CORRECTNESS, COMPLETENESS, AND TERMINATION OF PATTERN-BASED MODEL TO MODEL TRANSFORMATION

Fernando Orejas, UPC (Barcelona)

Esther Guerra, U. Carlos III (Madrid)

Juan de Lara, U. Autónoma (Madrid)

Hartmut Ehrig, TU Berlin

orejas@lsi.upc.edu

What is a model transformation

A model transformation is the translation of a description (specification) of a software system (or artifact) with the aim of:

- Refining that description towards implementation
- Analysing the specification
- Abstracting some details
- Improving the performance of the system
- ...

Model Transformations

There are many kinds of model transformations:

- Endogenous or exogenous
- Monodirectional or Bidirectional or Synchronized
- One-to-one or many-to-many
- To obtain "semantically equivalent" models
- From more abstract to more concrete models
- From more concrete to more abstract models

Description of model transformations

Model transformations may be defined:

- Operationally
- Declaratively

The OMG has defined the language QVT to describe model transformations including an operational and a declarative sublanguage

Aim of this work

To develop methods to implement declarative specifications of model transformations.

M2M pattern specification

- A visual declarative framework to describe bidirectional model to model transformations.
- Inspired in the relational fragment of QVT
- Two kinds of patterns:
  - Positive and Negative patterns

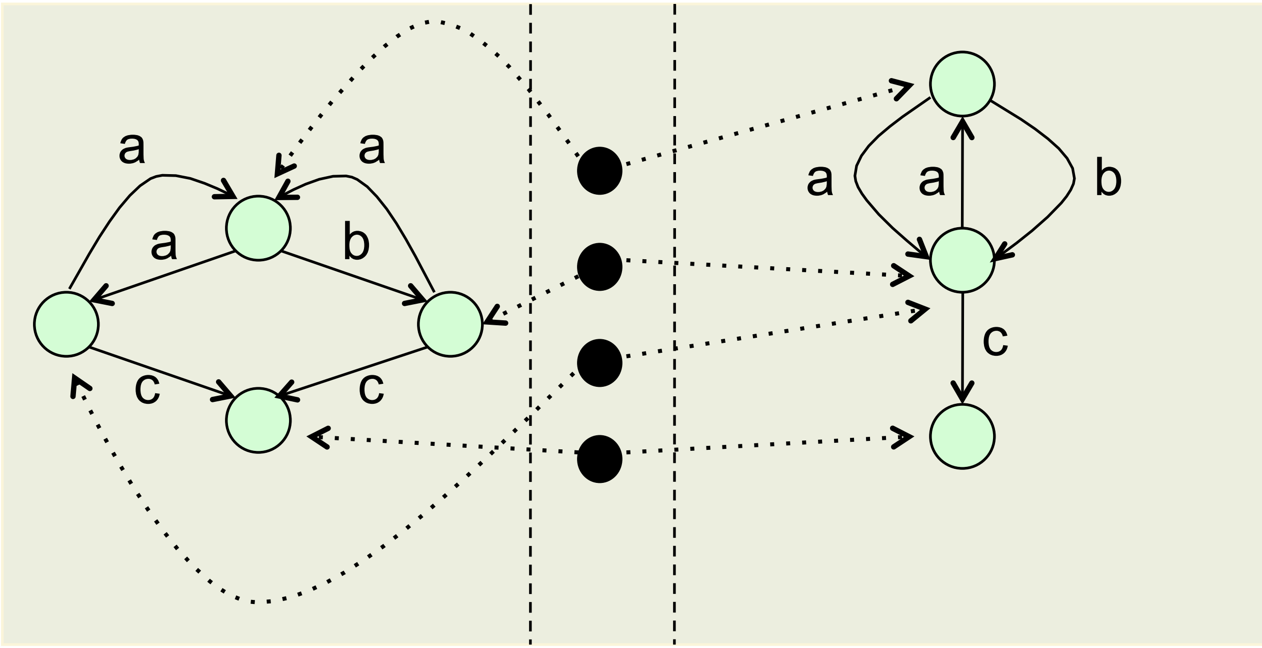
What is a Model Transformation

We may formalize model transformations by a span of triple graphs, called a *triple graph*:

$$M_S \xleftarrow{h_S} M_C \xrightarrow{h_T} M_T$$

Triple graphs

- A triple graph [Schurr 1994] models the relation between two graphs:

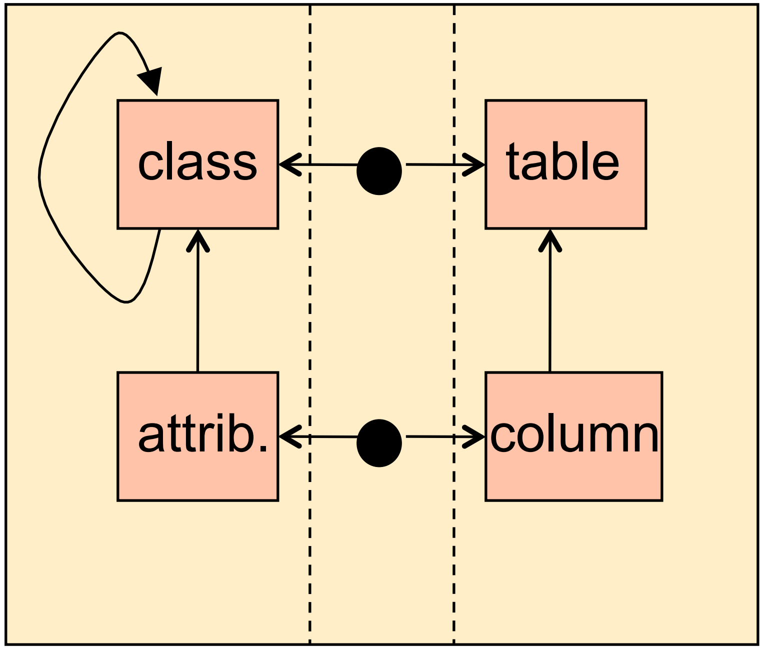


Specifying Model Transformations

Specifing a model transformation means describing:

- How the given source and target types are related.
- What are the possible transformations of each model or instance.

Example – the triple type graph



Triple Patterns

Triple patterns are constraints on triple models.

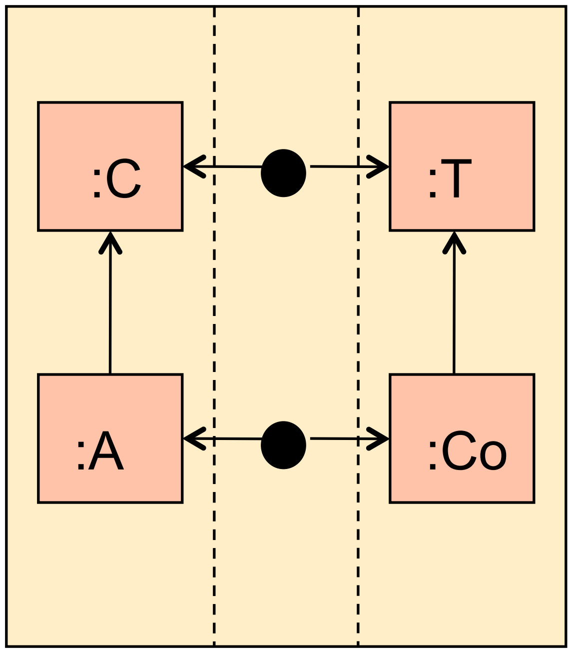
- Positive patterns describe possible relationships between source and target elements (under a given negative premise)

$$N(Q \rightarrow C_j)_{j \in J} \Rightarrow Q$$

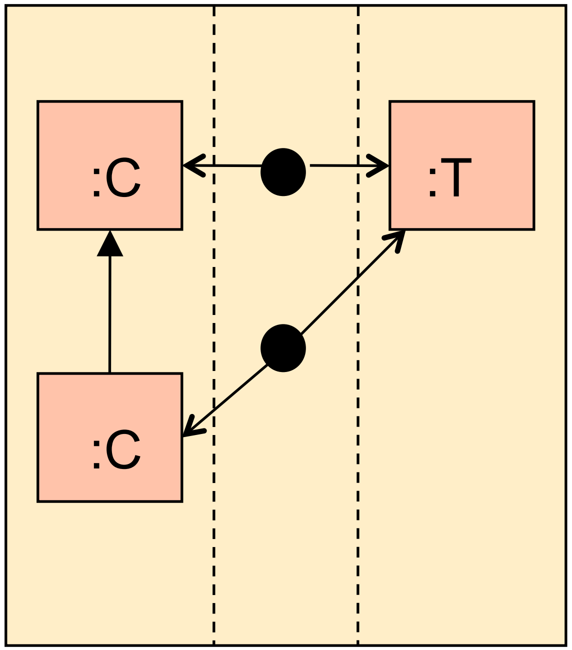
- Negative patterns describe forbidden relationships.

$$N(Q)$$

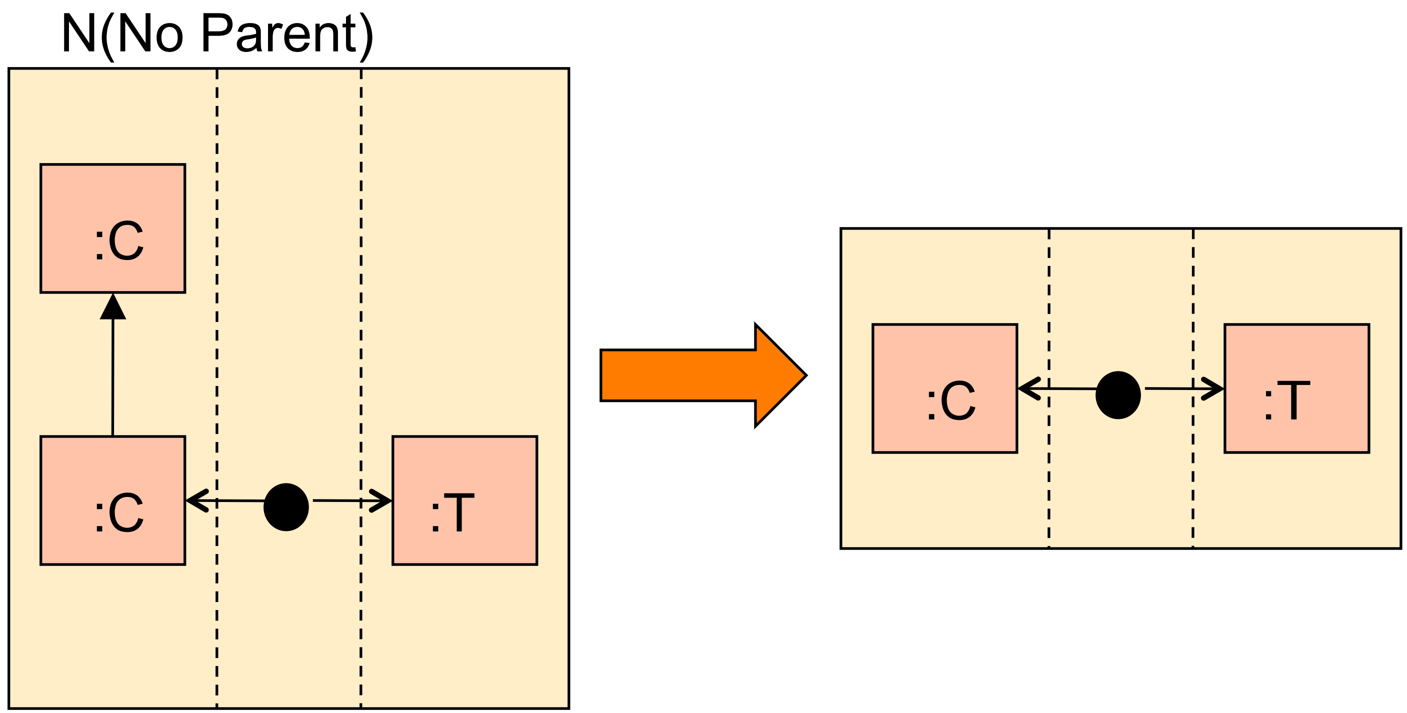
Example



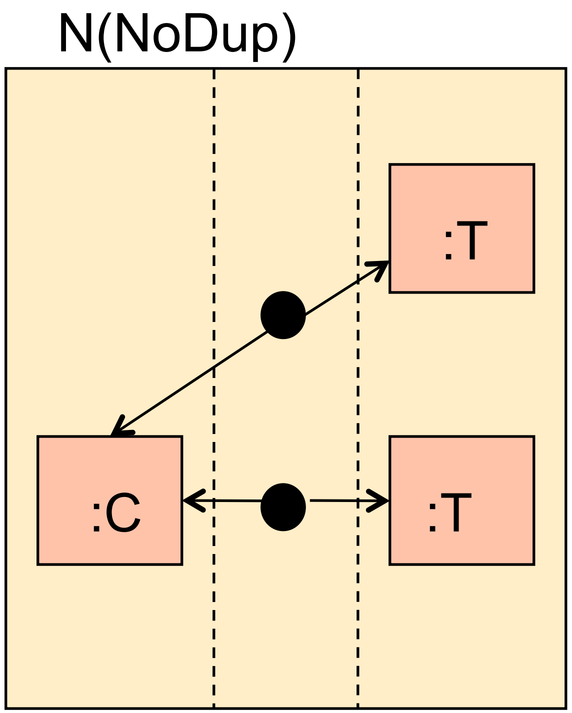
Example



Example



Example



Satisfaction

A positive pattern $N(Q \rightarrow C_j)_{j \in J} \Rightarrow Q$ is forward satisfied by a triple graph G if whenever Q_S can be matched to G_S via an injective h, and h satisfies the preconditions then h can be extended to an injective morphism $h': Q \rightarrow G$.

$$\begin{tikzcd}
C_j|_S \arrow[dr, "/\!/" marking] & Q|_S \arrow[l] \arrow[r] \arrow[d, "h"] & Q \arrow[d, "h'"] \\
& G|_S \arrow[r] & G
\end{tikzcd}$$

A negative pattern $N(Q)$ is forward satisfied by a triple graph G if there is no injective morphism $h: Q \rightarrow G$.

Specification of Model Transformations

A model transformation specification SP consists of a triple type graph set of positive patterns and negative patterns over this type graph.

The transformation specified by SP is defined by the class of triple typed graphs satisfying SP *that can be considered to be generated by the patterns in SP*.

Non-deleting GraphTransformation

- A (non-deleting) transformation rule is a graph monomorphism L → R.

- The application of a rule to a graph G is given by a pushout:

$$
\begin{tikzcd}
L \arrow[r] \arrow[d] \arrow[dr, phantom, "\text{po}"] & R \arrow[d] \\
G \arrow[r] & G'
\end{tikzcd}
$$

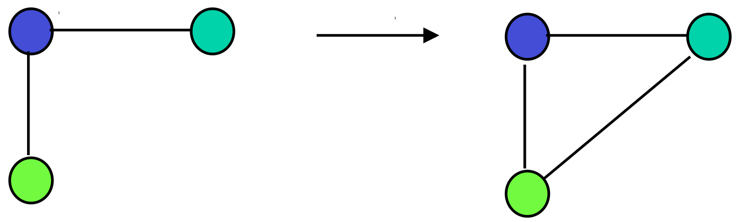
Negative application conditions

- A (left) Negative Application Condition (NAC) for a transformation rule is an embedding $L \rightarrow N$.

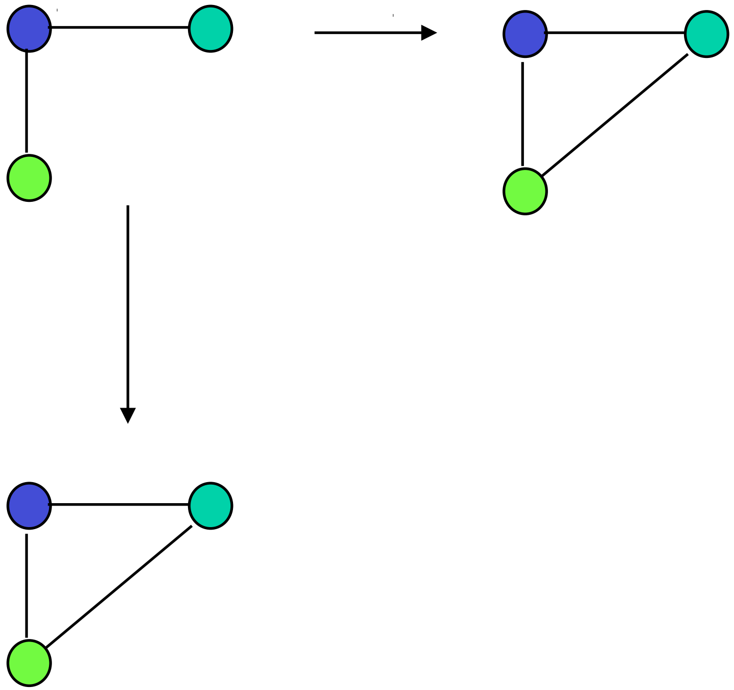
- The application of a rule with a NAC to G is given by a pushout, if there is no h making the triangle diagram commute

$$
\begin{tikzcd}
N & L \arrow[l] \arrow[r] \arrow[d] \arrow[dr, phantom, "po"] & R \arrow[d] \\
 & G \arrow[from=1-1, "h"', "/" marking] \arrow[r] & G'
\end{tikzcd}
$$

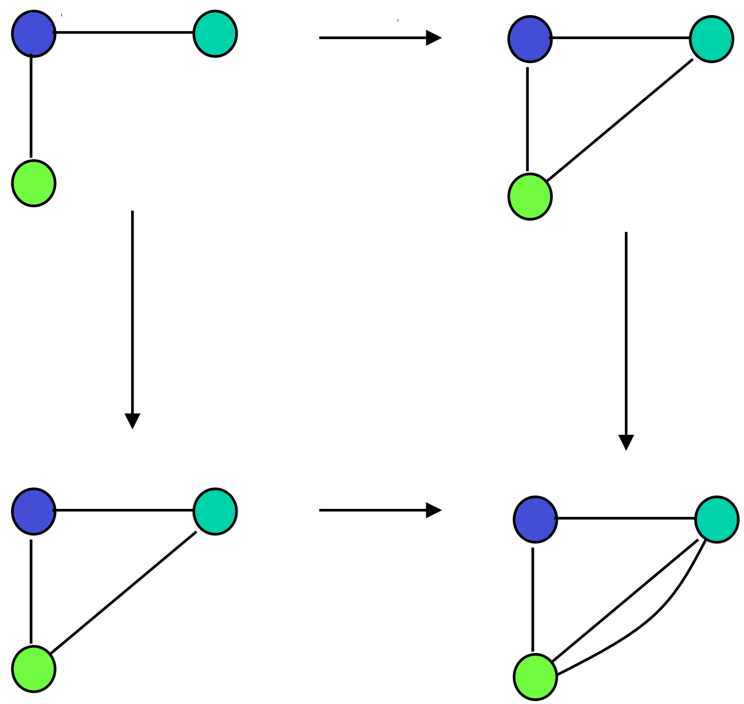
Example



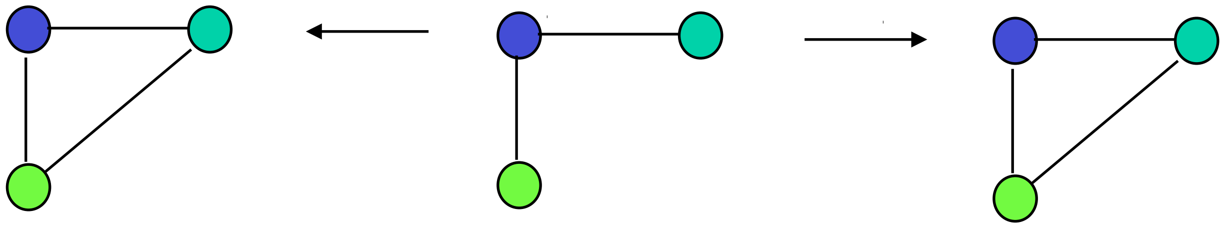
Example



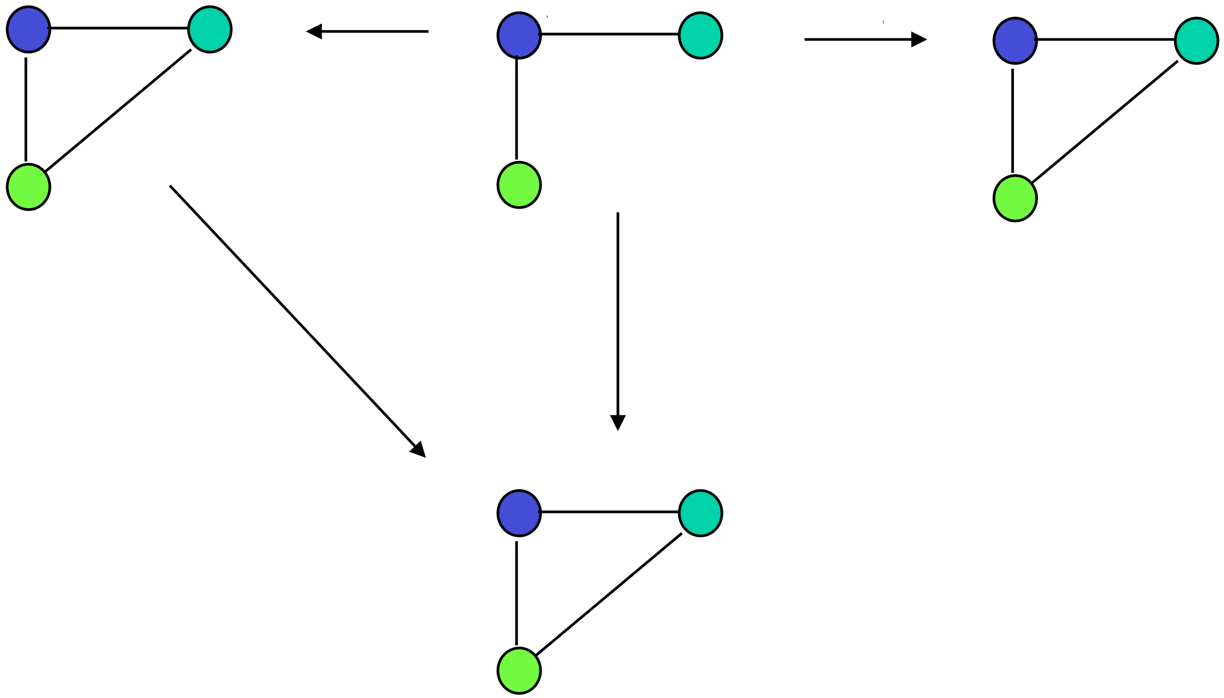
Example



Example



Example



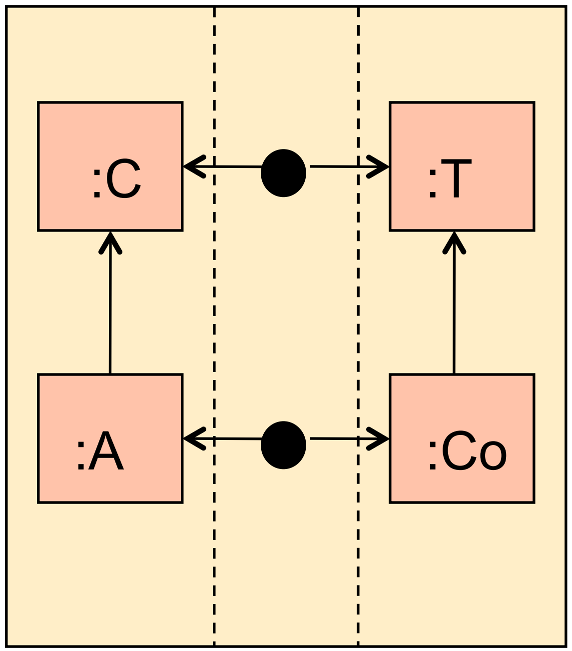
Compiling positive patterns into rules

Given $P = \langle N(Q \rightarrow C_j)_{j \in J} \Rightarrow Q \rangle$, the set of transformation rules associated to P, TR(P), consists of all the rules $r = \langle NAC(r), L \rightarrow Q \rangle$, such that:

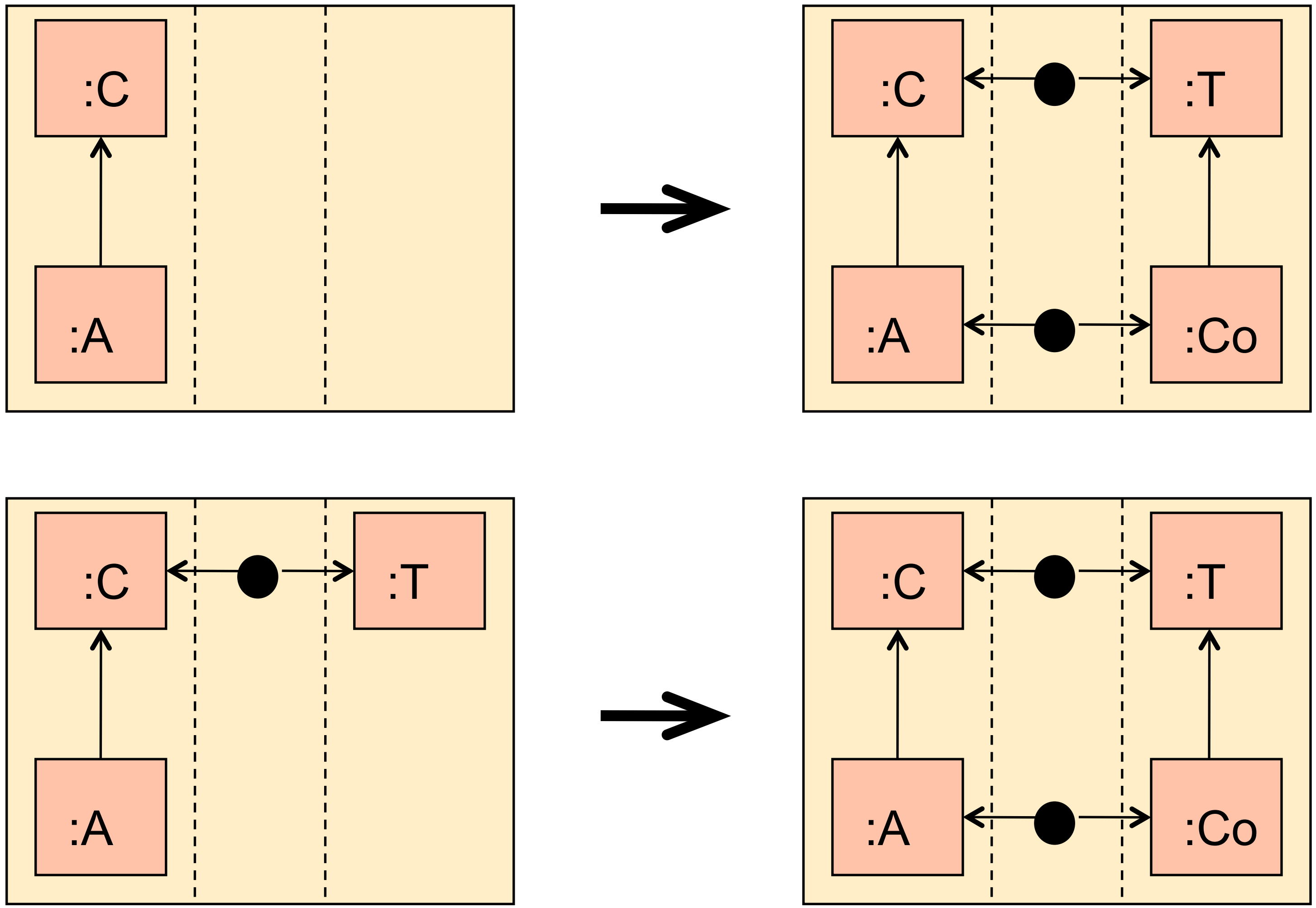
- $Q|_S \subseteq L \subset Q$
- NAC(r) consists of all the NACs:

$$\begin{tikzcd} Q|_S \arrow[r] \arrow[d] \arrow[dr, phantom, "\text{po}"] & C_j|_S \arrow[d] \\ L \arrow[r] & C'_j \end{tikzcd}$$

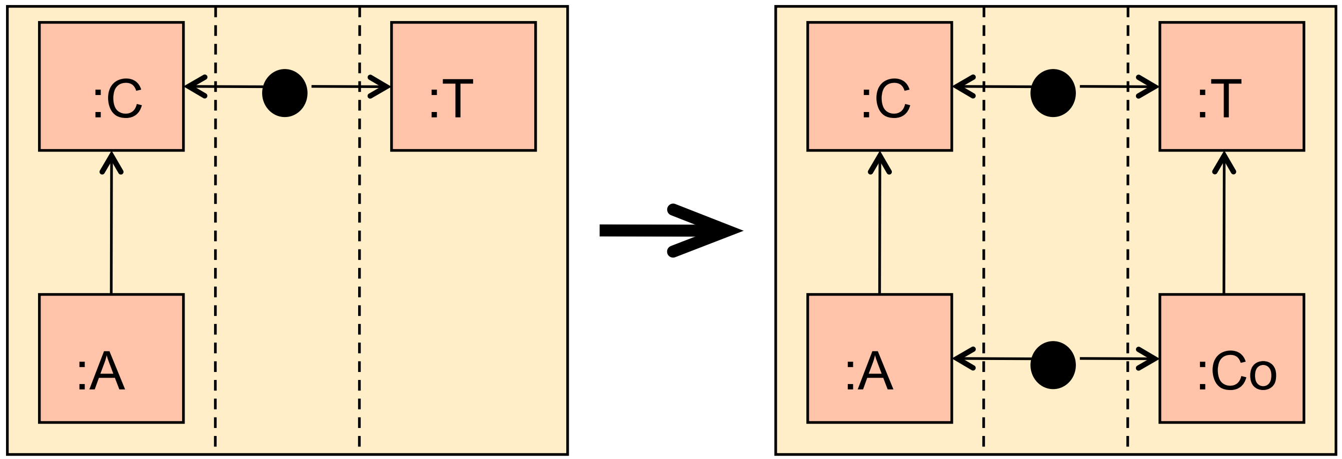
Examples



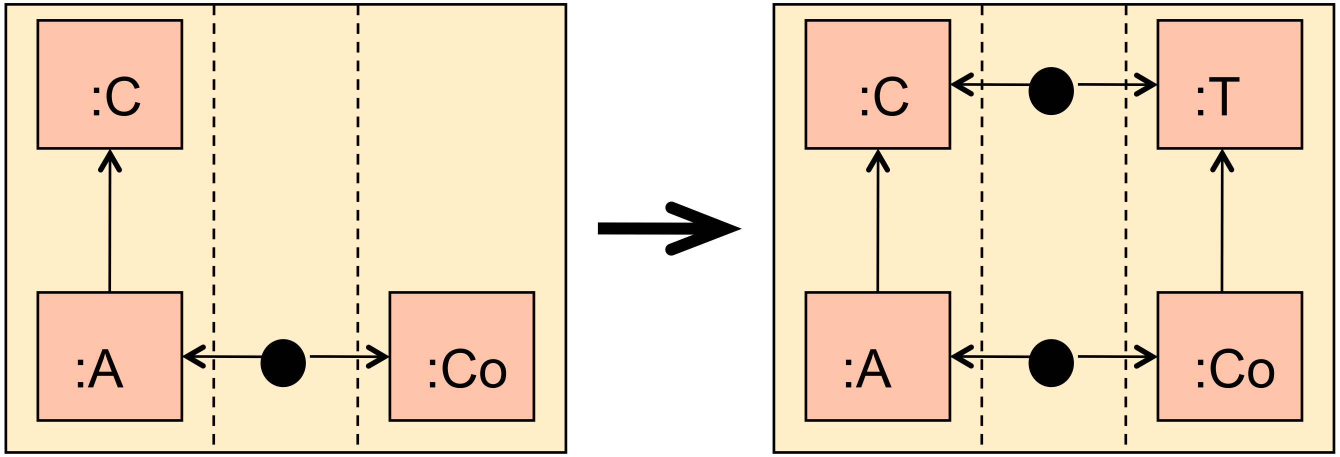
Examples



Examples



Examples



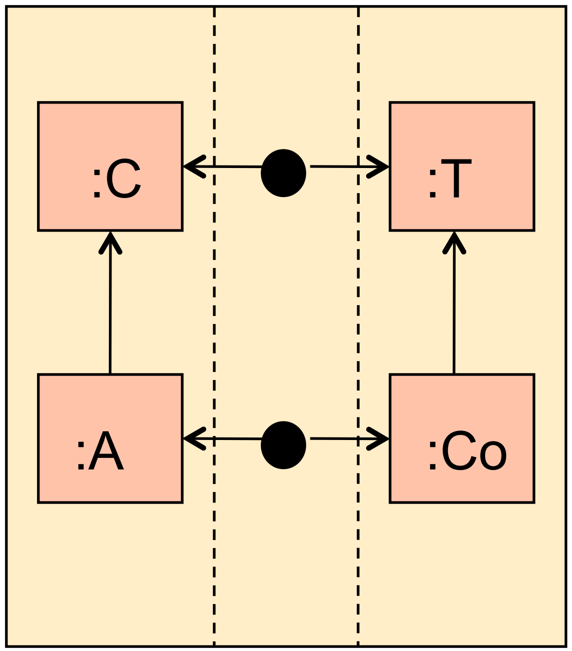
Compiling positive patterns into terminating rules

Given $P = \langle N(Q \rightarrow C_j)_{j \in J} \Rightarrow Q \rangle$, the set of terminating transformation rules associated to P, TTR(P), consists of all the rules $r = \langle NAC(r) \cup TNAC(r), L \rightarrow Q \rangle$, such that:

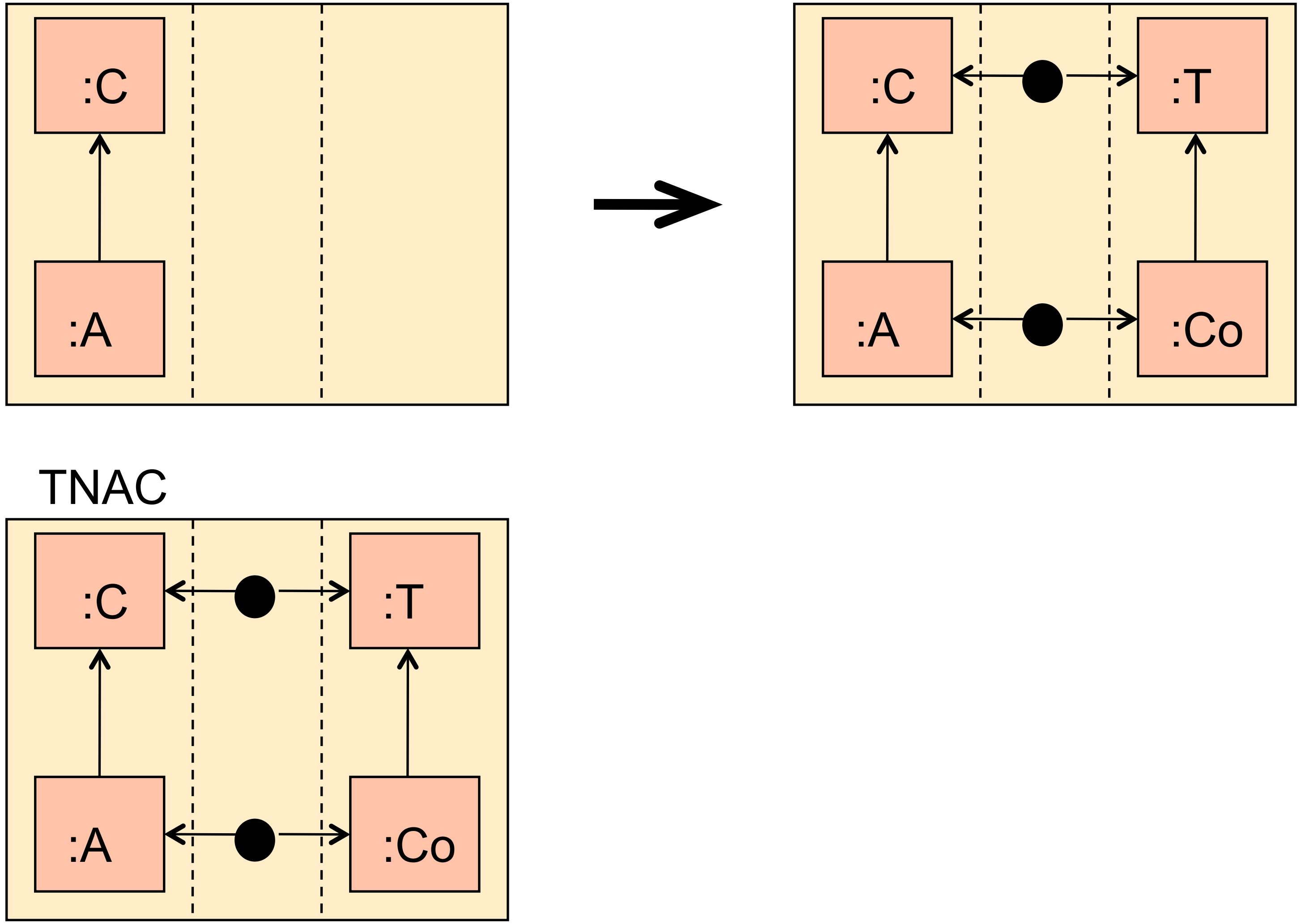
- $\langle NAC(r), L \rightarrow Q \rangle \in TR(P)$
- TNAC(r) consists of all the NACs

$$
\begin{tikzcd}
Q|_S \arrow[r] \arrow[d] \arrow[dr, phantom, "\text{j.s.}"] & Q \arrow[d] \\
L \arrow[r] & T
\end{tikzcd}
$$

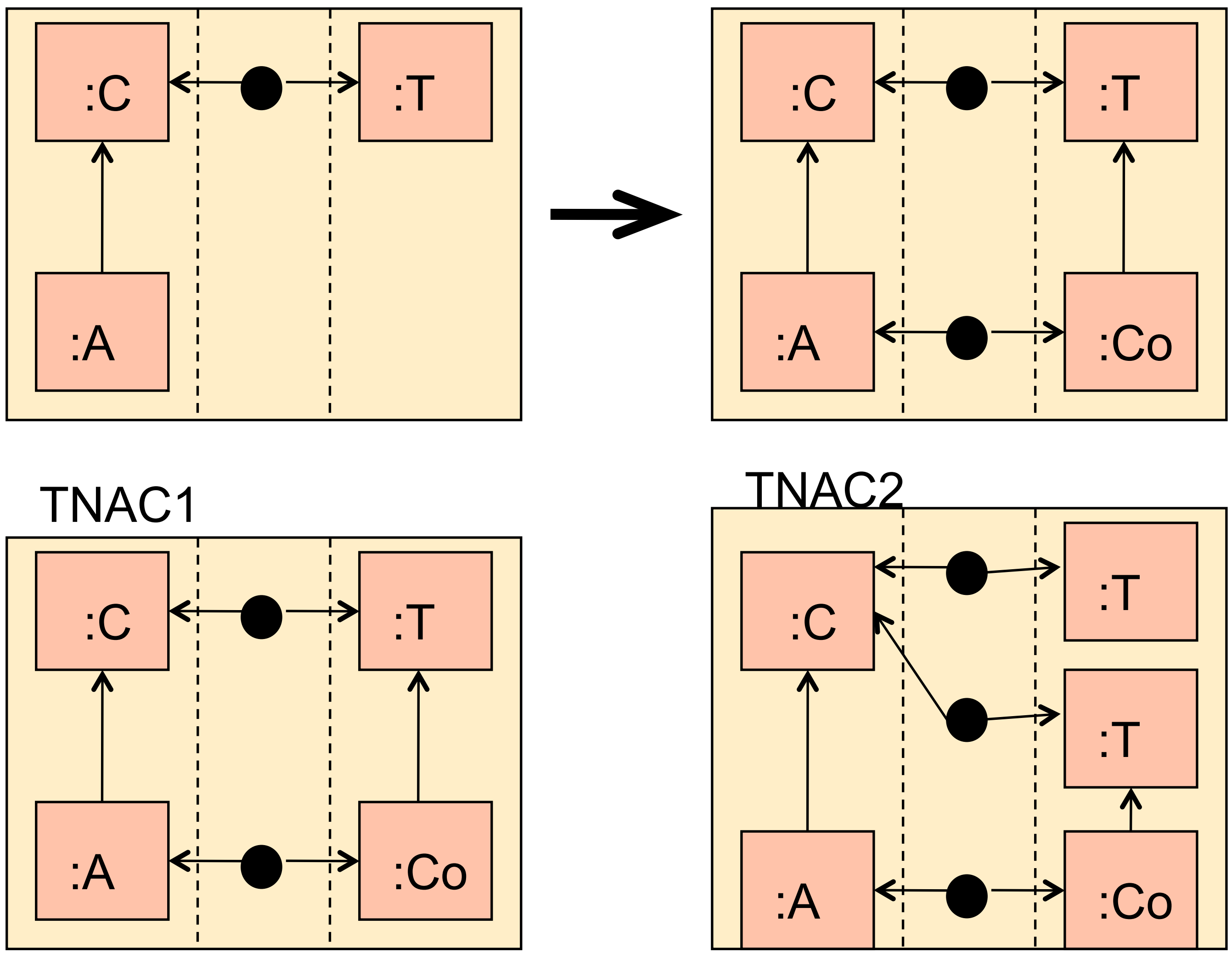
Examples



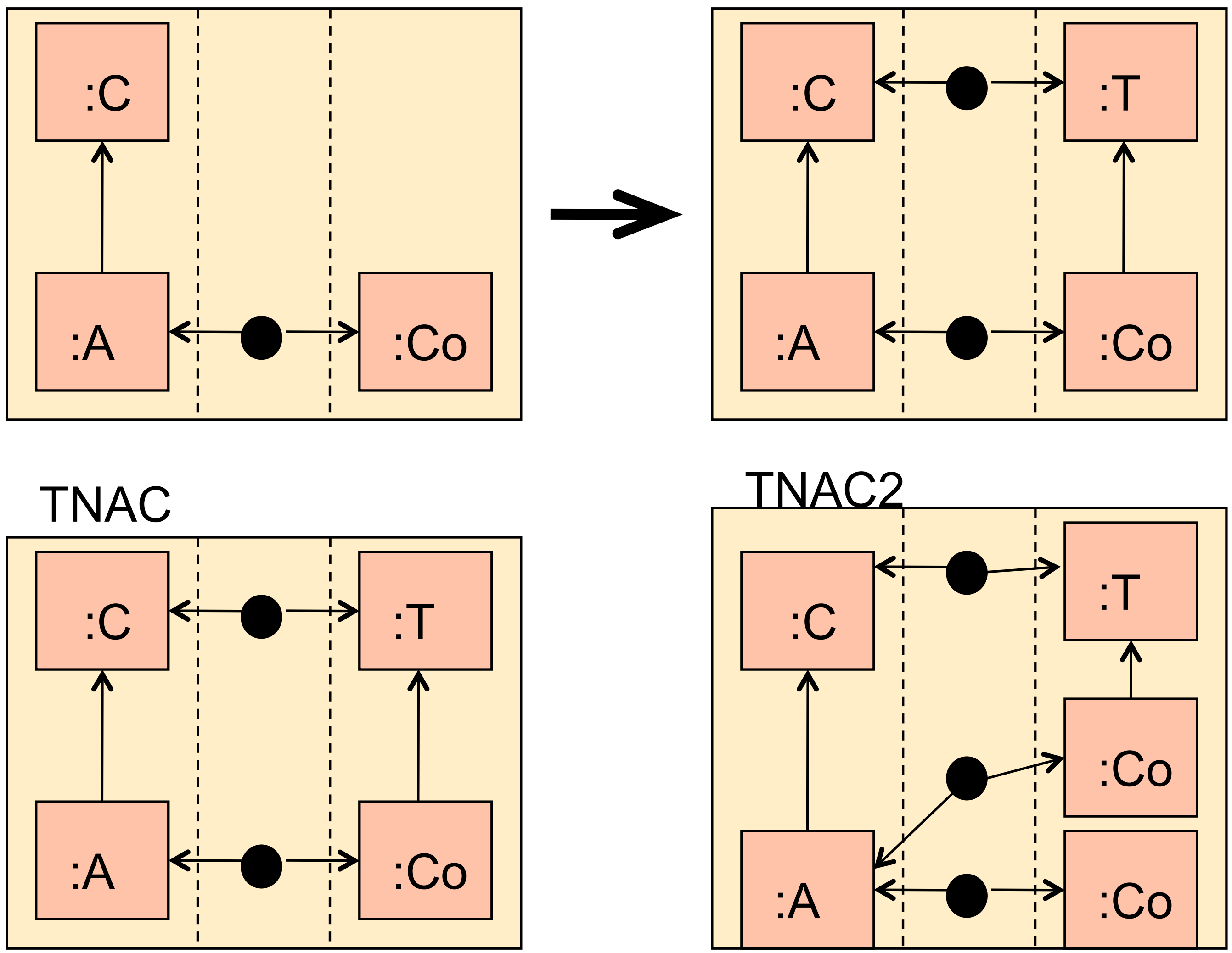
Examples



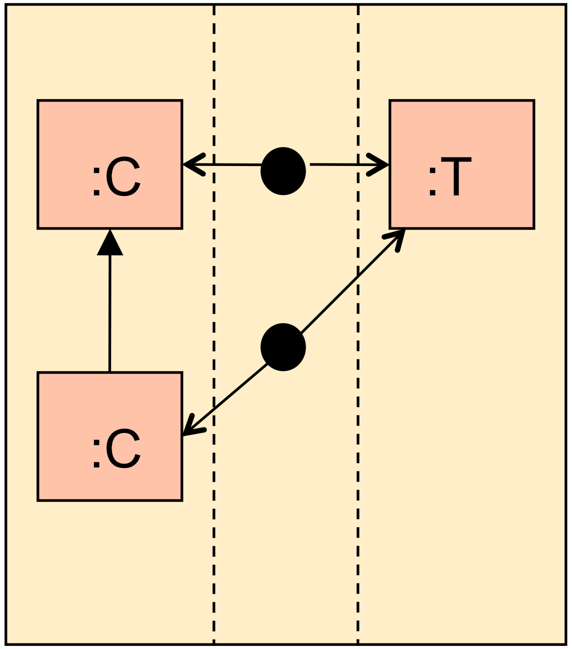
Examples



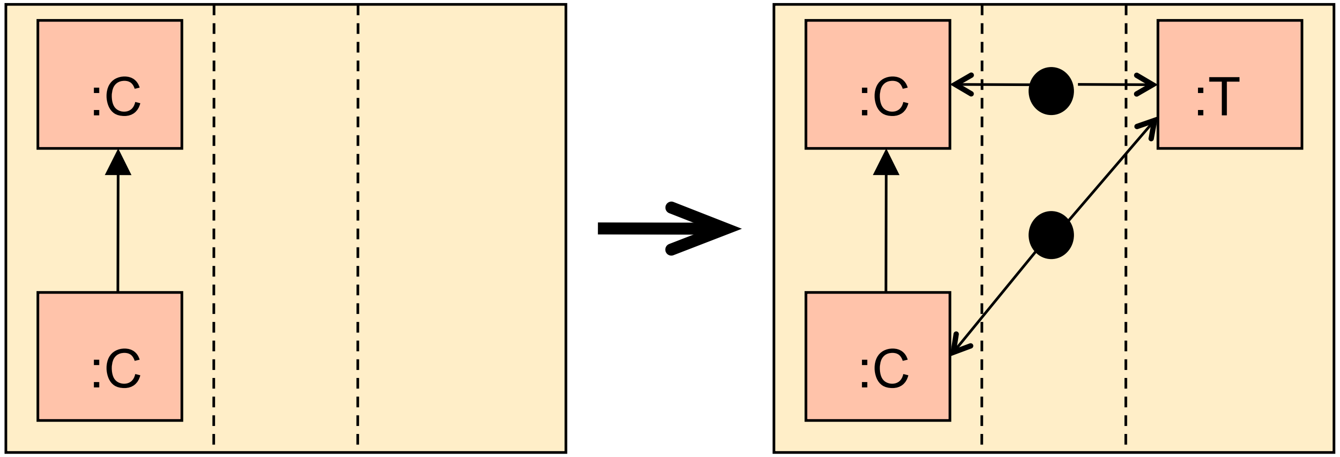
Examples



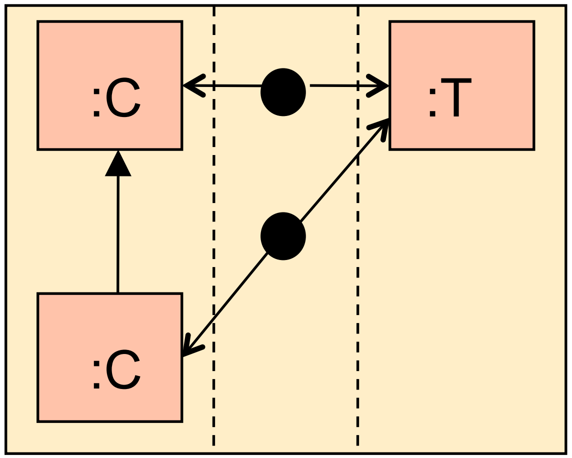
Examples



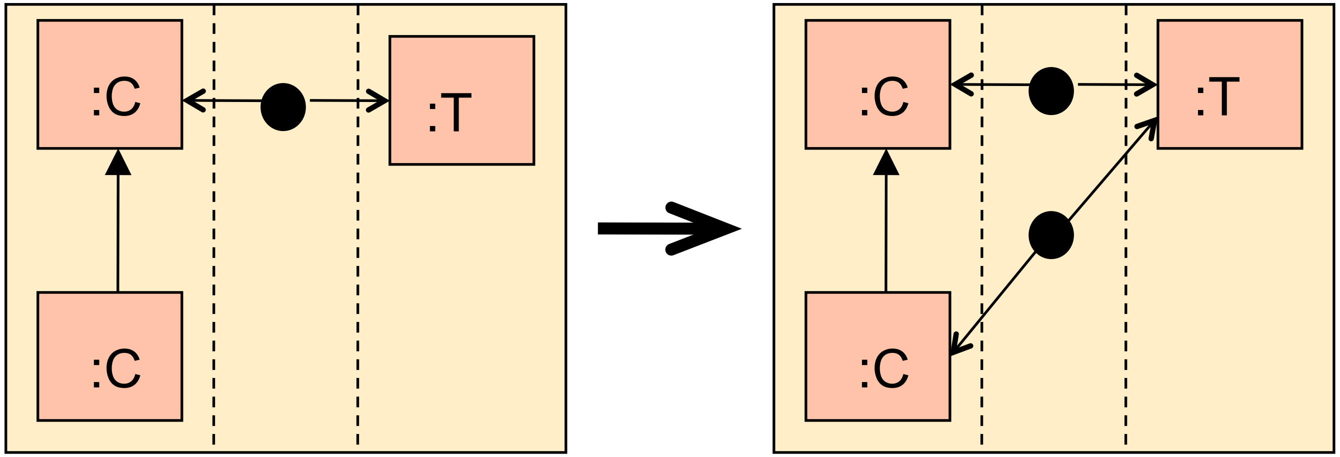
Examples



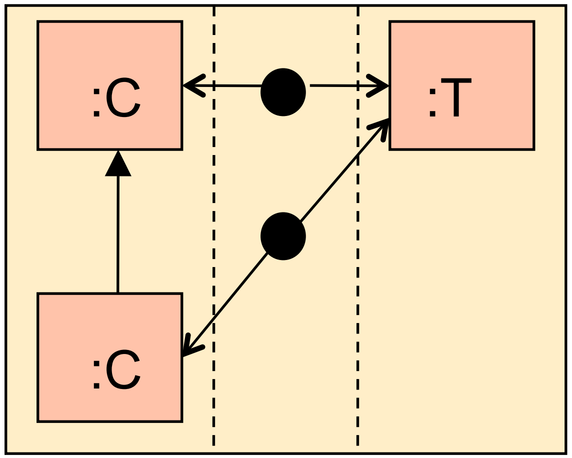
TNAC



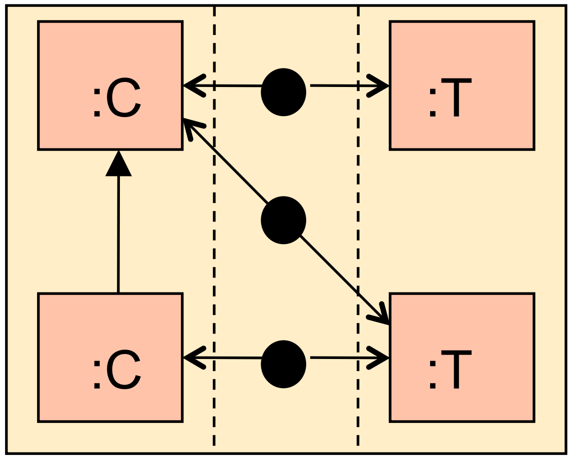
Examples



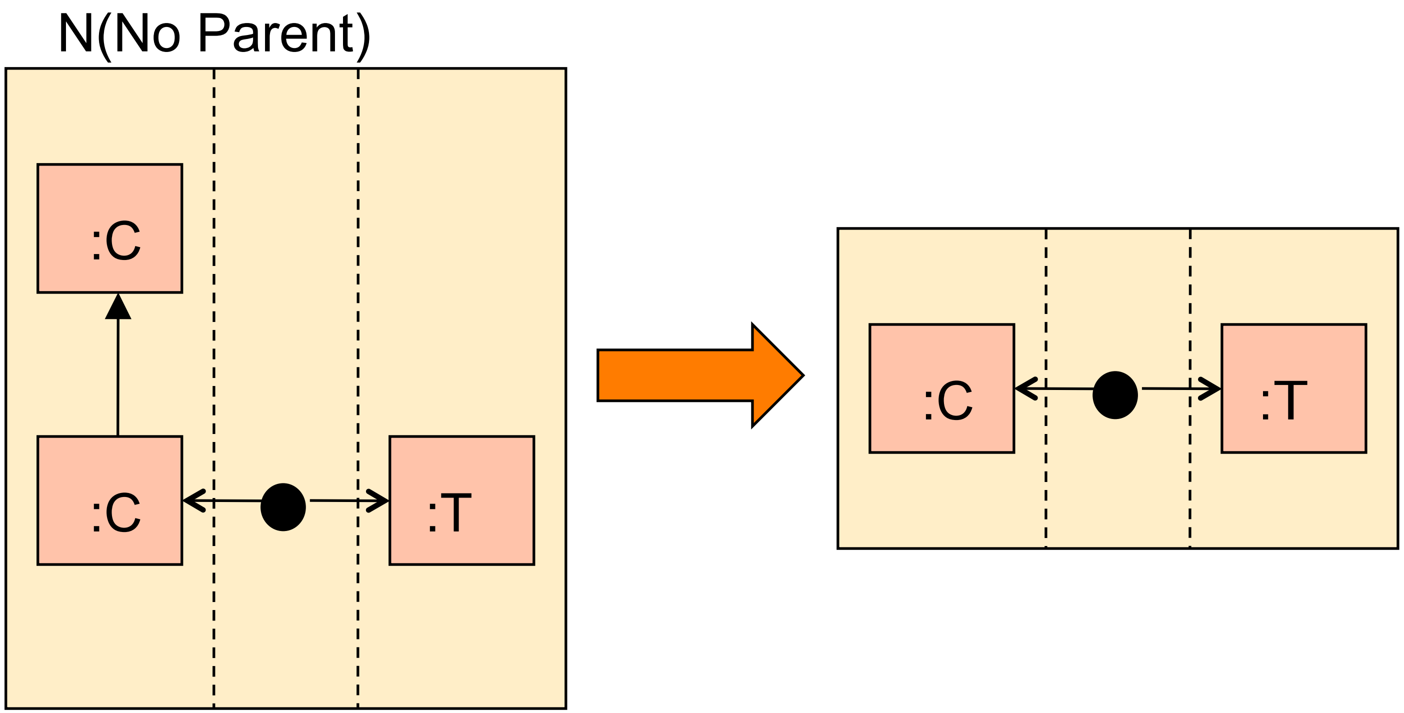
TNAC



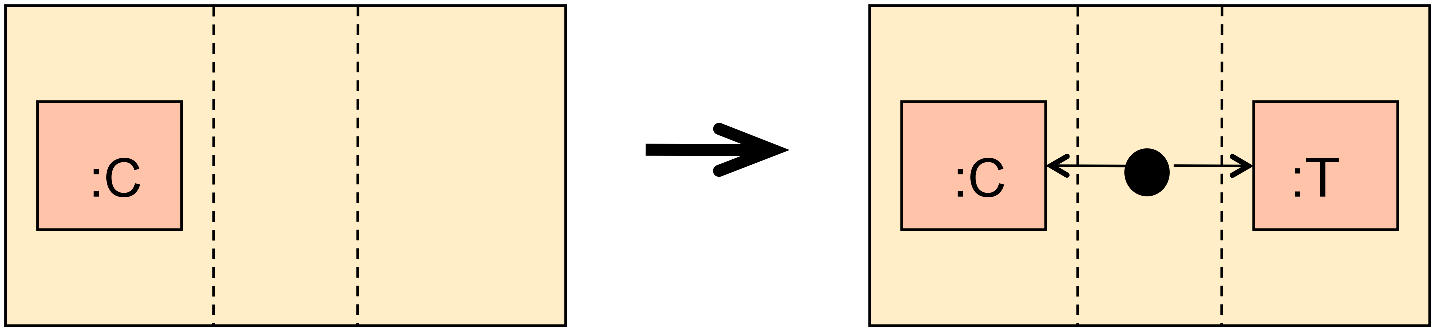
TNAC2



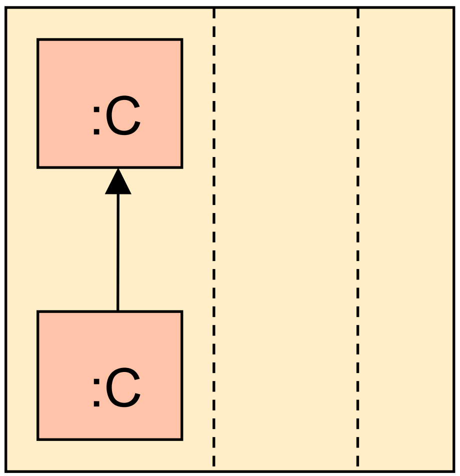
Examples



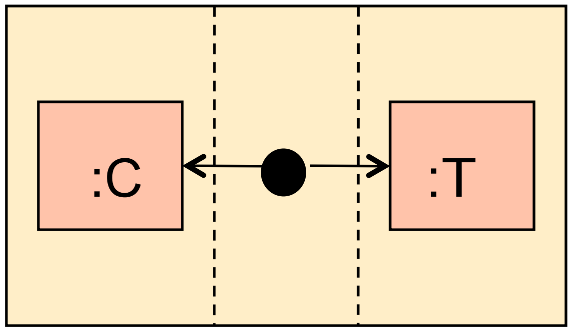
Examples



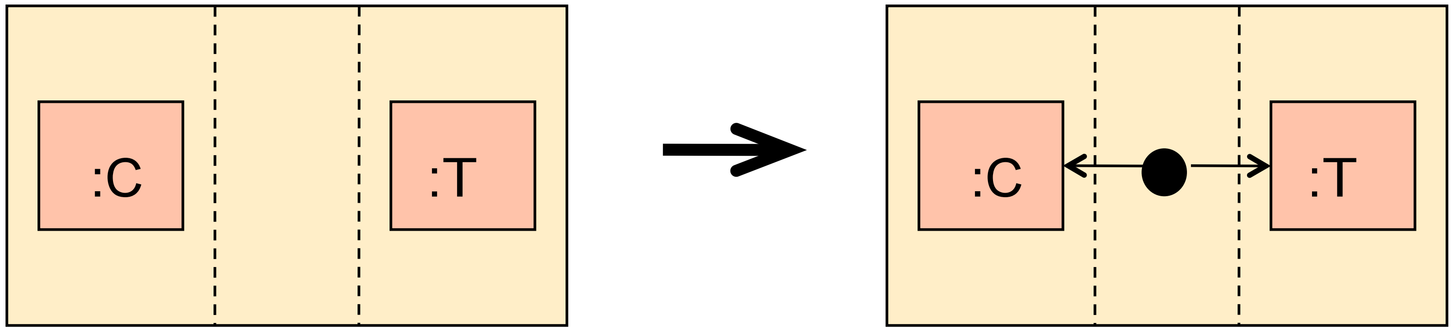
NAC1

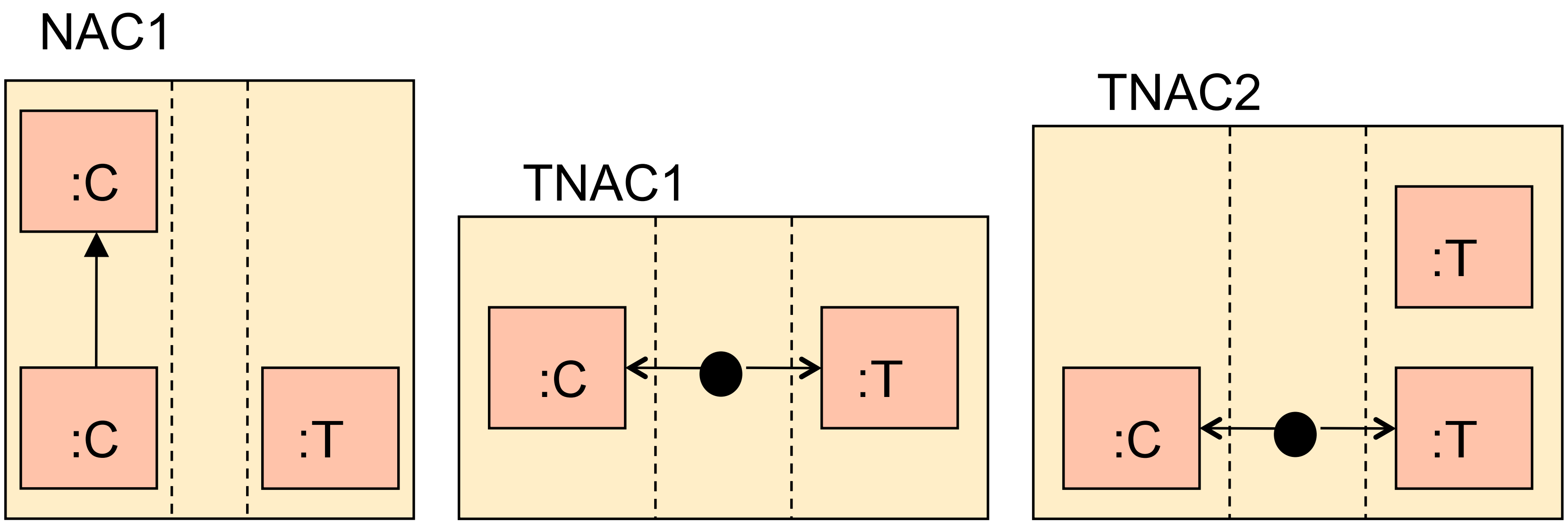


TNAC



Examples





Termination

- TTR(SP) is terminating.

Termination

A rule cannot be applied twice with the same source match:

$$\begin{tikzcd}
 & L \arrow[r] \arrow[d] & Q \arrow[d] \\
Q|_S \arrow[ur] \arrow[dr] & G \arrow[r] & G' \\
 & L \arrow[ur] &
\end{tikzcd}$$

Soundness

G is a normal form for TTR(SP) if and only if G (forward) satisfies all the positive patterns in SP.

SP-generated triple graphs

G is SP-generated if there are positive patterns $P_1, \ldots, P_n$ in SP, with $P_i = N(Q_i \rightarrow C_{ij})_{j \in J} \Rightarrow Q_i$, and monomorphisms $h_1, \ldots, h_n$, such that each h_i satisfies the preconditions $N(Q_i \rightarrow C_{ij})_{j \in J}$, and:

$$
\begin{tikzcd}
G|_S \arrow[drr] & Q_1 \arrow[dr, "h_1"] & \cdots & Q_n \arrow[dl, "h_n"'] \\
 & & G &
\end{tikzcd}
$$

are jointly surjective

Completeness 1

- G is SP-generated if and only if we can transform $G|_S$ into G using rules from TR(SP).

Strictly SP-generated triple graphs

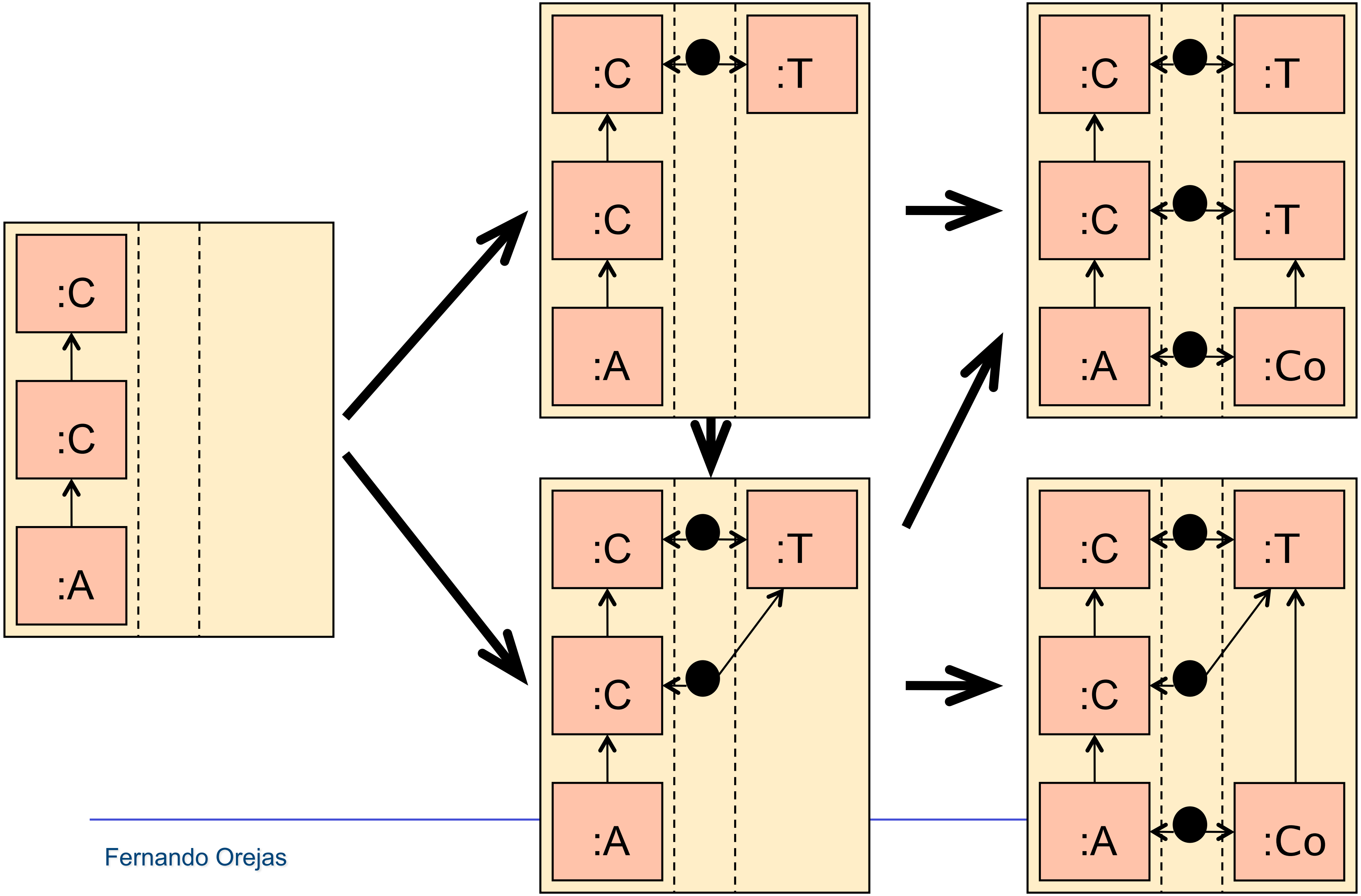
G is strictly SP-generated if:

- G is SP-generated
- For every pattern $P = N(\mathcal{P}_j)_{j \in J} \Rightarrow \mathcal{P}$ in SP, and all monomorphisms $f_1, f_2 : Q \rightarrow C$, if $(f_1)_S = (f_2)_S$ such that they both satisfy the preconditions in P, then $f_1 = f_2$.

Completeness 2

- If G is strictly SP-generated then G forward satisfies SP^+ if and only if we can transform $G|_S$ into G using rules from TTR(SP) and G is a normal form for TTR(SP).

Examples



Negative patterns

Using standard techniques, negative patterns can be converted into NACs of the rules associated to the given specification.

Conclusion and future work

We have seen:

- A formal framework to deal with model transformations
- A general method to specify transformations
- A sound and complete method to *compile* these specifications

Conclusion and future work

Some further work:

- Synchronized transformations
- Verification of transformations

Outline of the talk

1. Models and Model transformation
2. Specification of model transformations by triple patterns
3. Compiling patterns into transformation rules
   - Introduction to graph transformation
   - Translation of patterns into rules
   - Termination
   - Soundness
   - Completeness
4. Conclusion