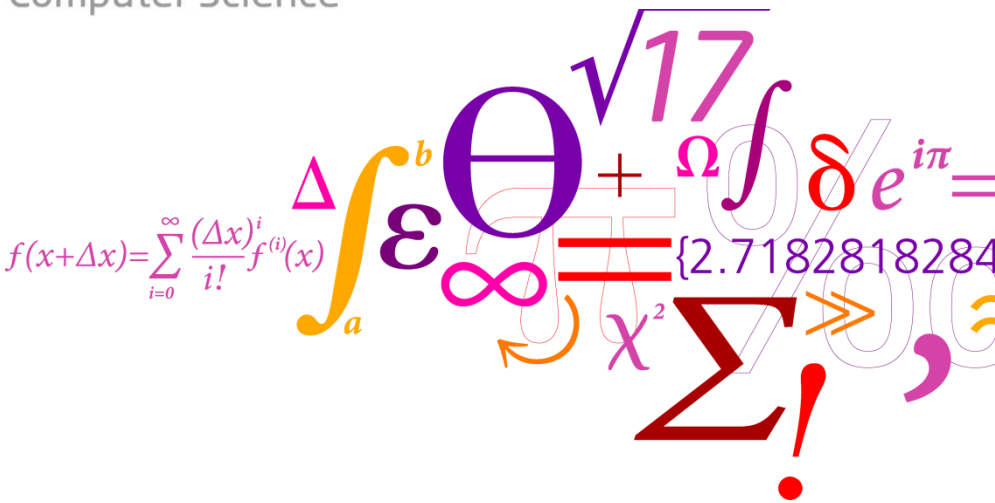
Advanced Topics in Software Engineering (02265)

Ekkart Kindler

DTU Compute
Department of Applied Mathematics and Computer Science



Recapitulation TGGs

Note that slides 3-41 were provided for lecture 6 already and slides 3-19 have been discussed in lecture 6 already (at least briefly).

Grammars

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

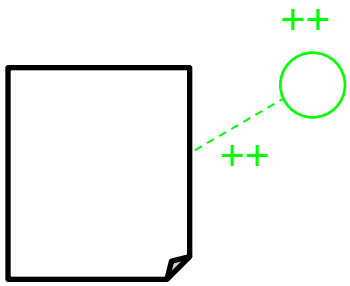
- Grammar:
  - rules (+ axiom)
  - replacing left-hand side by right-hand side
- Grammars are meta-models
  (for textual modelling notations)
- The two ”uses” of grammars
  - Use 1: defining a language (→ parsing, etc.)
  - Use 2: defining behaviour (→ Markov algorithms)

Graph grammars: Use 1

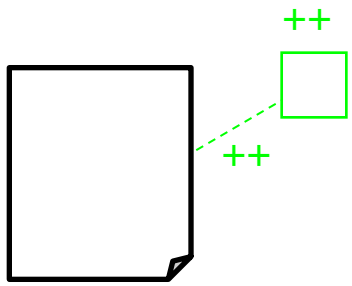
This graph grammar defines the syntax of Petri nets. It can be used to generate or parse a syntactically correct Petri net.

Defining the syntax of Petri nets

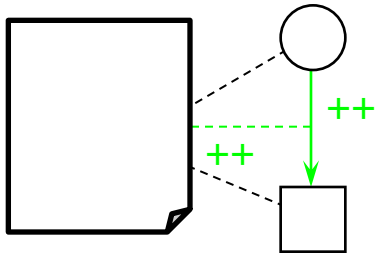
Rule 1:



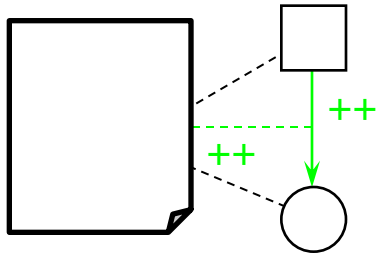
Rule 2:



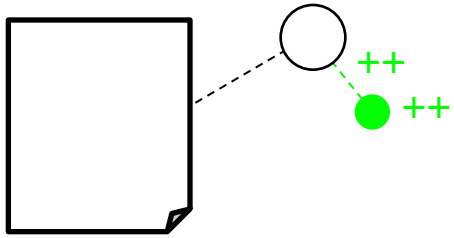
Rule 3:



Rule 4:



Rule 5:

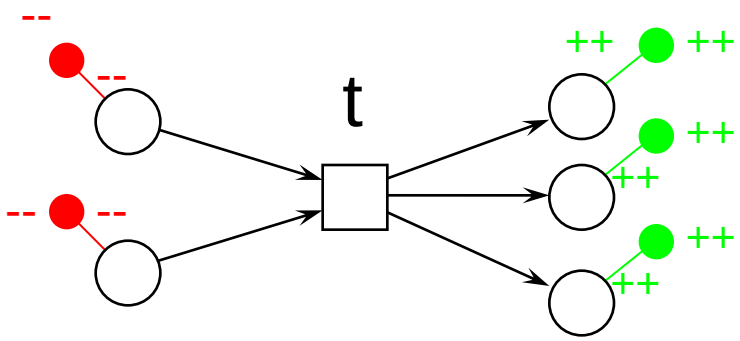


Note that this is not the (main) purpose of GGs here; the example should just illustrate the “Use 1” of GGs.

Graph grammars: Use 2

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

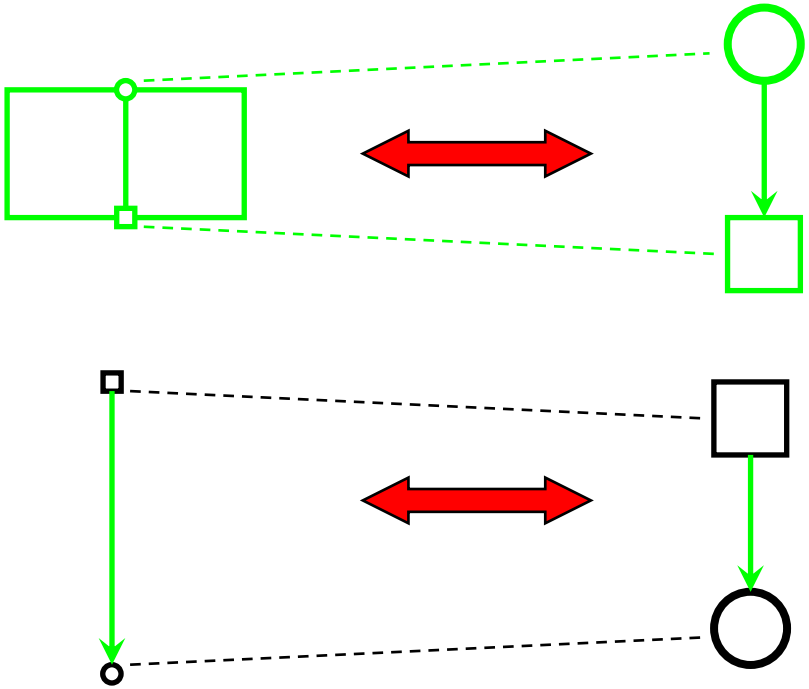
Different representation: single graph, indicating in colours (and labels) what does not change, what is deleted and what is added:



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

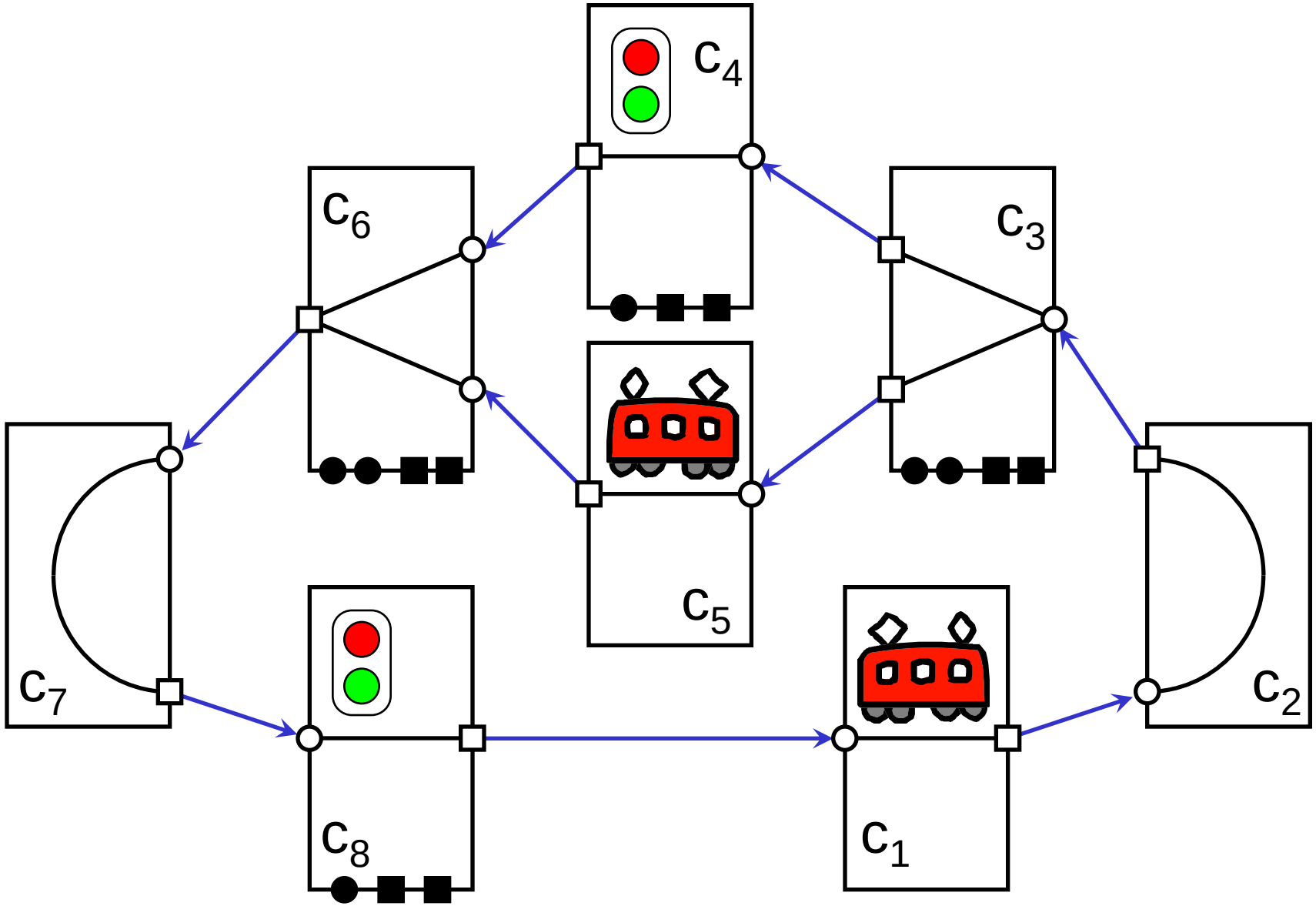
DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

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



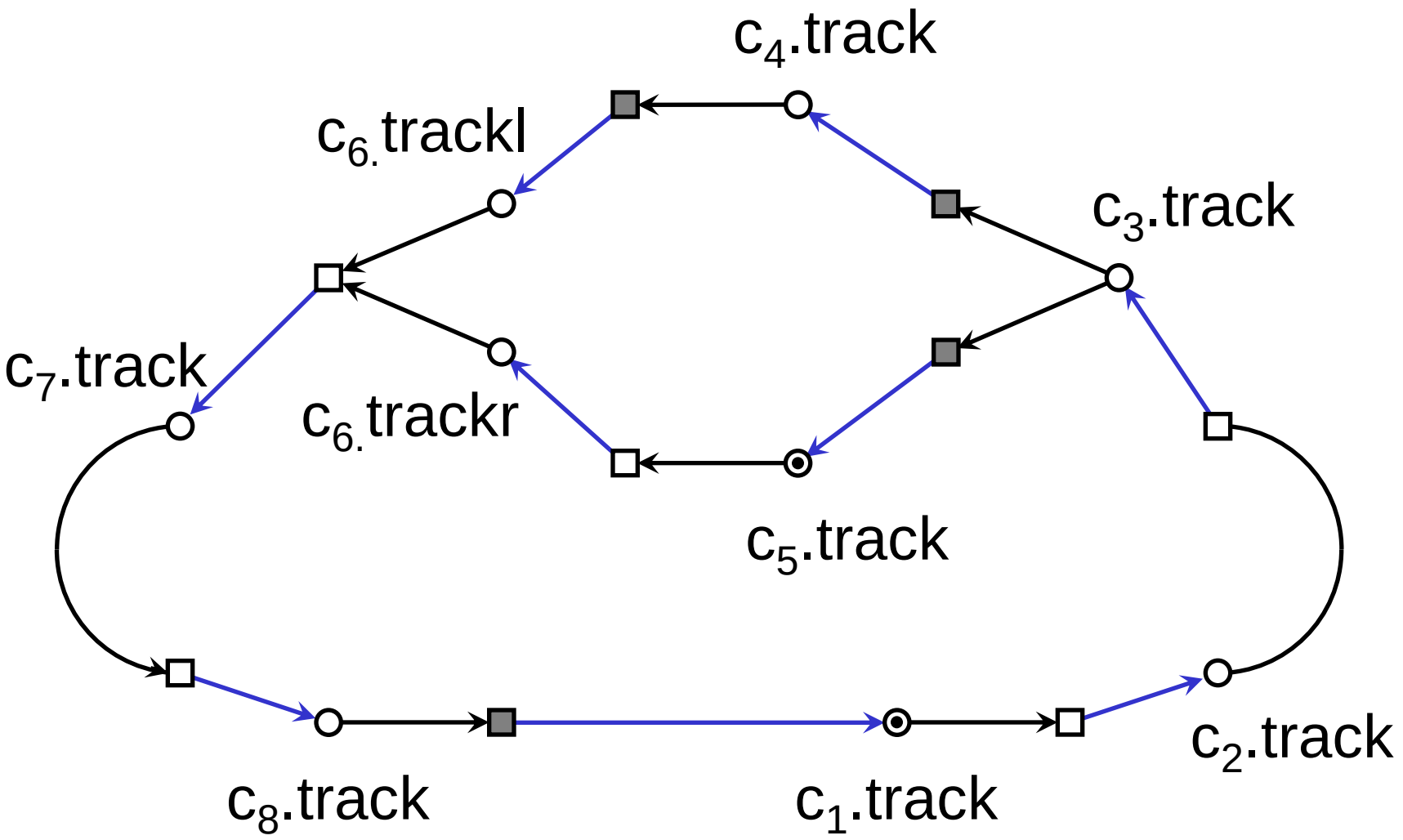
Example: From Project Plan …

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



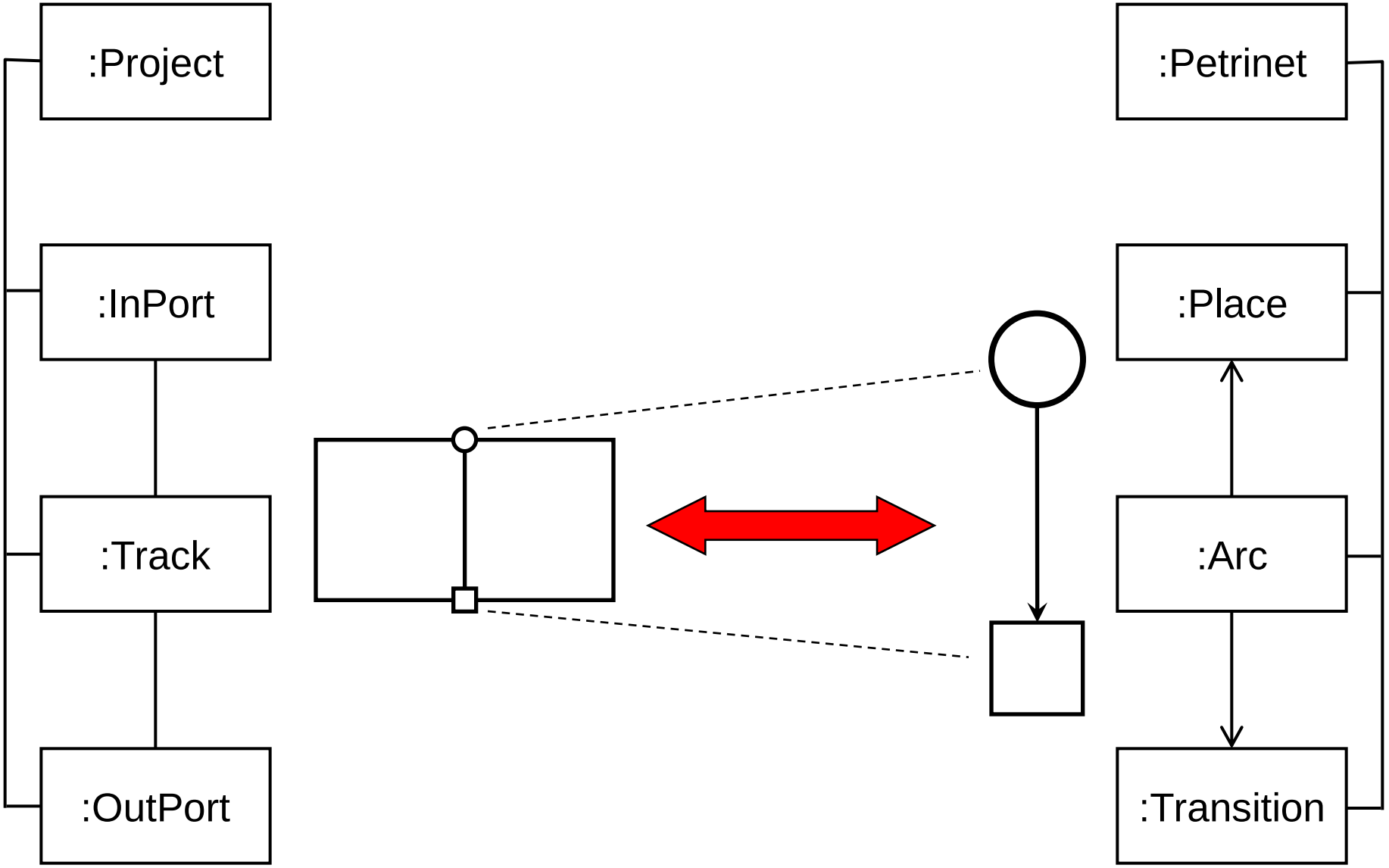
Example: … to Petri Net

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



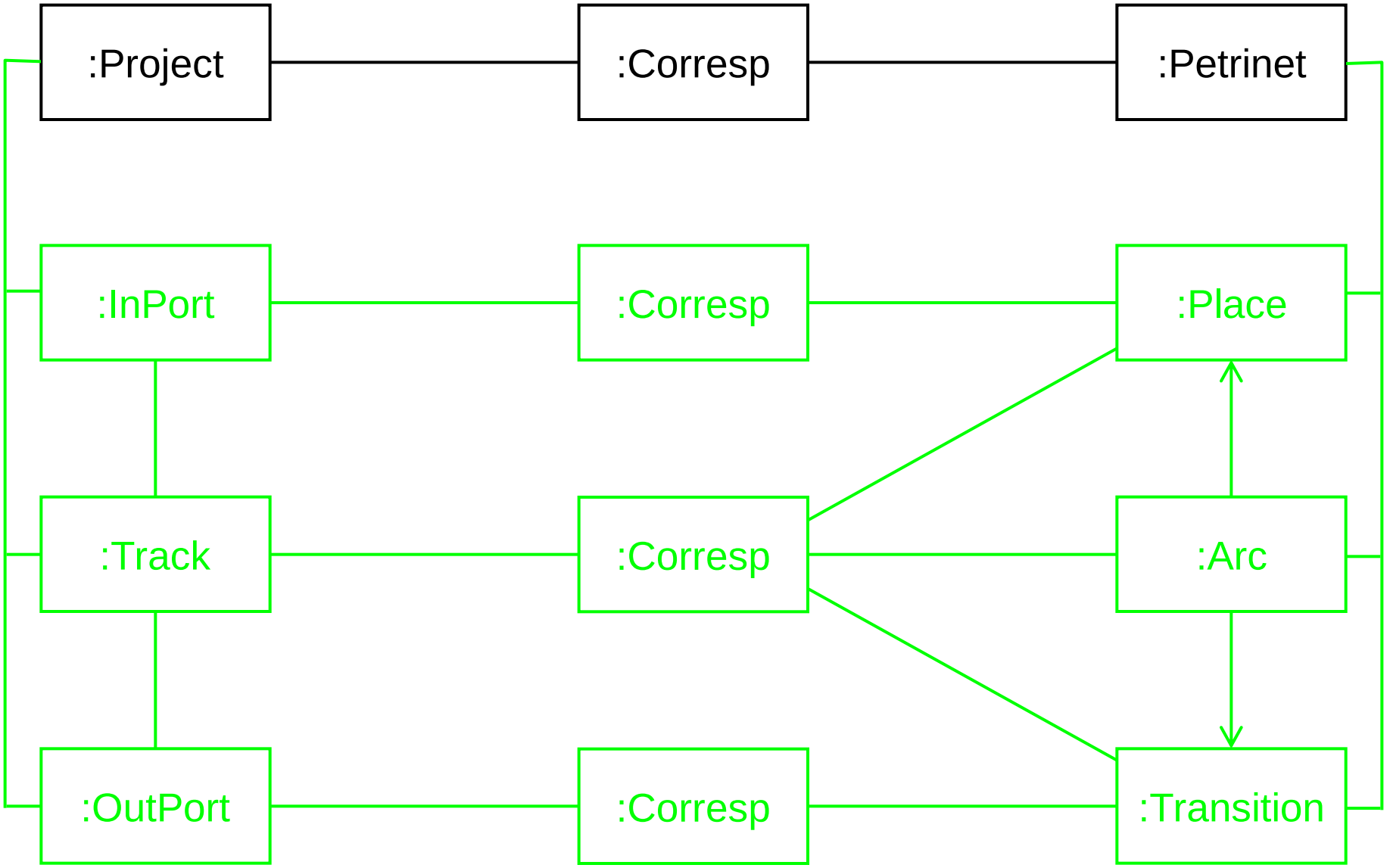
Transformations

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



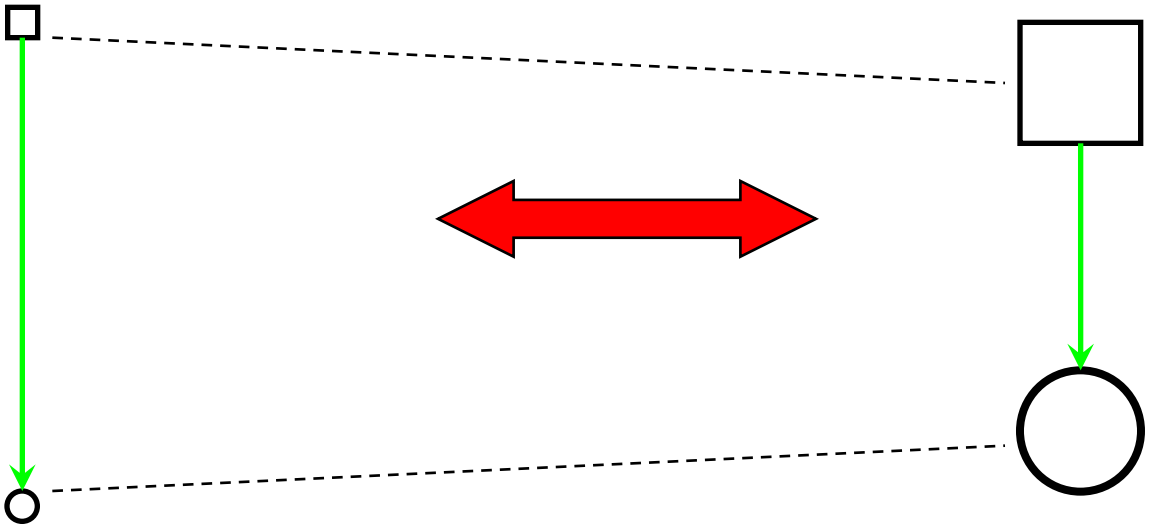
Triple Graph Grammar Rule

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



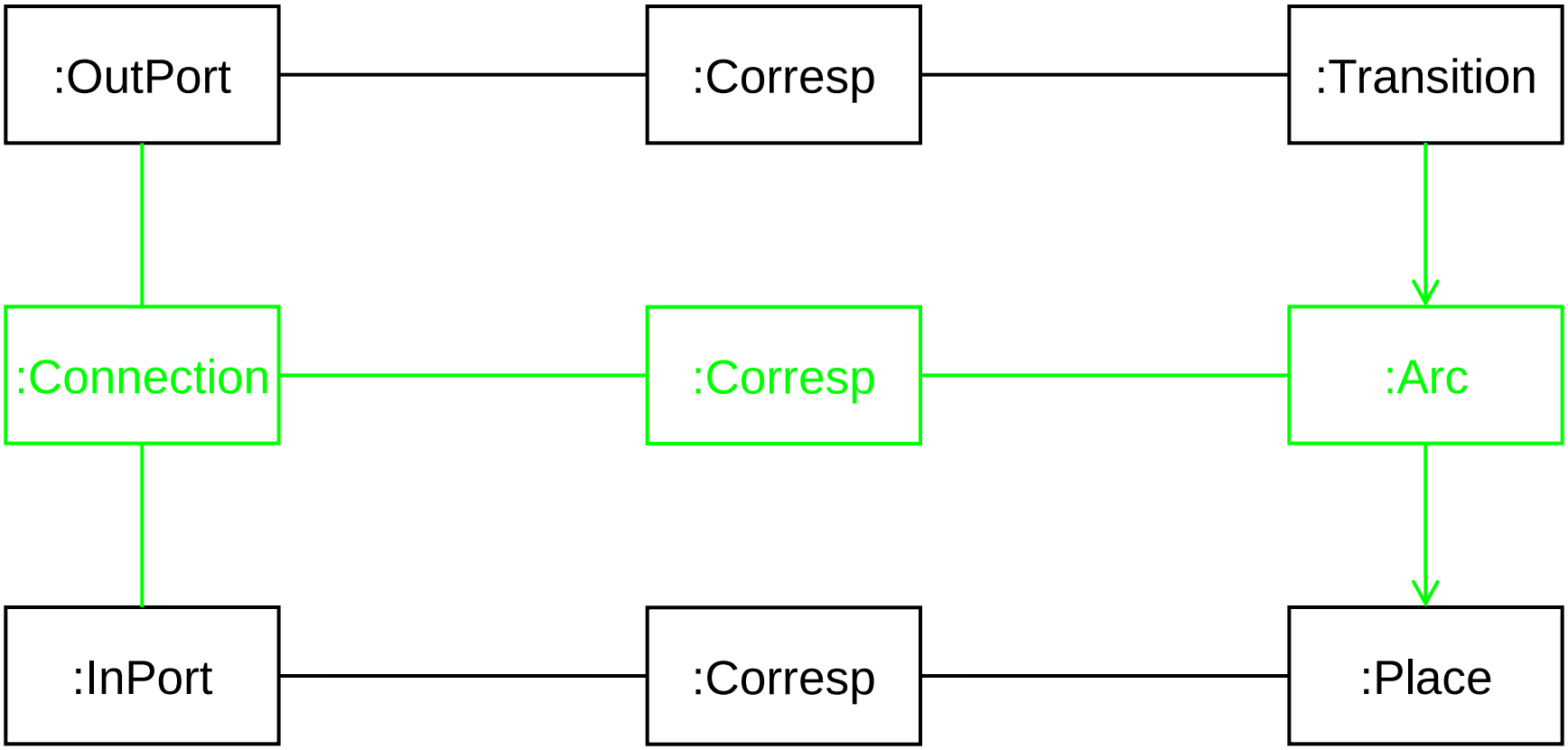
Transf. of connection

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



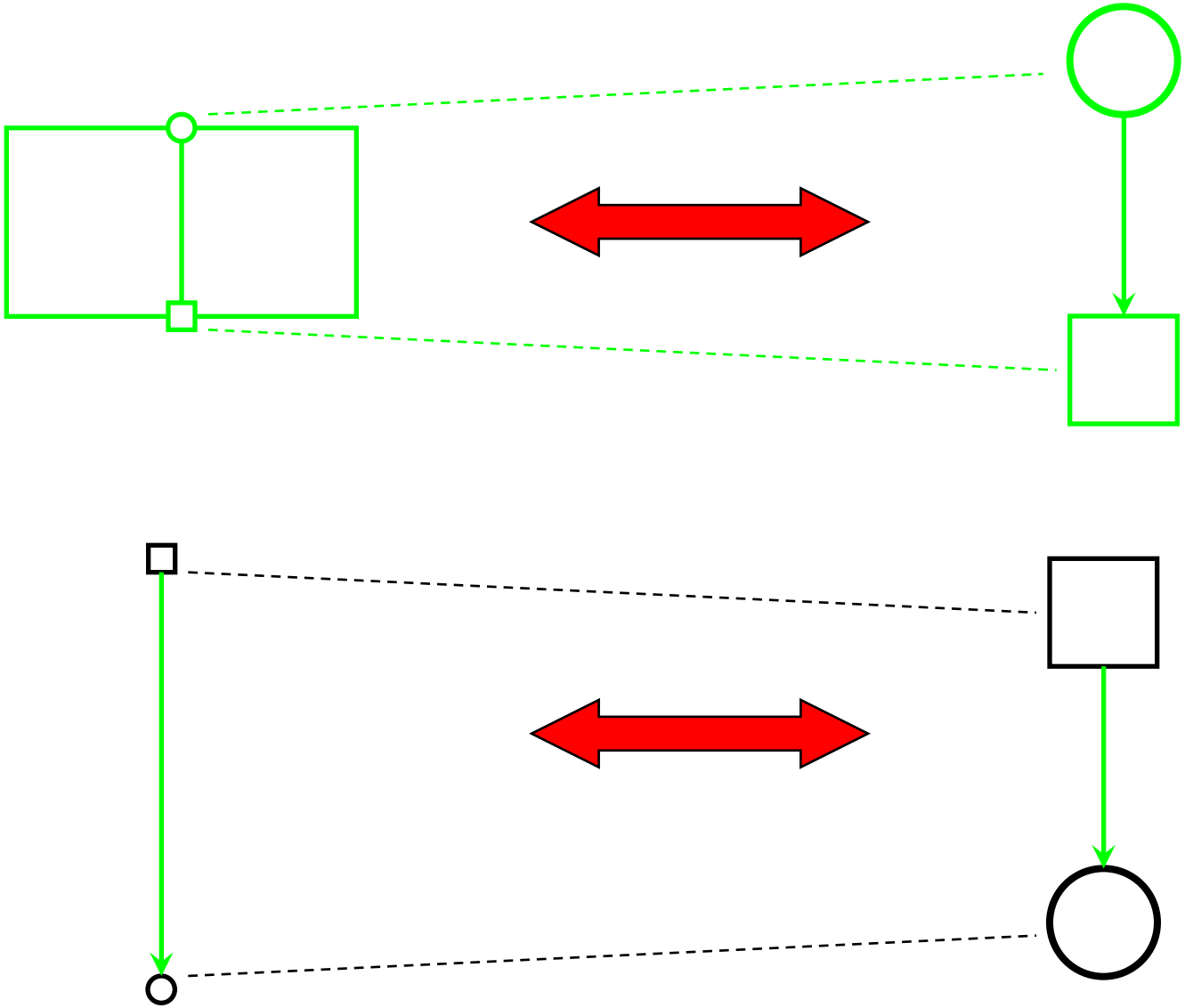
TGG-rule: Connection

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



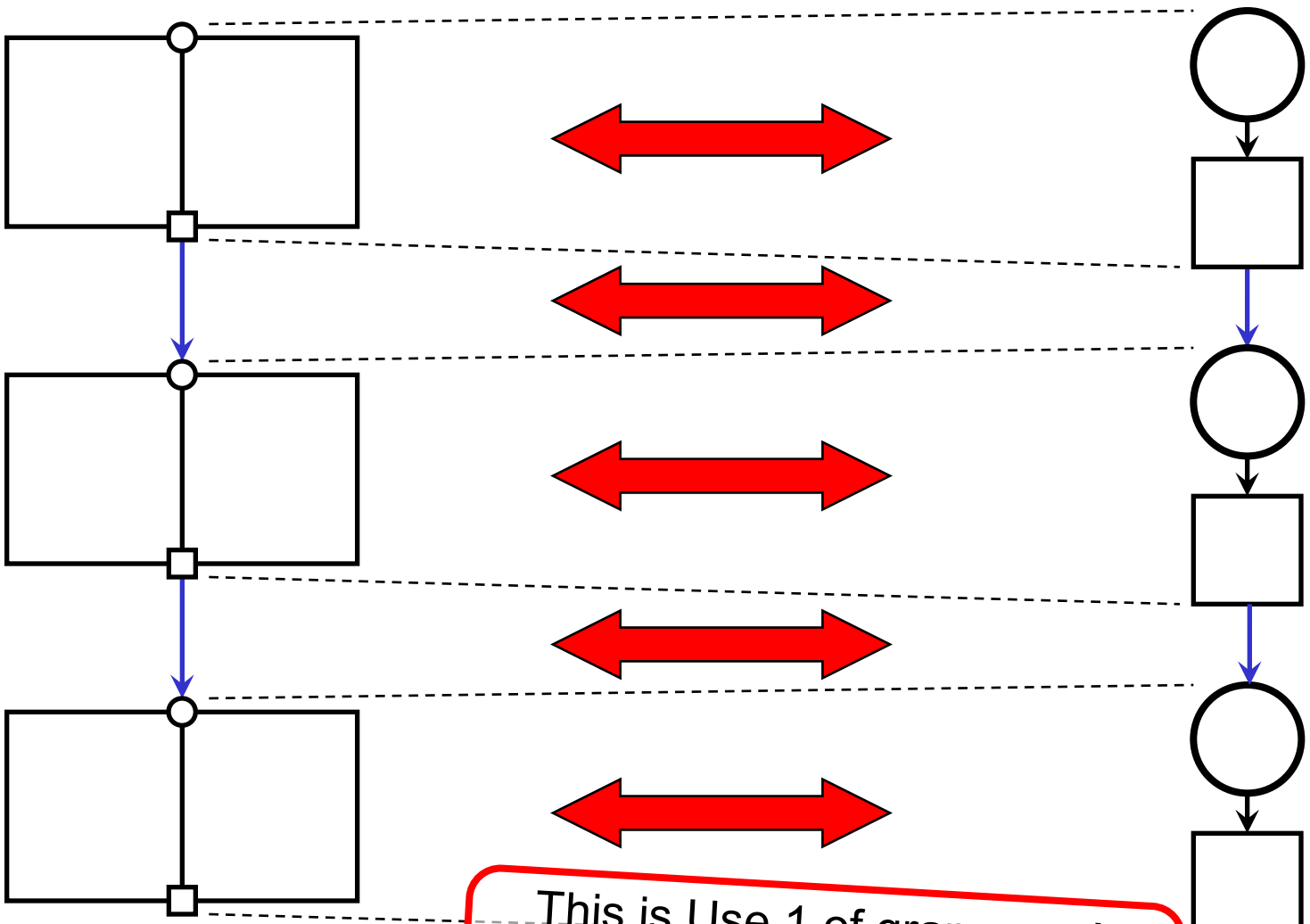
TGG-Rules in “graphical syntax”

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



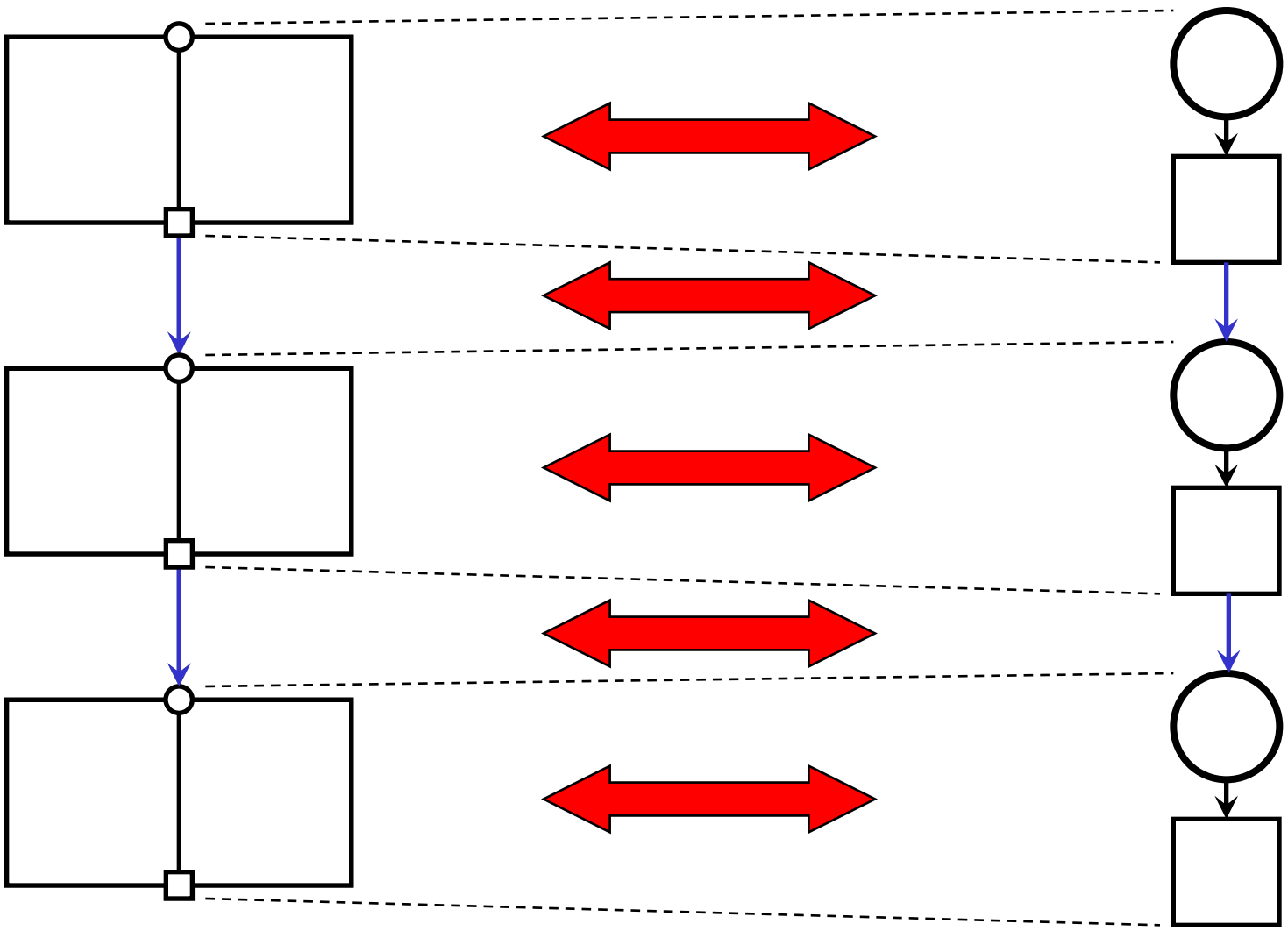
Semantics of TGG-Rules

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



“Model-driven Execution”

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



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 both ways!

Corollary of locality

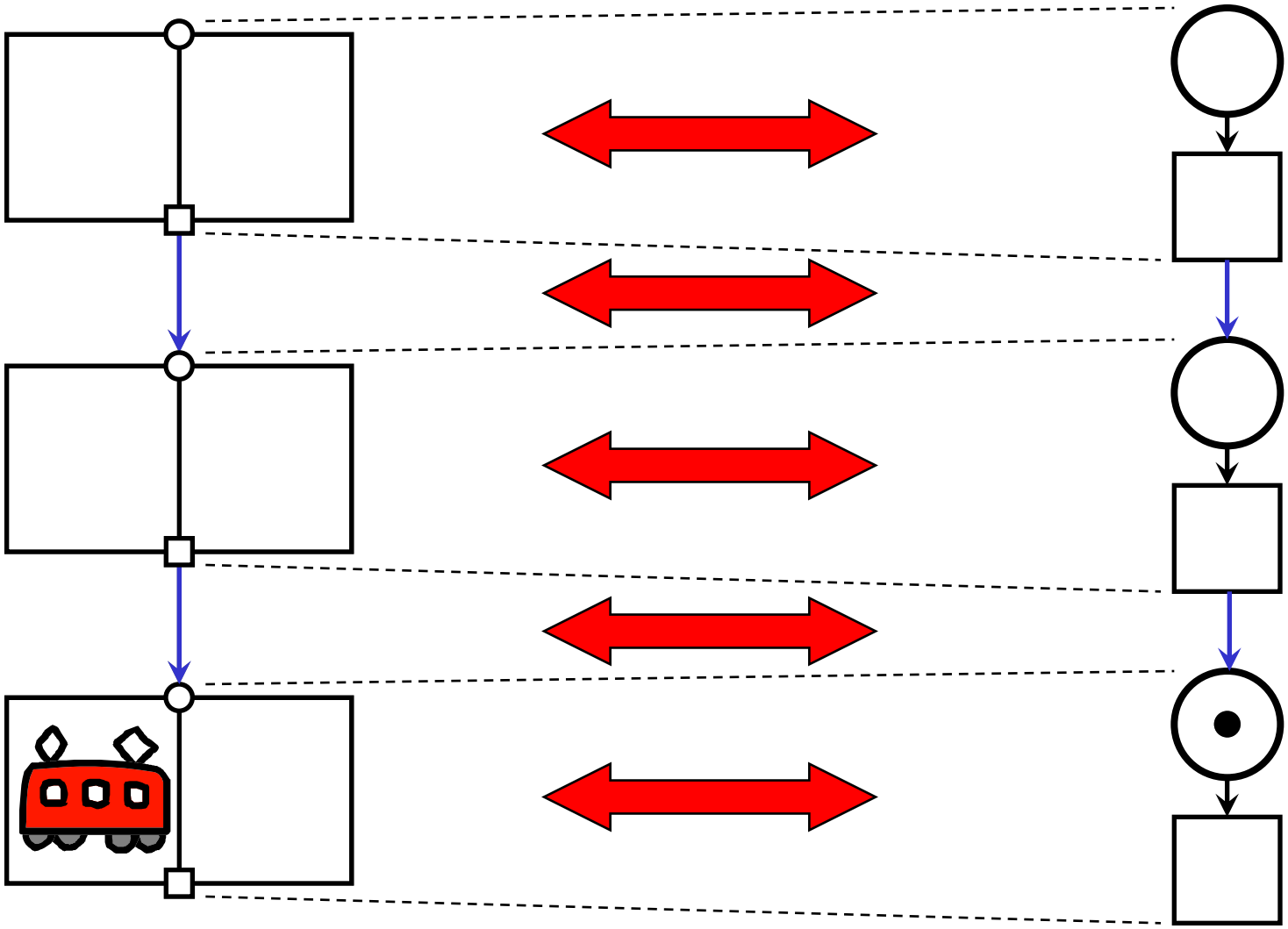
DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- 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

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Defining transformations between models that are structurally similar

- Executing these transformations (in models of reasonable size)

Outline

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

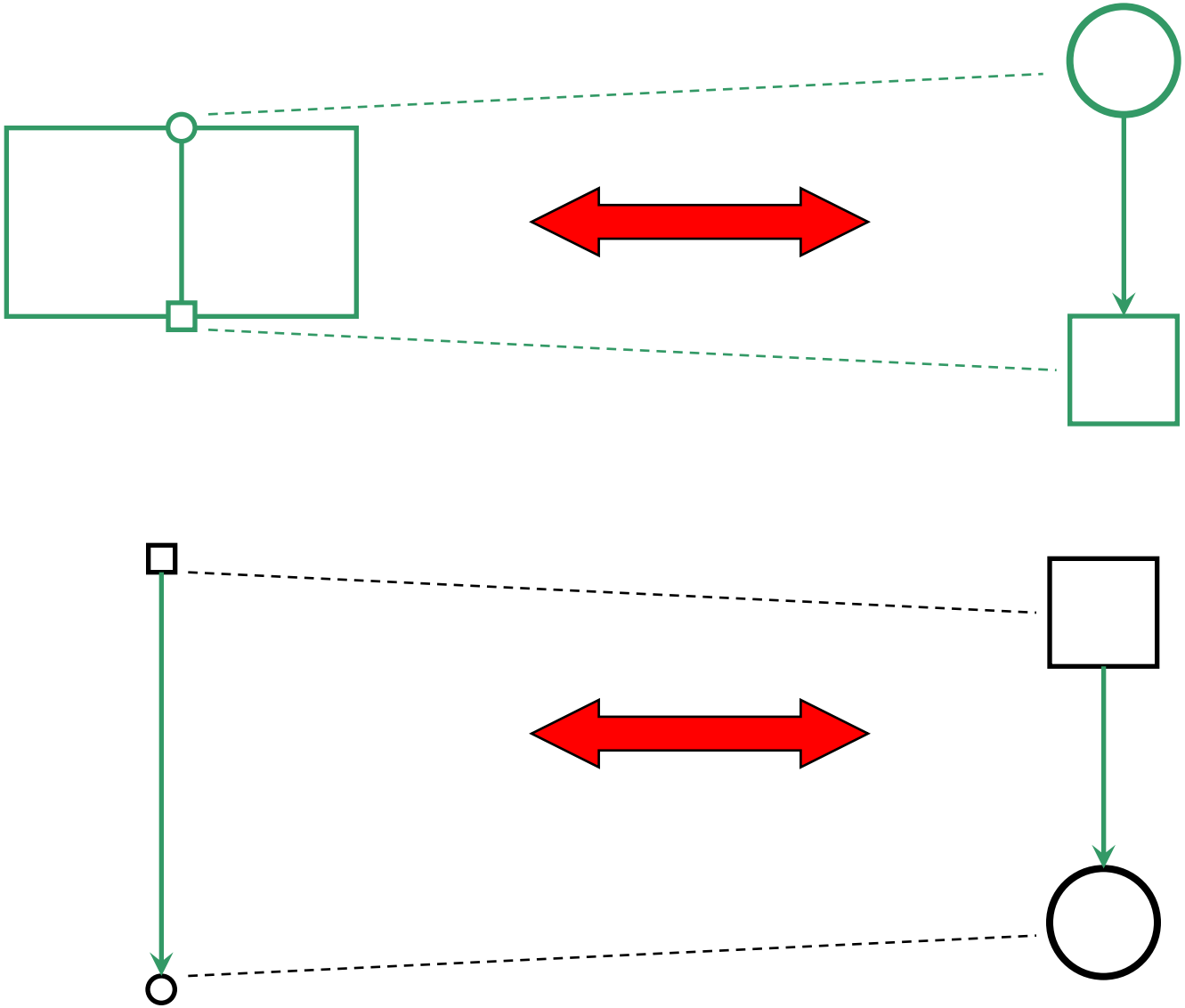
Extensions

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

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

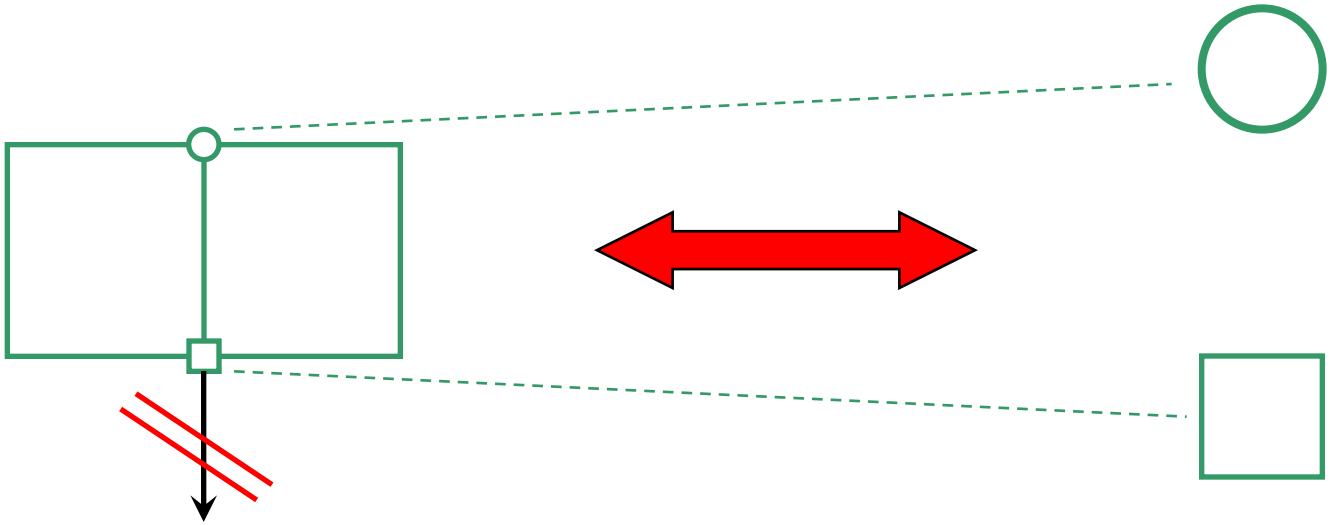
TGG-Rules in “graphical syntax”

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



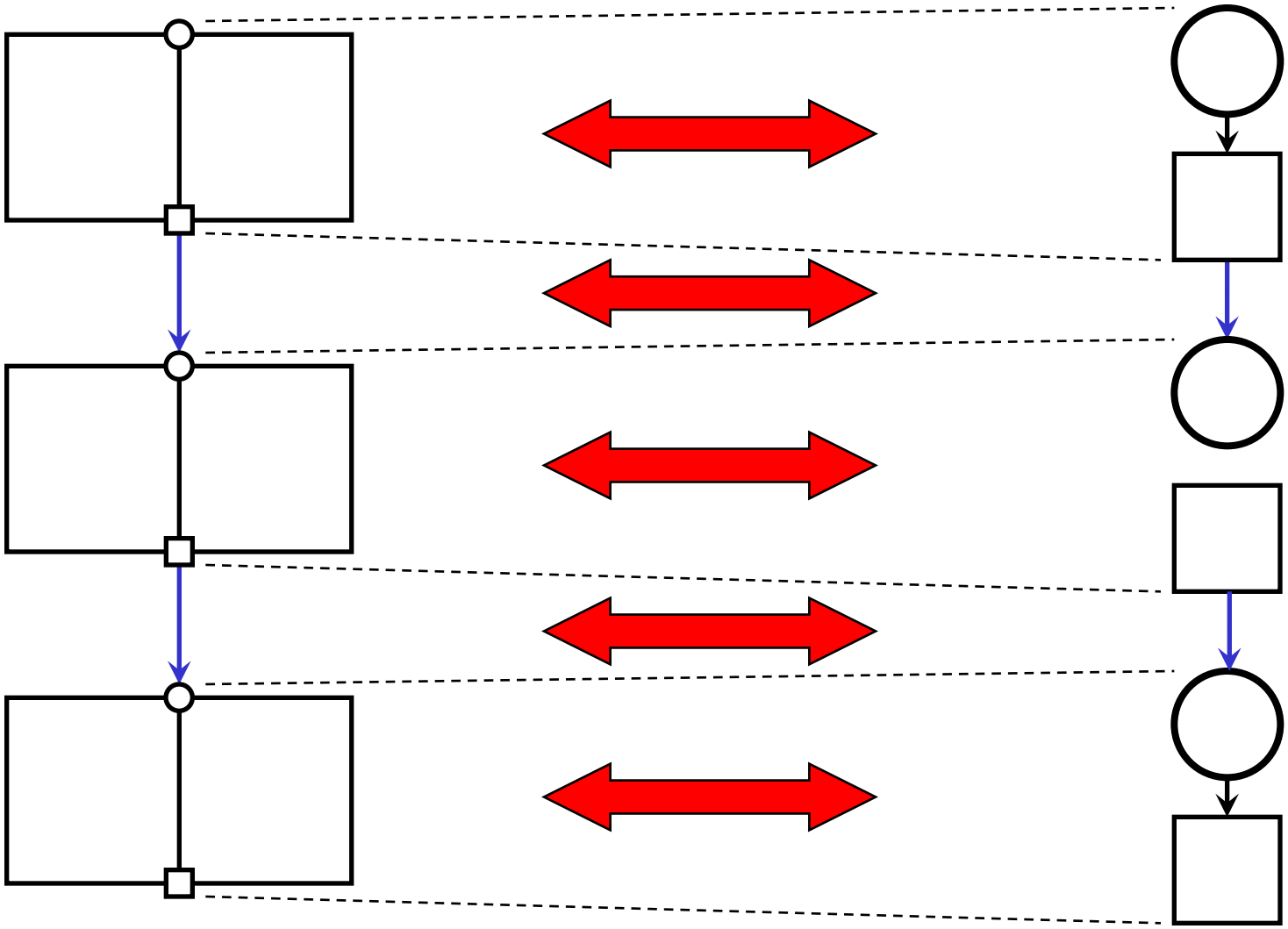
TGG-Rules in “graphical syntax”

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



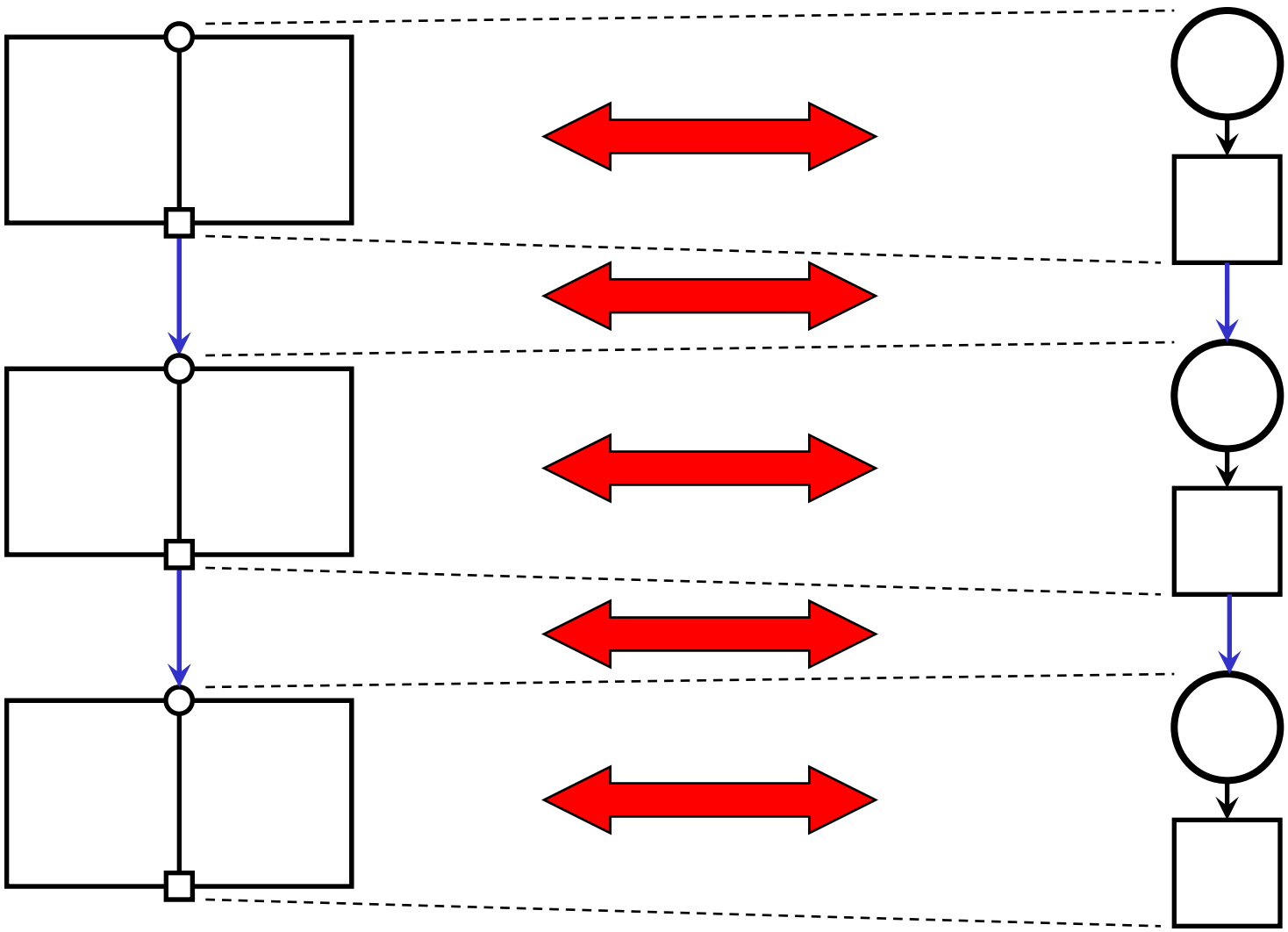
Generation Semantics

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



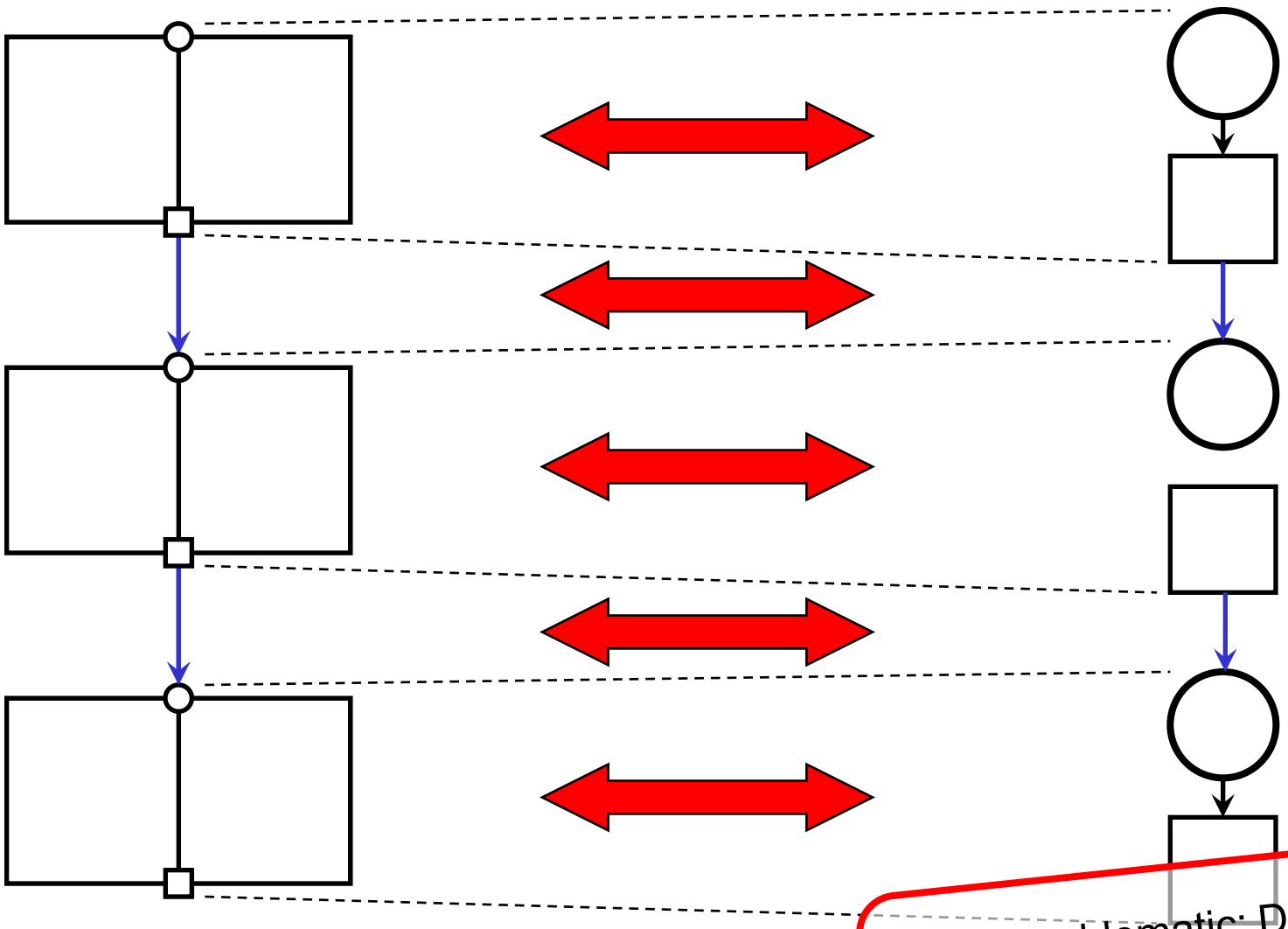
“Model-driven Semantics”

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



“Incremental approach”

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



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

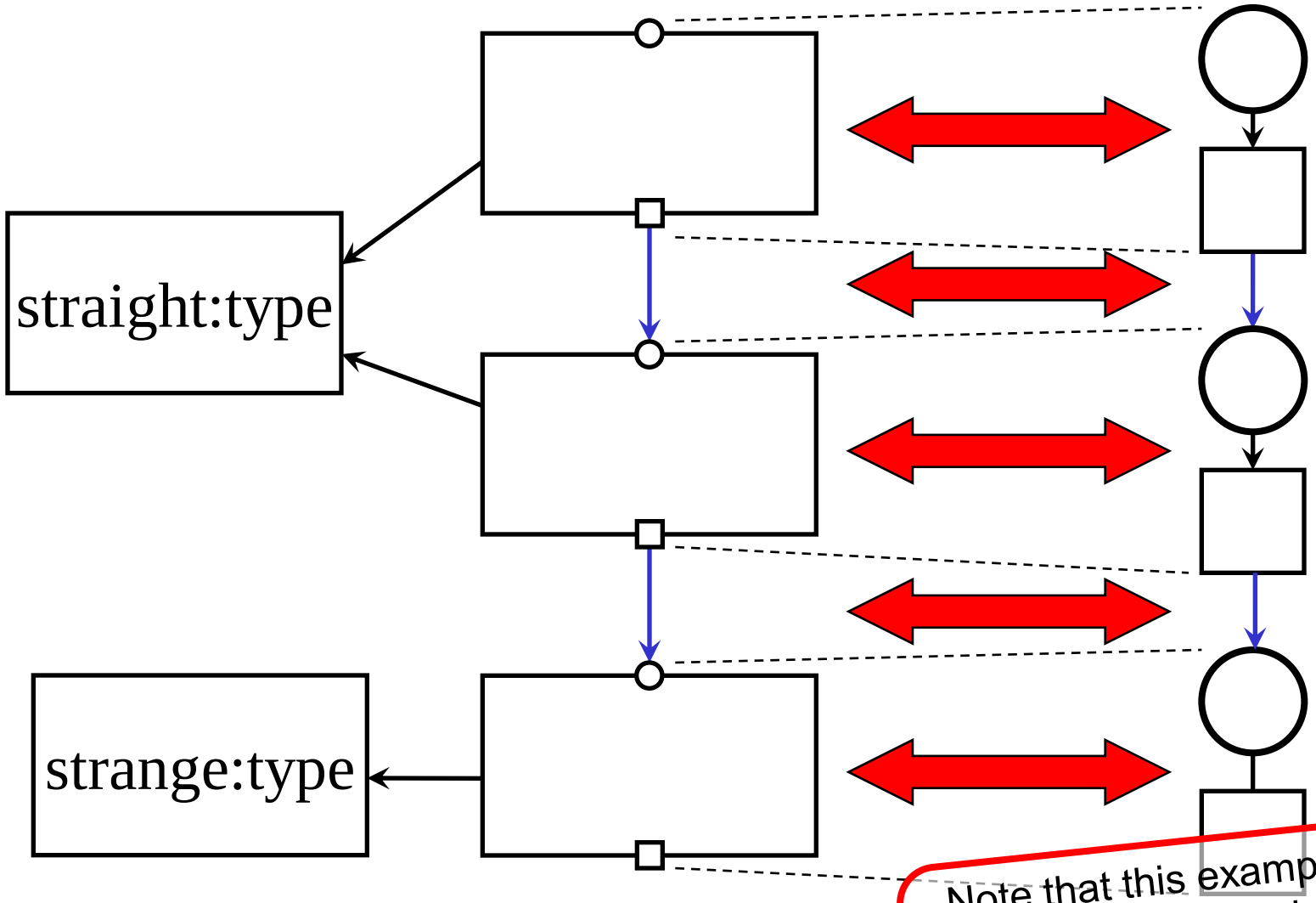
Extensions

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- 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

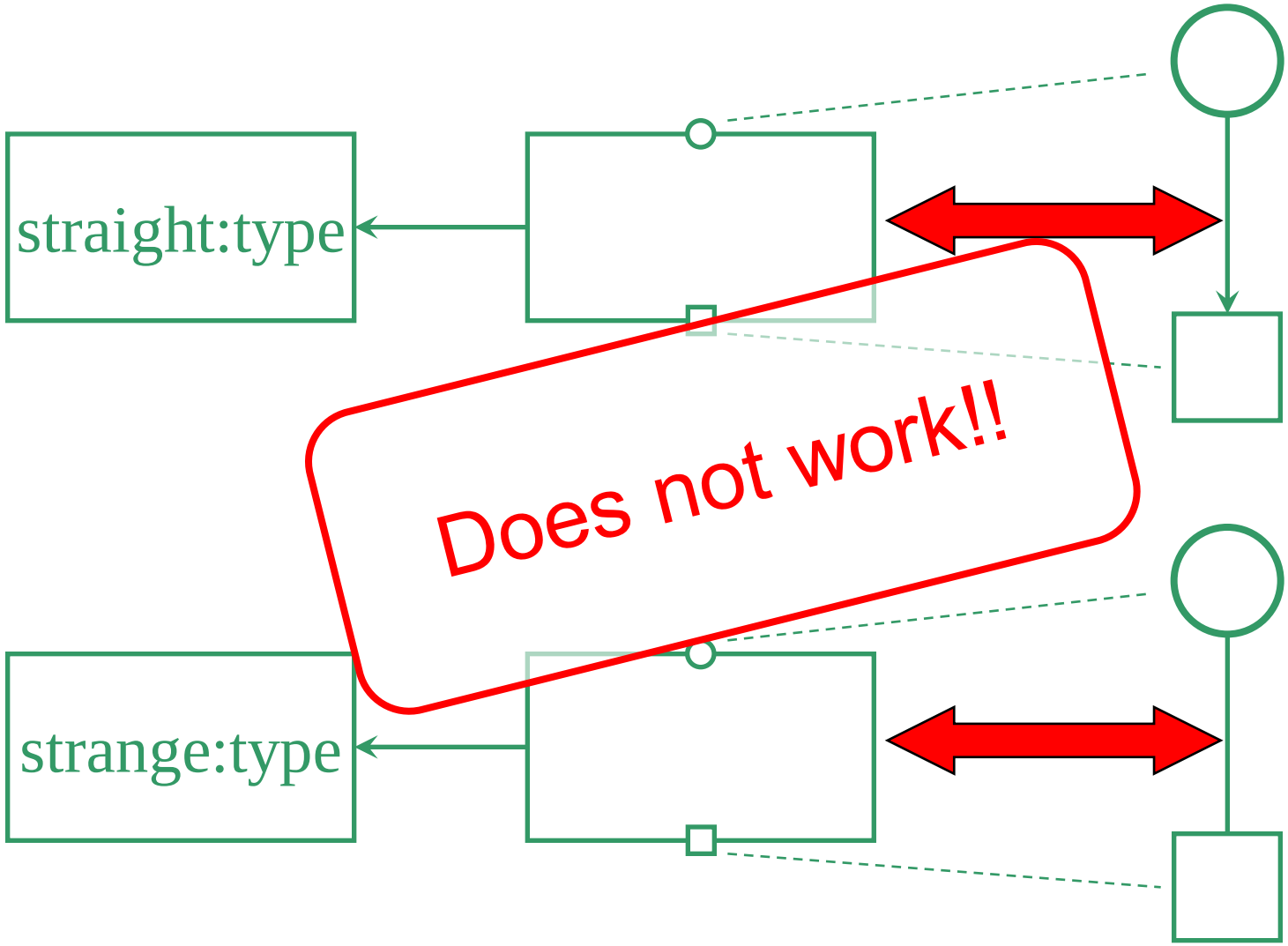
DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



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

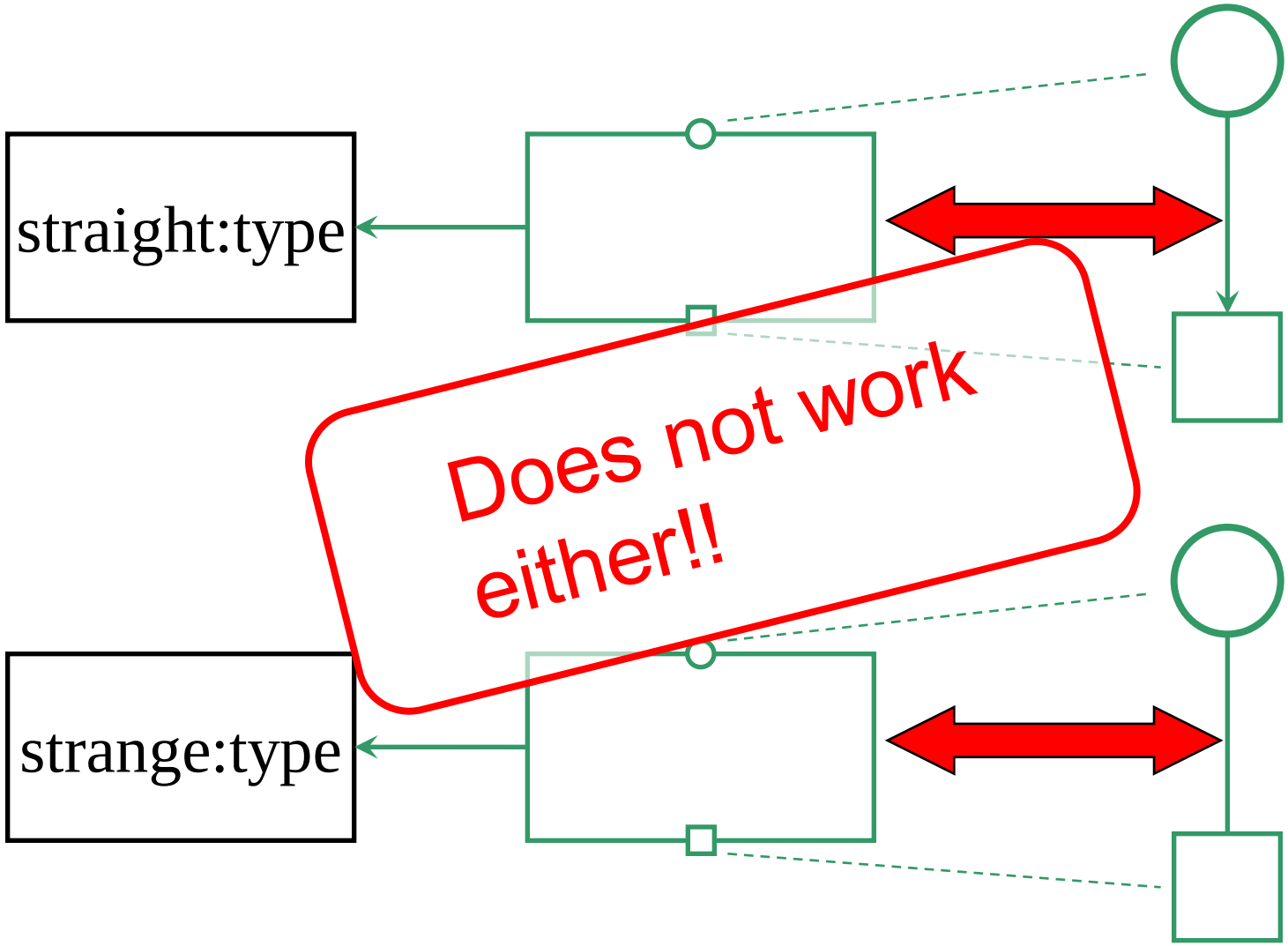
Rules?

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



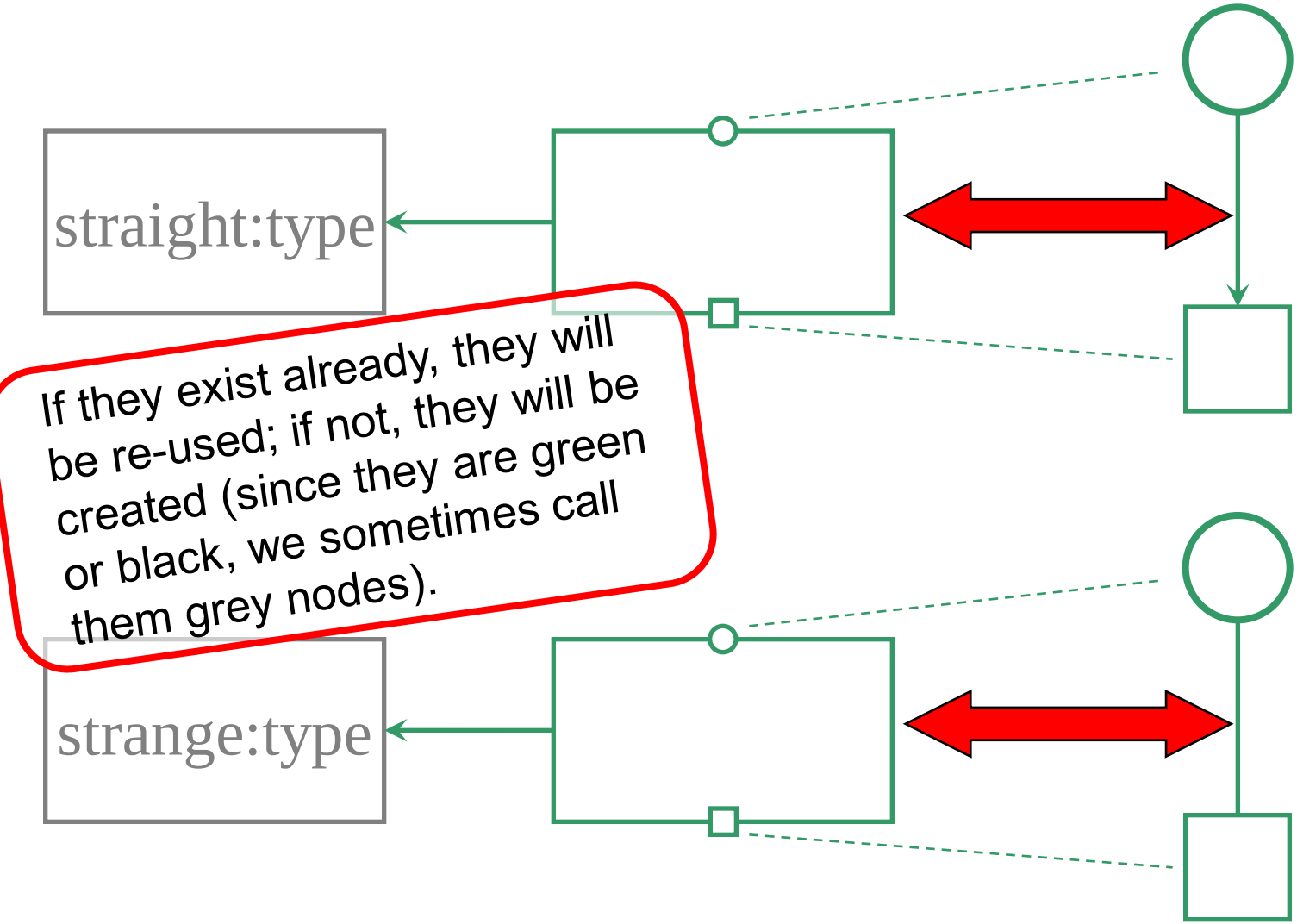
Rules?

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



Re-usable nodes (##)

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



If they exist already, they will be re-used; if not, they will be created (since they are green or black, we sometimes call them grey nodes).

Extensions

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

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

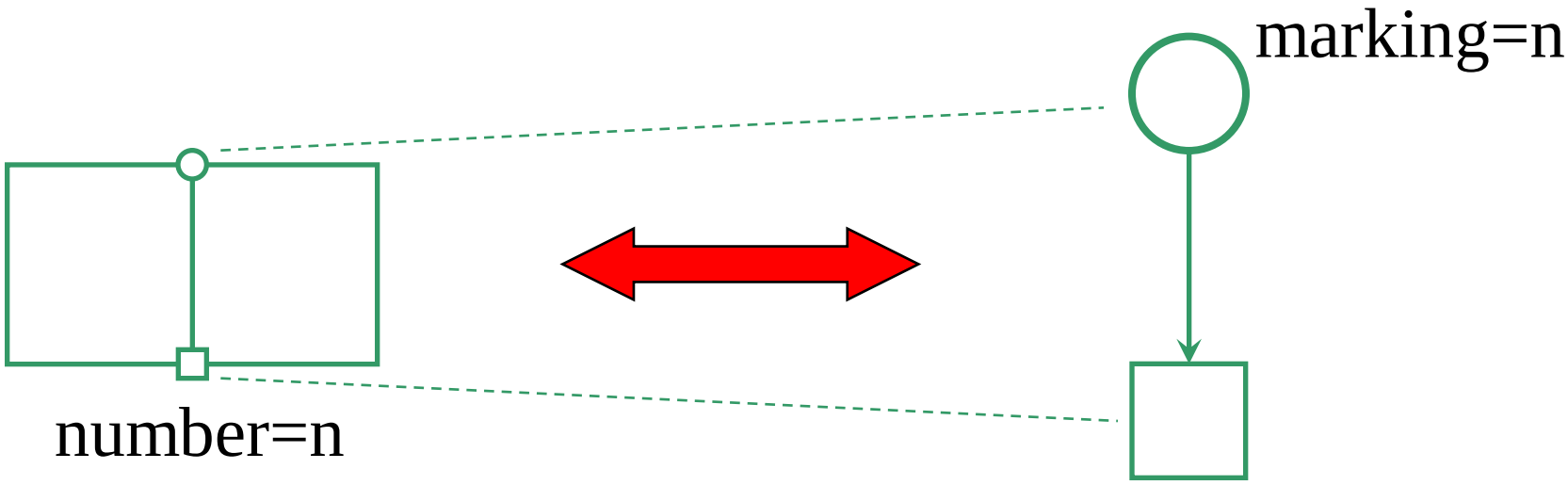
Clean definitions

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Attributes
- Inheritance in graph models

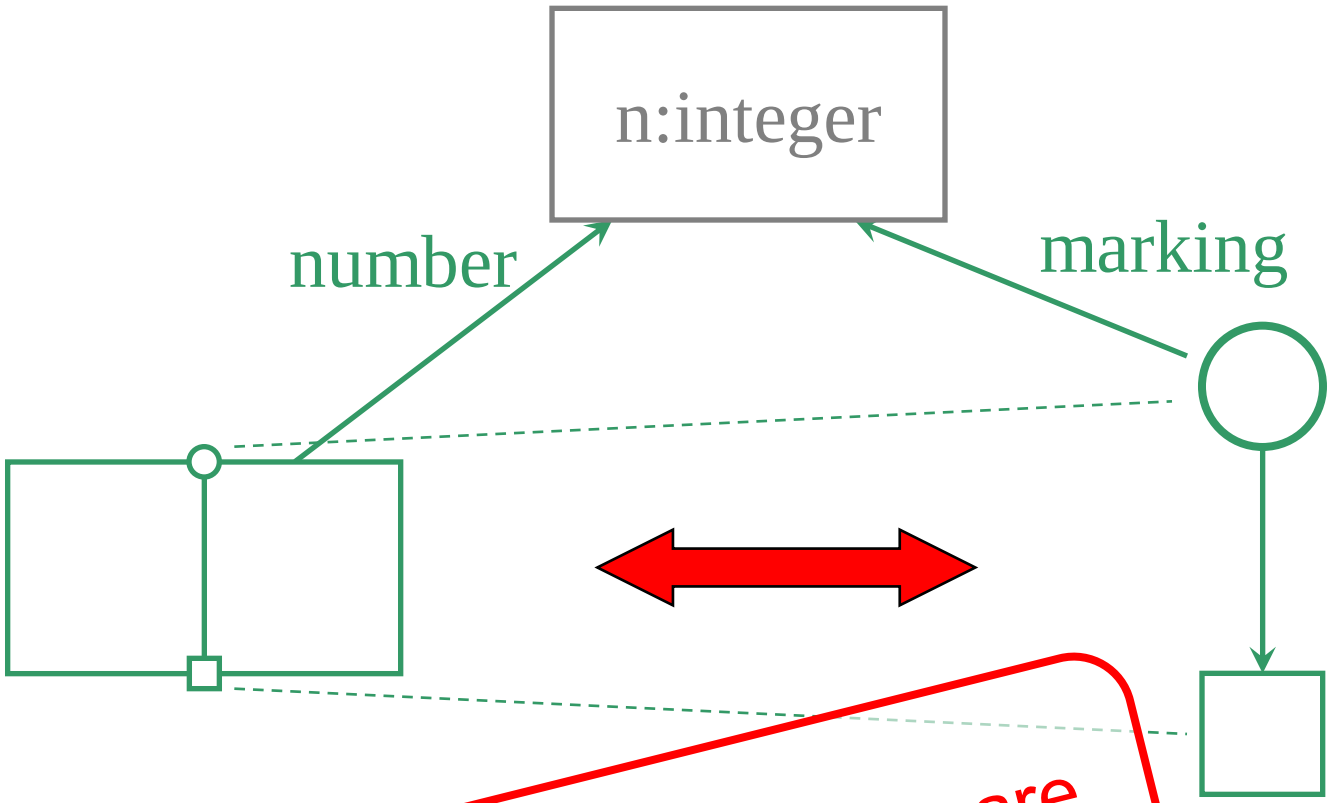
Rules in with attributes

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



Rules in with attributes

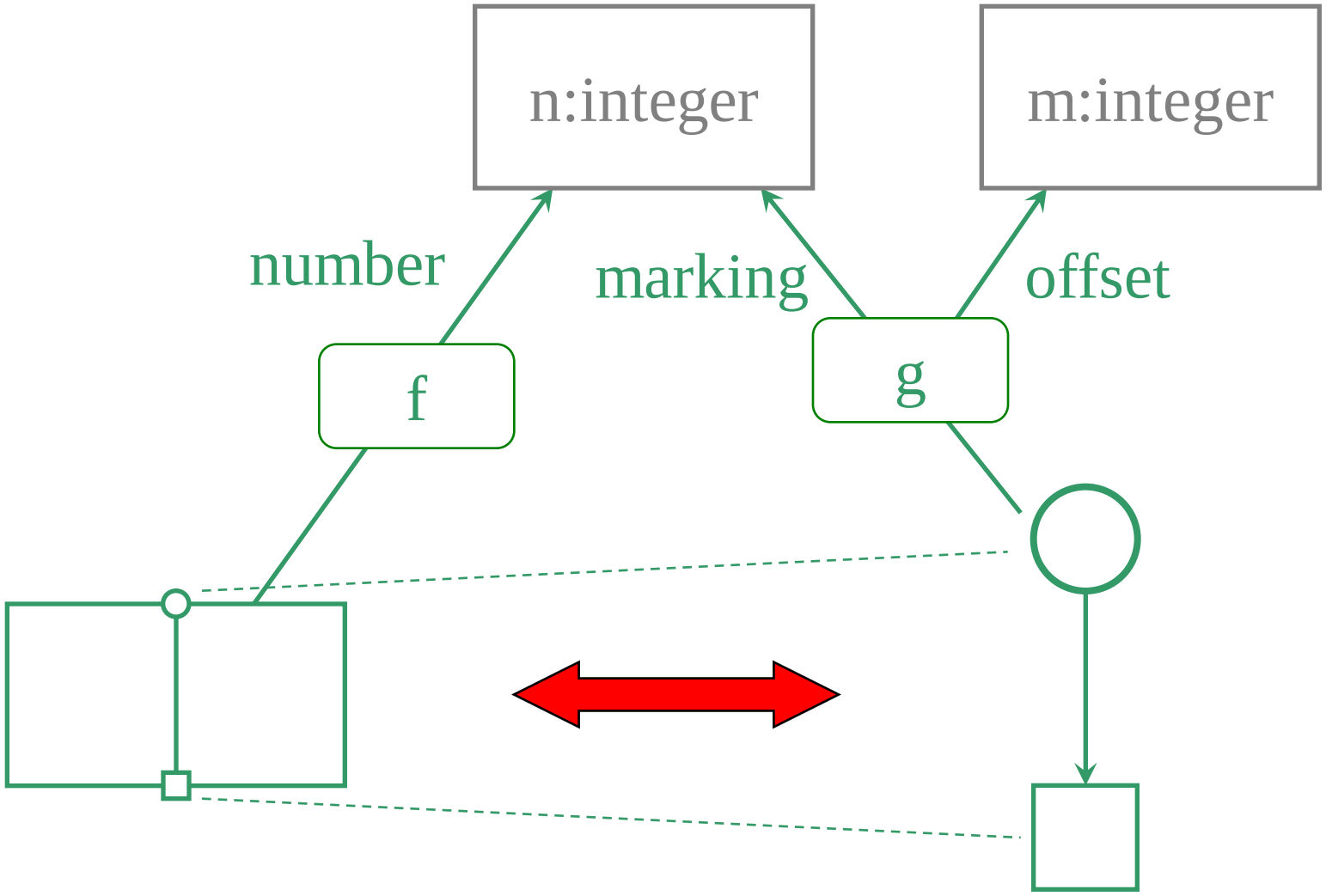
DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



Values of attributes are "grey nodes"!

Attributes: General Concept

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



Clean definitions

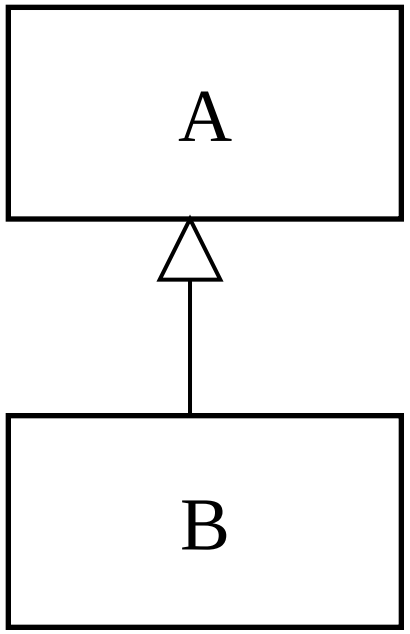
DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- 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

Meta model



Node in a TGG rule

«final»
node:A

Model node

b:B

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

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

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.

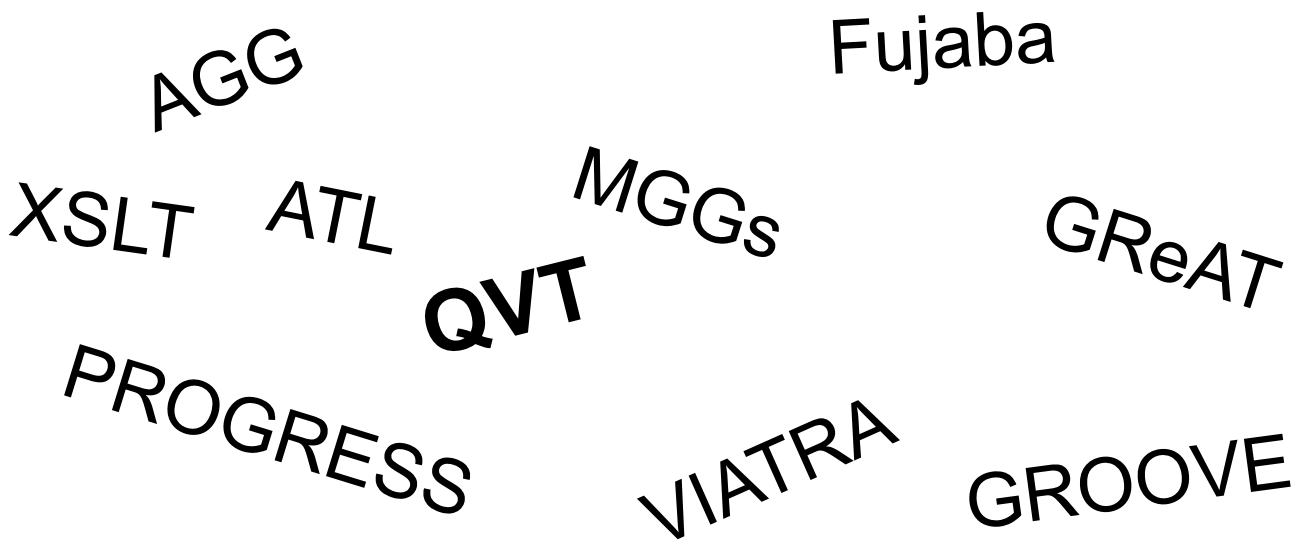
→ We did NOT invent TGGs (that was Andy Schürr more than 20 years ago)

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

3. Overview and Classification

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Besides TGGs, there are many other model transformation approaches and tools



- with different features, strengths and weaknesses and on different technical levels (some of them on the border between M2M and M2T)

Motivation

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

There are many approaches (and we cannot have a look into all of them).

Instead, we try to identify the

- main features,
- main differences and
- characteristics

of transformation technologies

This classification loosely follows ideas of van Gorp, Karsai, Mens, Varro, and others

After this overview, we will look at one other technology (QVT); partly as kind of an "exercise" to apply the classification.

Characteristics

- Program vs **model (graph)**

This is very close to the distinction M2T vs M2M.

- Graph vs **model**
  - on models: which kind of meta-modelling technology:
    - proprietary
    - MOF/XMI
    - MOF/Ecore/XMI/EMF
    - MOF/UML

Though not relevant from the conceptual point of view, this is a very important criterion when deciding for a transformation technology (tool).

Characteristics

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- horizontal vs vertical
  - horizontal: envolved models are on the same level of abstraction
  - vertical: envolved models are on different levels of abstraction

Conceptually, a transformation technology can be used for both. But, technologies tend to be better in one than in the other.

Characteristics

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- endogen vs exogen
  - endogen: there is a single meta-model for all involved models! (often it ”is” a single model → in-place)
  - **exogen**: there is (or can be) a different meta-model for every involved model

Characteristics

- in-place vs ”out-place”
  - in-place: changes the model itself
    (by definition, this is endogen)
  - **out-place**: works on different models

Conceptually, an “in-place transformation” is not even a transformation; it is more a change of a model itself. But, by making a copy first, it becomes a transformation.

Characteristics

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- two models vs multiple models
  - two: there is exactly one source and one target model
  - multiple: any number of models can be involved

TGGs can, in principle, deal with any number of involved models.

Then, they are often called MGGs.

Considered as a transformation, more than two models seem to be a bit awkward.

But, it makes much sense, to keep many models consistent.

Characteristics

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- operational vs declarative
  - operational: describes imperatively (algorithmically) in which way the target model is constructed from the source model
  - declarative: defines the relation between two (classes of) models, but not how a respective transformation is done

If a declarative approach is any good, at least in some cases, the definition can be made operational in at least one direction.

Characteristics

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- uni-directional vs bi-directional
  - uni-directional: the transformation can be executed in one direction only

This is typical for most operational approaches.

  - bi-directional (multi-directional): the transformation can be done in both directions

This is typical for most declarative approaches.

  - synchronisation (→ see incremental)

Characteristics

- en-bloc vs incremental
  - en-bloc: a transformation is always done from scratch; the complete source model is transformed into a new target model every time the transformation is executed

Often, there are additional mechanisms on top of en-bloc transformations, that **merge** elements from an earlier transformation (and the manual changes) with the new transformation result.

  - incremental: changes on a model can be incrementally transformed (typically, the relation between the model elements is stored permanently)

Being incremental and bi-/multi-directional is the main prerequisite for model synchronisation.

Characteristics

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- non-standard vs standard

Though, conceptually and technically this does not play a role. It does for industrial applications!

Exercise

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

What are the characteristics of

- TGGs / MGGs
- JET
- QVT

4. Other approaches

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

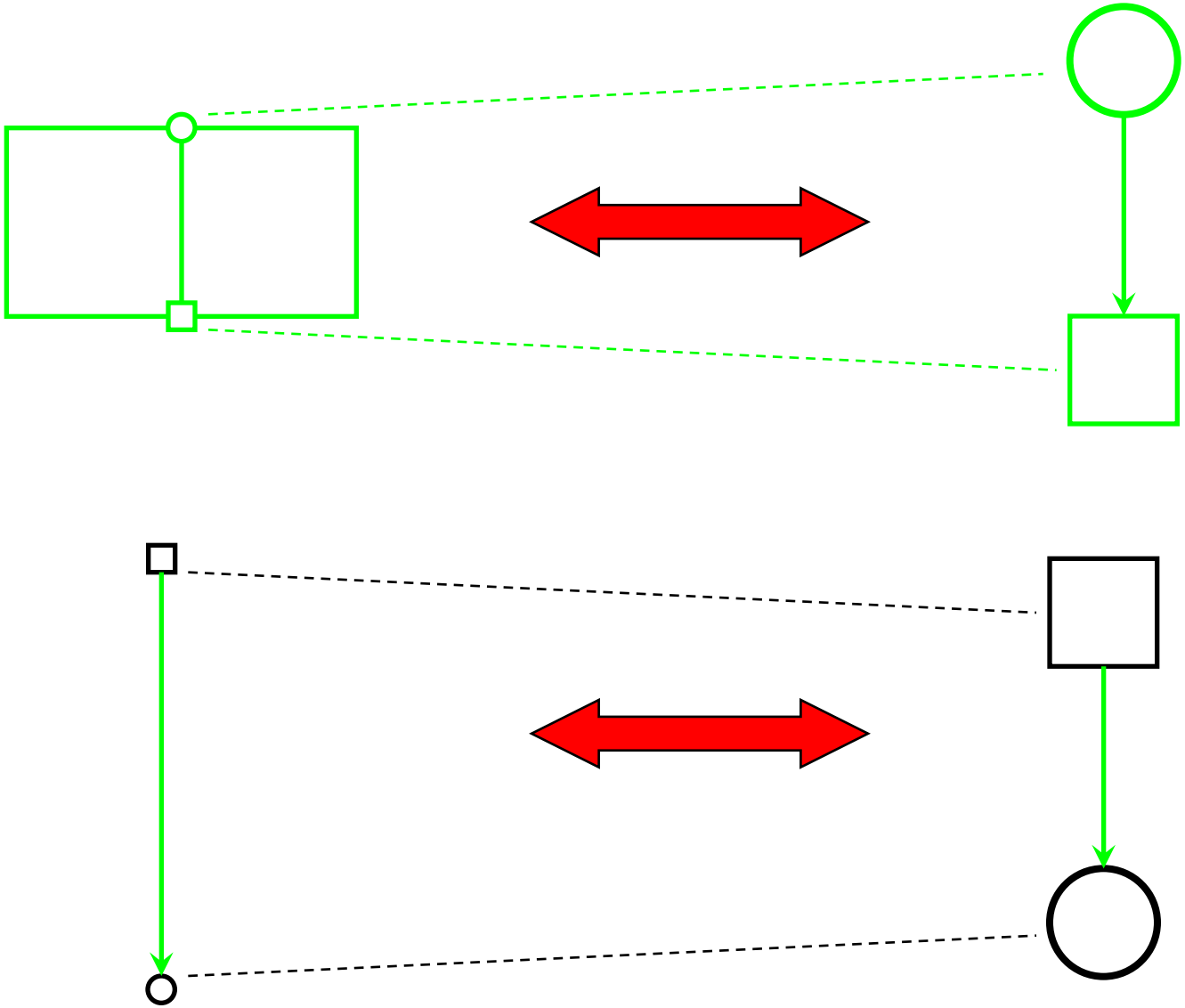
Query/View/Transformation (QVT)

- is the OMG Standard that comes along with MOF
- its purpose is to support the transformations necessary in the MDA
- is based on several other OMG standards: MOF, OCL
- consists of two major parts:
  - QVT Operational Mappings (operational)
  - QVT Relations (declarative)
  - QVT Core (declarative)

Here, we give a rough overview on QVT Relations and QVT Operational by example only.

Reminder: TGG-Rules

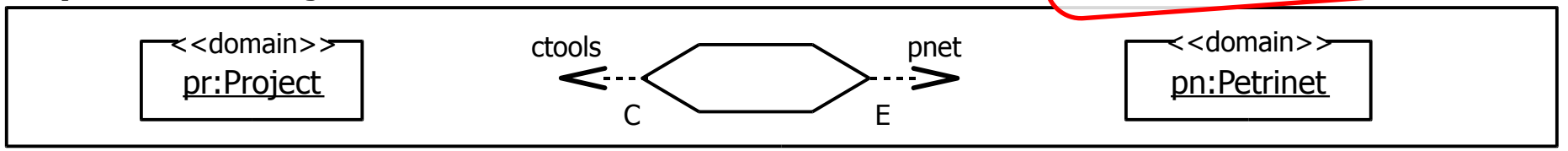
DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



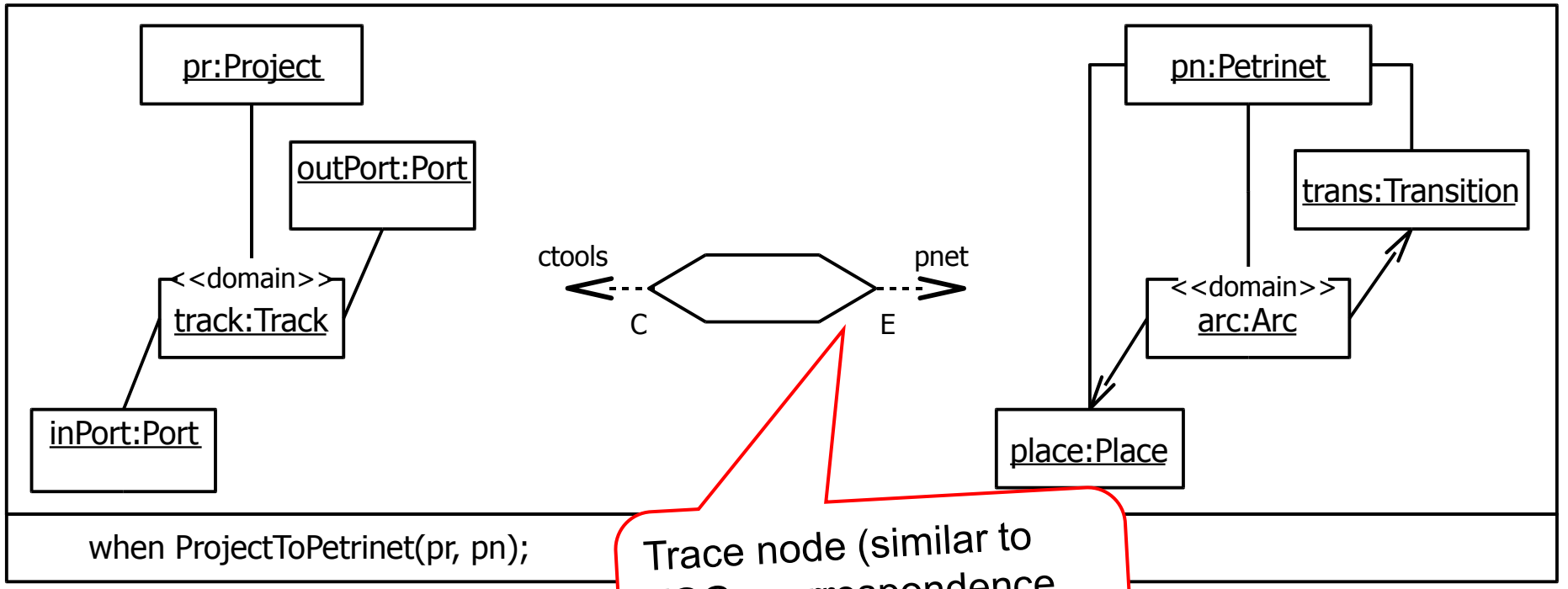
QVT Relations: Two relations

QVT Relations in graphical notation!

top relation ProjectToPetrinet



top relation TrackToPlaceArcTransition



Trace node (similar to TGGs correspondence nodes); exactly one for each relation

QVT Relation: textual

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

```
top relation ProjectToPetrinet {
  checkonly domain ctools pr : ctools::Project{
     name = n
  };

  enforce domain pnet pn : pnet::Petrinet {
     name = n
  };
}
```

QVT Relation: textual

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

```
top relation TrackToPlaceArcTransition {
   checkonly domain ctools track : ctools::Track{
      componentToProject = pr : ctools::Project{},
      componentToPort = portIn : ctools::Port{ type = 'In' },
      componentToPort = portOut : ctools::Port{ type = 'Out' },
   };

   enforce domain pnet arc : pnet::Arc {
      arcToPetrinet = pn : pnet::Petrinet{},
      arcToPlace = place : pnet::Place{ placeToPetrinet = pn },
      arcToTransition = trans : pnet::Transition{
         transitionToPetrinet = pn
      }
   };

   when { ProjectToPetrinet(pr, pn); }
}
```

QVT Relational

Note that there are some subtle but important differences, which we cannot discuss here!

Observations

- variables in QVT correspond to nodes in TGGs
- assignments to variables correspond to arcs in TGGs

- **`when`** is similar to the black nodes of TGGs; but there are some subtle differences
- the **`domain`** is similar to TGG domains; but in QVT, there is one distinguished node indicating the domain
- **`checkonly`** and **`enforce`** indicate a direction (though, in principle, the QVT-rules are independent of a transformation direction).

QVT Operational Mappings

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Imperative definition of a transformation from one model to another
- Not bi-directional
- More efficient
- Can be combined with QVT Relations

Again, just a rough overview by the help of an example.

The example is the “Standard example” of QVT, which transforms UML packages to a database schema!

Model definition

This is QVT's concrete syntax for meta-models; QVT can also refer to external meta-models.

```
metamodel SimpleUml {
   abstract class UMLModelElement {
      kind : String;
      name : String; }

   class Package extends UMLModelElement {
      composes elements : PackageElement [*] ordered
         opposites namespace [1]; }

   abstract class PackageElement extends UMLModelElement
   {  }

   class Classifier extends PackageElement {
   }

   class Attribute extends UMLModelElement {
      references type : Classifier [1]; }
```

Model definition

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

```
class Class extends Classifier {
   composes attribute : Attribute [*]
      ordered opposites owner [1];
   references general : Classifier [*] ordered;
}

class Association extends PackageElement {
   source : Class [1] opposites reverse [*];
   destination : Class [1] opposites forward [*];
}

class PrimitiveDataType extends Classifier {
}
}
```

Model definition

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

```
metamodel SimpleRdbms {
   abstract class RModelElement {
      kind : String;
      name : String; }

   class Schema extends RModelElement {
      composes tables : Table [*] ordered
         opposites schema [1]; }

   class Table extends RModelElement {
      composes column : Column [*] ordered
         opposites owner[1];
      composes _key : Key [*] ordered opposites owner[1];
      composes foreignKey : ForeignKey [*] ordered
         opposites owner[1]; }
```

Model definition

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

```
class Column extends RModelElement {
   type : String;
}

class Key extends RModelElement {
   references column : Column [*] ordered
      opposites _key [*];
}

class ForeignKey extends RModelElement {
   references refersTo : Key [1];
   references column : Column [*] ordered
      opposites foreignKey [*];
}
}
```

QVT Operational Mappings

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

```
transformation Uml2Rdb(in srcModel:UML,out dest:RDBMS);

  intermediate class LeafAttribute {
     name:String;
     kind:String;
     attr:UML::Attribute; };

  intermediate property UML::Class::leafAttributes :
     Sequence(LeafAttribute);

  query UML::Association::isPersistent() : Boolean {
     result = (self.source.kind='persistent' and
        self.destination.kind='persistent'); }

  main() {
     srcModel.objects()[Class]->map class2table();
     srcModel.objects()[Association]->map asso2table();}
```

Here comes the actual transformation part

QVT Operational Mappings

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

```
mapping Class::class2table () : Table
   when {self.kind='persistent'; }
{ init {
     self.leafAttributes := self.attribute ->
       map attr2LeafAttrs("","");
   }
   population {
     name := 't_' + self.name;
     column := self.leafAttributes->
       map leafAttr2OrdinaryColumn("");
     key_ := object Key {
               name := 'k_'+ self.name;
               column := result.column[kind='primary'];
             };
   }
} ...
```

OCL-like way to select a particular element from a collection.

Some parts and some details left out.

QVT Operational Mappings

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Observations

- uni-directional
- imperative
- driven by source model
- programming possible
- can be combined with QVT relations (and QVT core)

5. Summary

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Many different concepts and technologies for model transformation
- different characteristics
- to date:
  different technologies for different purposes necessary

- Missing: Concepts for easy combination of transformation / synchronisation mechanisms

Discussion: M2M vs M2T

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Is there a difference between M2T and M2M?

1st answer:

No: Most languages have meta-models and APIs for accessing and manipulating them. These are called Abstract Syntax Trees (AST).

Then, M2M technologies can be used for M2T transformations. Even more, using this API can guarantee syntactical corrctness of the result.

Discussion: M2M vs M2T

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Is there a difference between M2T and M2M?

2nd answers:

Yes: Sometime, it is easier to produce text directly; using the API is overkill. M2T is just easier to handle.

Sometimes, there is no strictly fixed syntax, and no AST. Then, we need M2T anyway.

As almost always, the answer is “it depends”.

Recommended reading

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Eclipse has a nice way of accessing and manipulating Java-programs via an API for ASTs.

Have a look at the Eclipse Article / Tutorial on

Abstract Syntax Trees:

http://www.eclipse.org/articles/article.php?file=Article-JavaCodeManipulation_AST/index.html