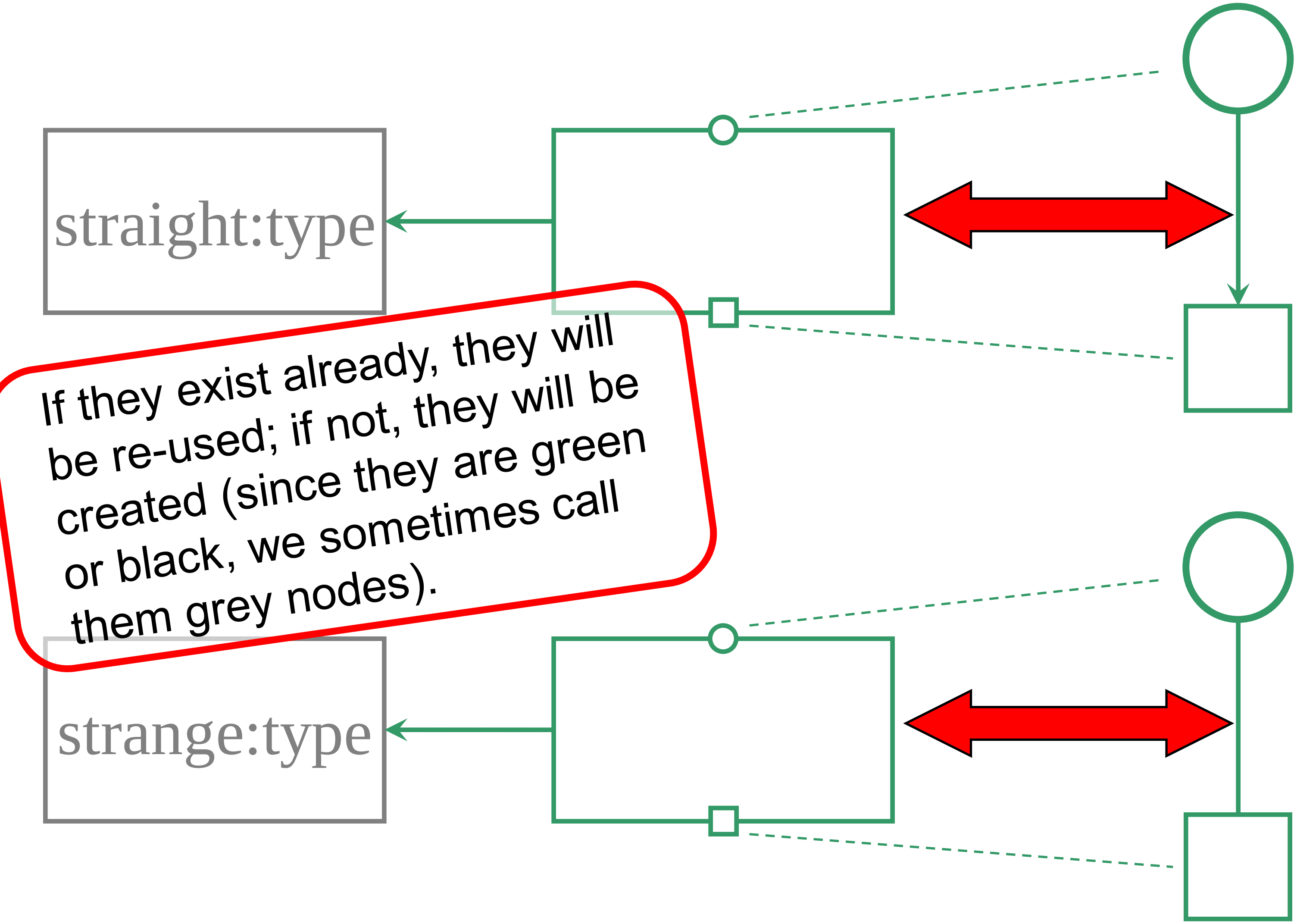


Rules?



Re-usable nodes

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



Extensions

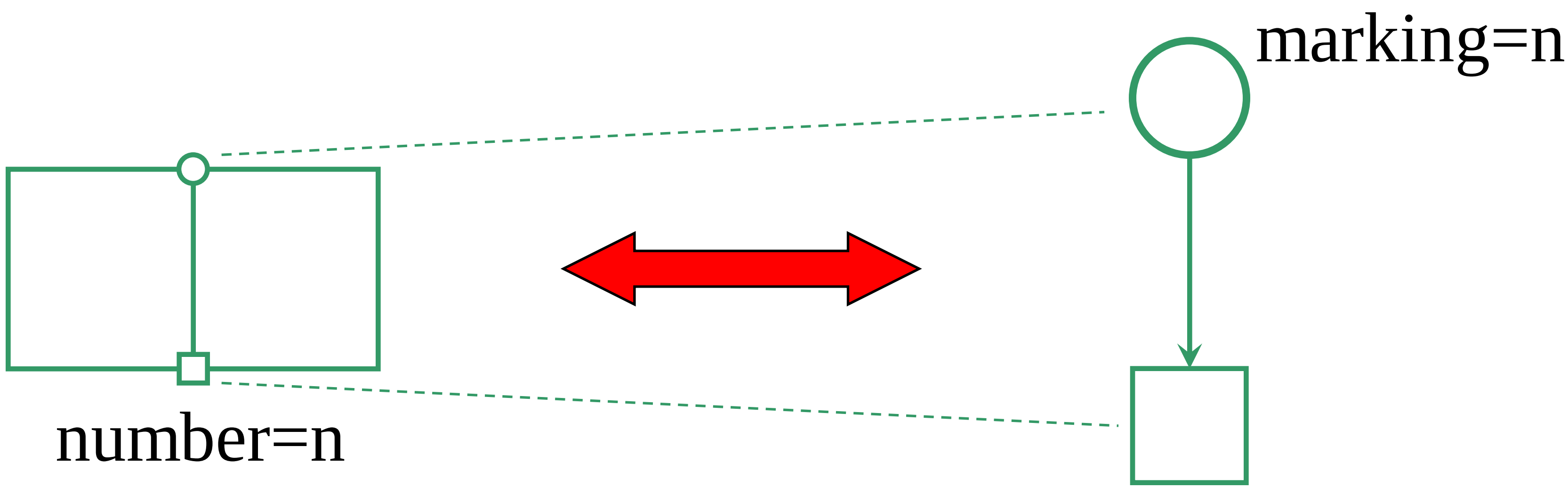
- TGG++
- Negation
- Re-usable nodes (“grey nodes” / ##)

Clean definitions

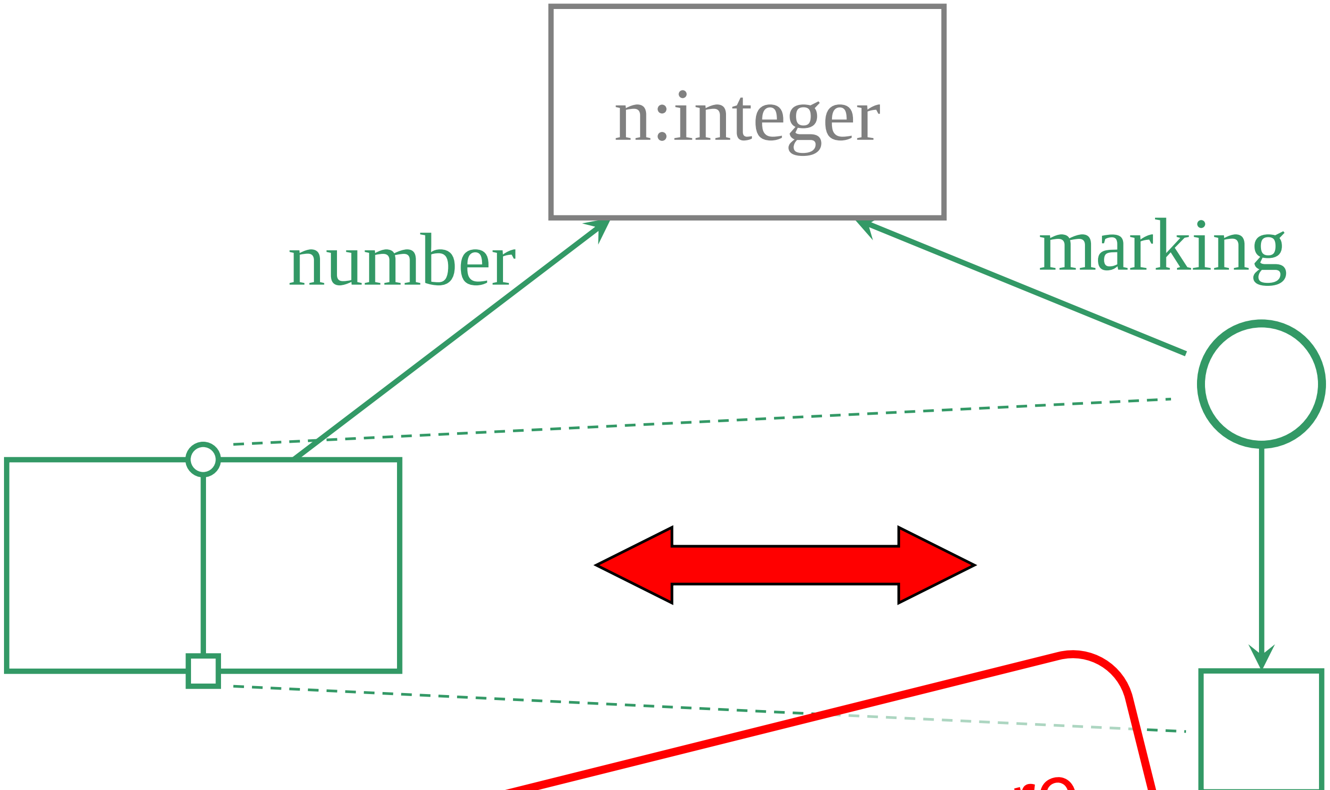
- Attributes

- Inheritance in graph models

Rules in with attributes

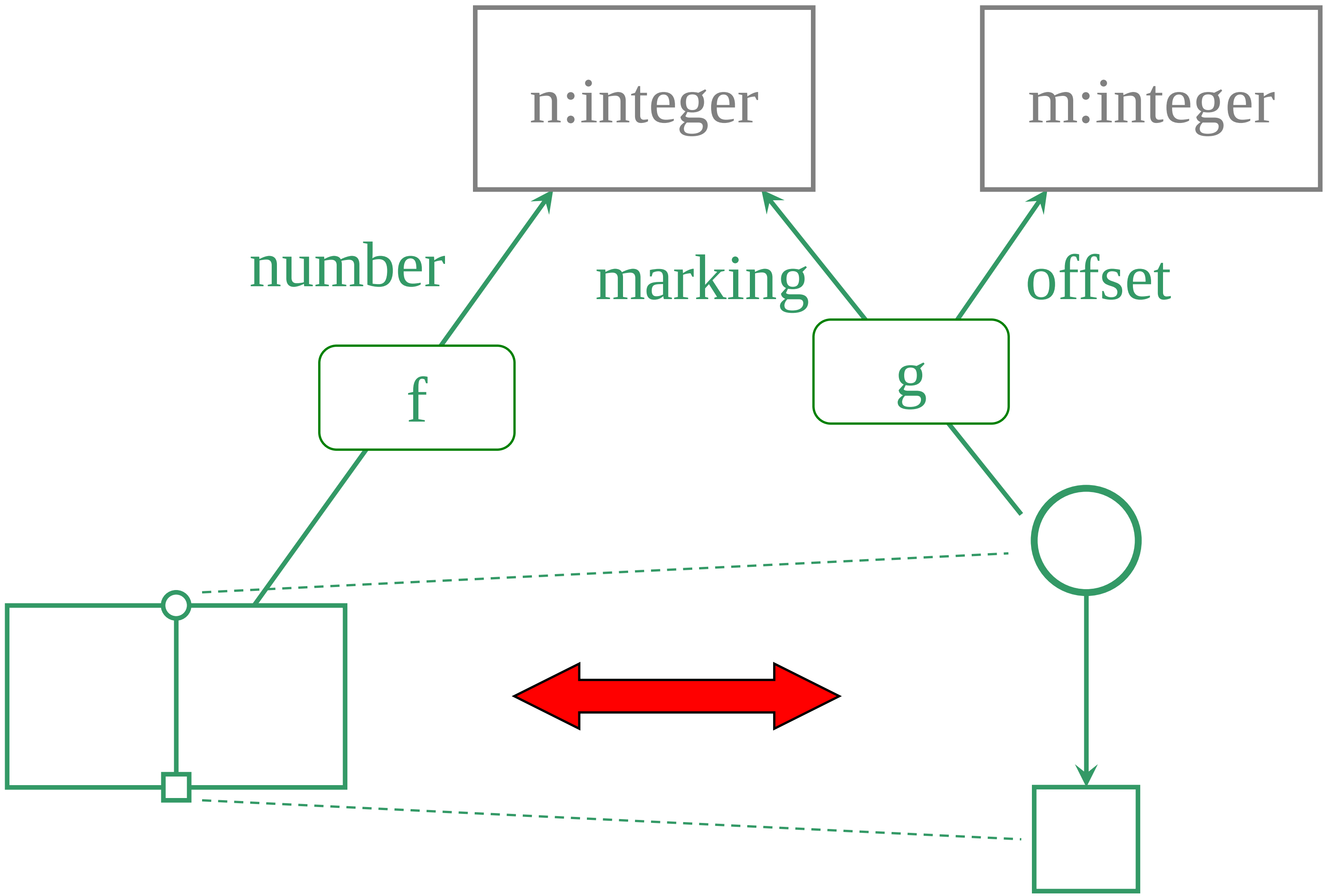


Rules in with attributes



Values of attributes are “grey nodes”!

Attributes: General Concept



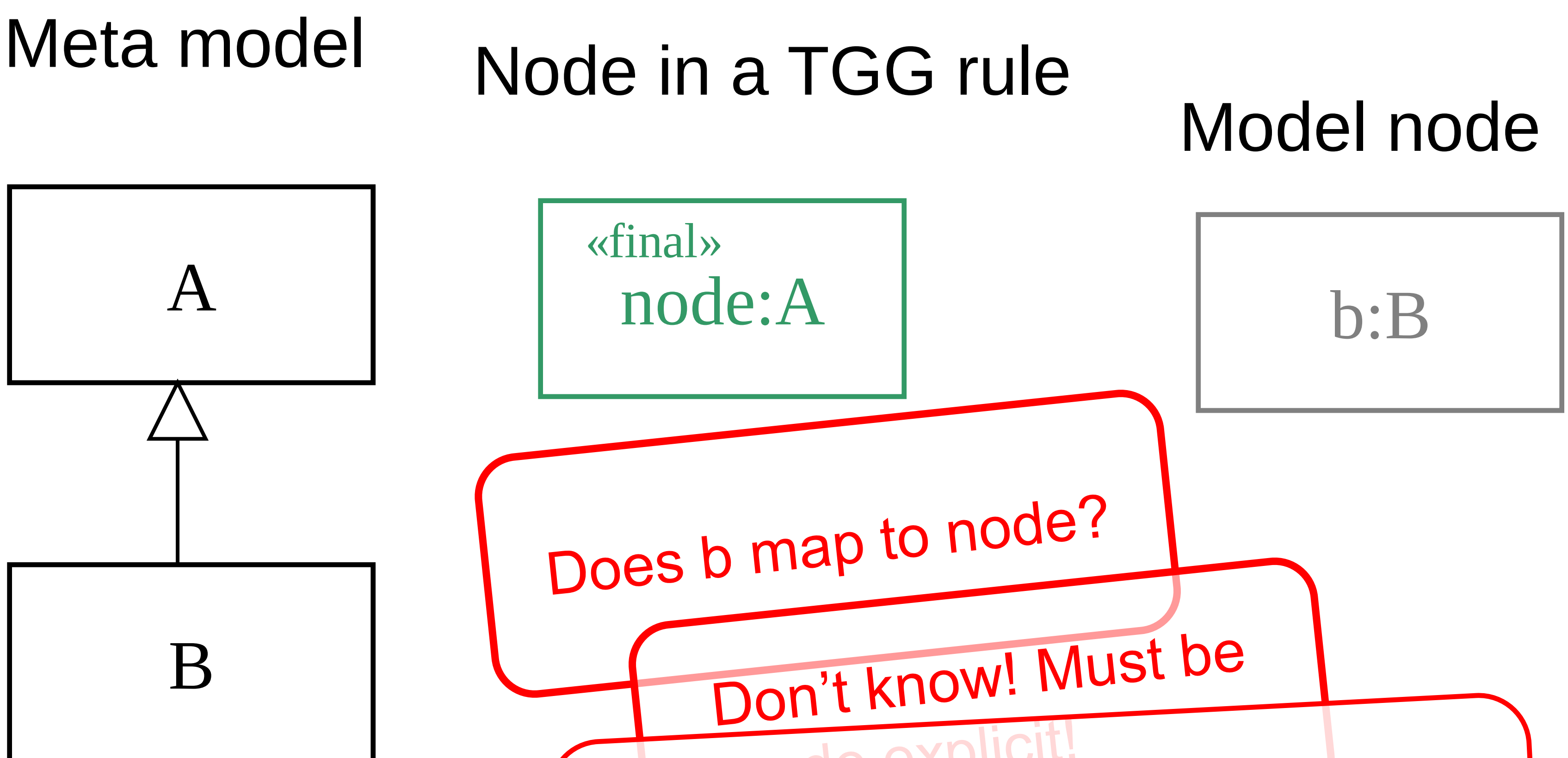
Clean definitions

- Attributes
  - are grey nodes
  - problem: operational interpretation needs inverse functions

- Inheritance in graph models

Inheritance

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



Does b map to node?

Don't know! Must be made explicit!

→ In our tool: the property MatchSubtypes of a node defines, what we want.

Research issues

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Good examples
- Benchmarks
- "Theory" of sufficient conditions for deterministic transformations /
  deterministic "partial transformations"
- Verification techniques
- Uniform interface / integration of strategies
- Efficient transformations / synchronisation
- …

→ Some of the concepts discussed here, are not implemented in the TGG interpreter we use in our tutorial. Values to attributes are assigned via constraints (see examples in tutorial).

TGGs: Summary

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- (Often) elegant way of defining the relation between two kinds of models
- Based on this definition, models can be
  - transformed in either direction
    (different approaches: compile rules, interprete rules)
  - corresponding models can be kept consistent
    (synchronization)
- Good for defining the relation between structurally similar models

TGGs: Literature

1. A. Schürr. *Specification of graph translators with triple graph grammars*. In E. W. Mayr, G. Schmidt, and G. Tinhofer, editors, Graph-Theoretic Concepts in Computer Science, 20th International Workshop, WG '94, Springer LNCS 903, 151-163, June 1994.

2. E. Kindler, R. Wagner: *Triple Graph Grammars: Concepts, Extensions, Implementations, and Application Scenarios*. Technical Report, Department of Computer Science, University of Paderborn, tr-ri-07-284, June 2007.

→ We did NOT invent TGGs (that was Andy Schürr more than 20 years ago)

→ Due to their nice concepts we are enthusiastic about them anyway and try to promote them.