Transformation of Rigid Structures

Reiko Heckel

joint work with
Pawel Sobocinski and Vincent Danos

Motivations

- Stochastic graph transformation good for modelling *adaptive networks*, where topology and state changes are interdependent
  - biological systems
  - social or technical networks
- Need a *constructive approach* towards model creation, refinement and analysis following example of *kappa*

Goals

- Make it easy to define domain-specific languages, including constraints for
  - patterns and state graphs
  - rules and matches
- Axiomatise conditions for constructive approach
- Be able to *engineer* approaches to satisfy such conditions

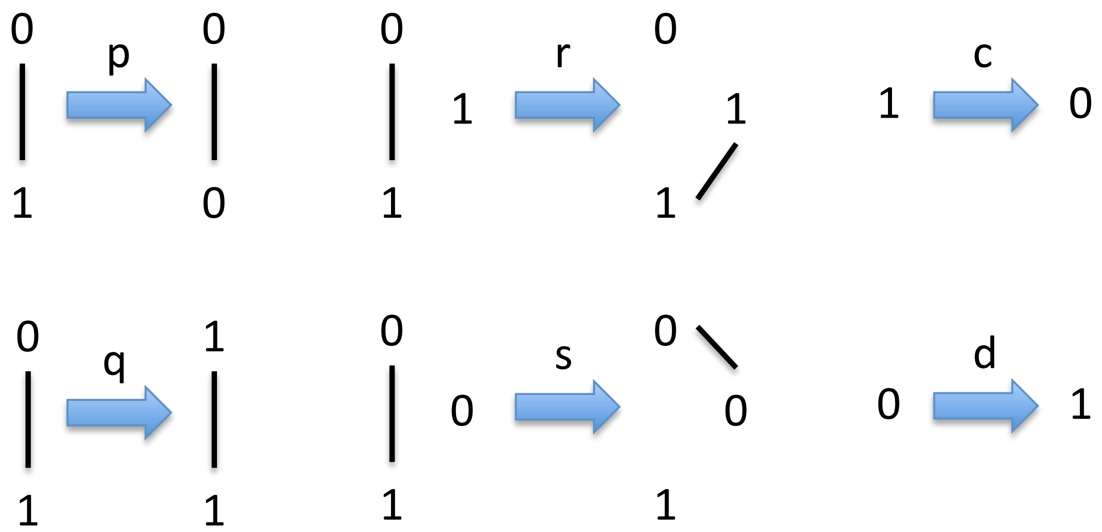
Social Network Model

Individuals hold one of two opinions (votes 0 or 1), coevolving with their connections:

- If two connected individuals hold different opinions,
  - one is converted to the opinion of the other, or
  - their link is broken and one makes a new connection to a random individual of the same opinion
- Individuals can change opinions spontaneously

[Graph Fission in Evolving Voter Model, Durret et al., 2012]

Intuitively



- p, q: be converted
- r, s: split up and reconnect
- c, d: change spontaneously

Questions

How does the number of occurrences of such patterns change

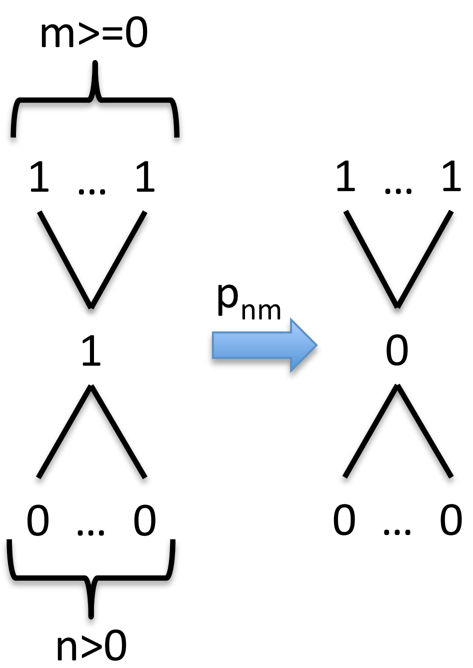
- with the application of individual rules?
- over time?

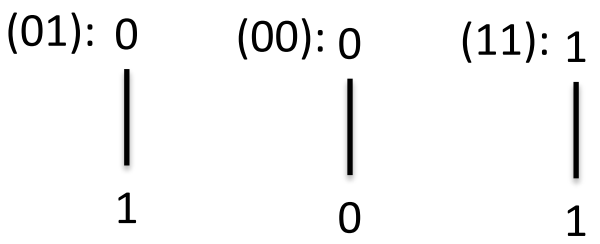
Given observed number of occurrences of patterns such as

- 0 or 1 nodes
- 01, 11, or 00 links

how to consistently assign rates to rules?

How do p, q, r, s change numbers of occurrences of patterns?



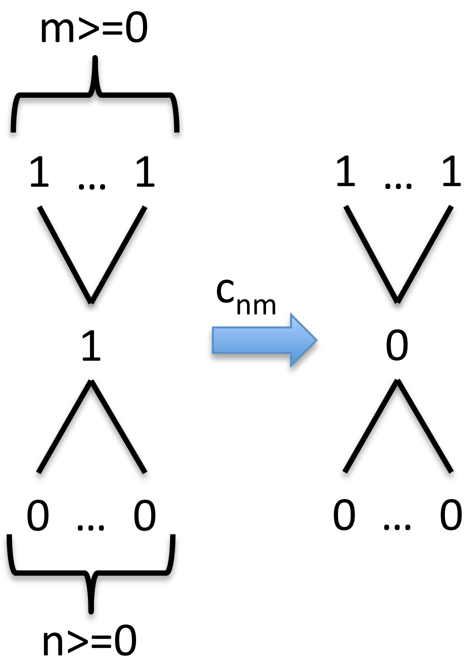


- p_{nm} for n+m <= max degree
  - creates m-n occurrences of (01)
  - creates 2n occurrences of (00)
  - destroys 2m occurrences of (11)

- dually for q_{nm}

- r, s each destroy one (01) and create two (00), (11), resp.

How do c, d change numbers of occurrences of patterns?



- c_{nm} for n+m <= max degree
  - creates m-n occurrences of (01)
  - creates 2n occurrences of (00)
  - destroys 2m occurrences of (11)

- dually for d_{nm}

Depends on the context? But can we produce a refined set of rules, where these numbers are fixed?

Petrification

Produce balanced model: each rule creates or destroys fixed number of occurrences

- How to prevent application of an instance rule to larger context?
- →Constraints on matches preventing certain contexts: *open maps*

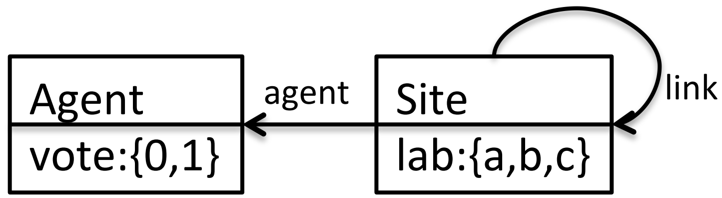
Preserve stochastic behaviour: (functional) stochastic bisimilation between original and refined model

- How to guarantee 1-to-1 between rule/match pairs of original and refined models?
- →Constraints on graphs to guarantee unique extension of match to added context: *rigidity*

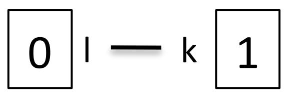
Methodology

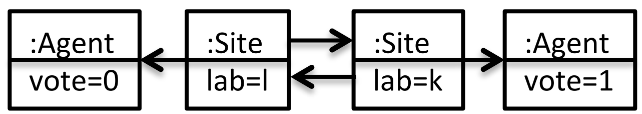
1. Define explicit model: type graph and rules
2. Define constraints: patterns and states
3. Select minimal permissible rules
4. Derive constraints for matches
5. Petrify

1. Explicit Model: Type Graph

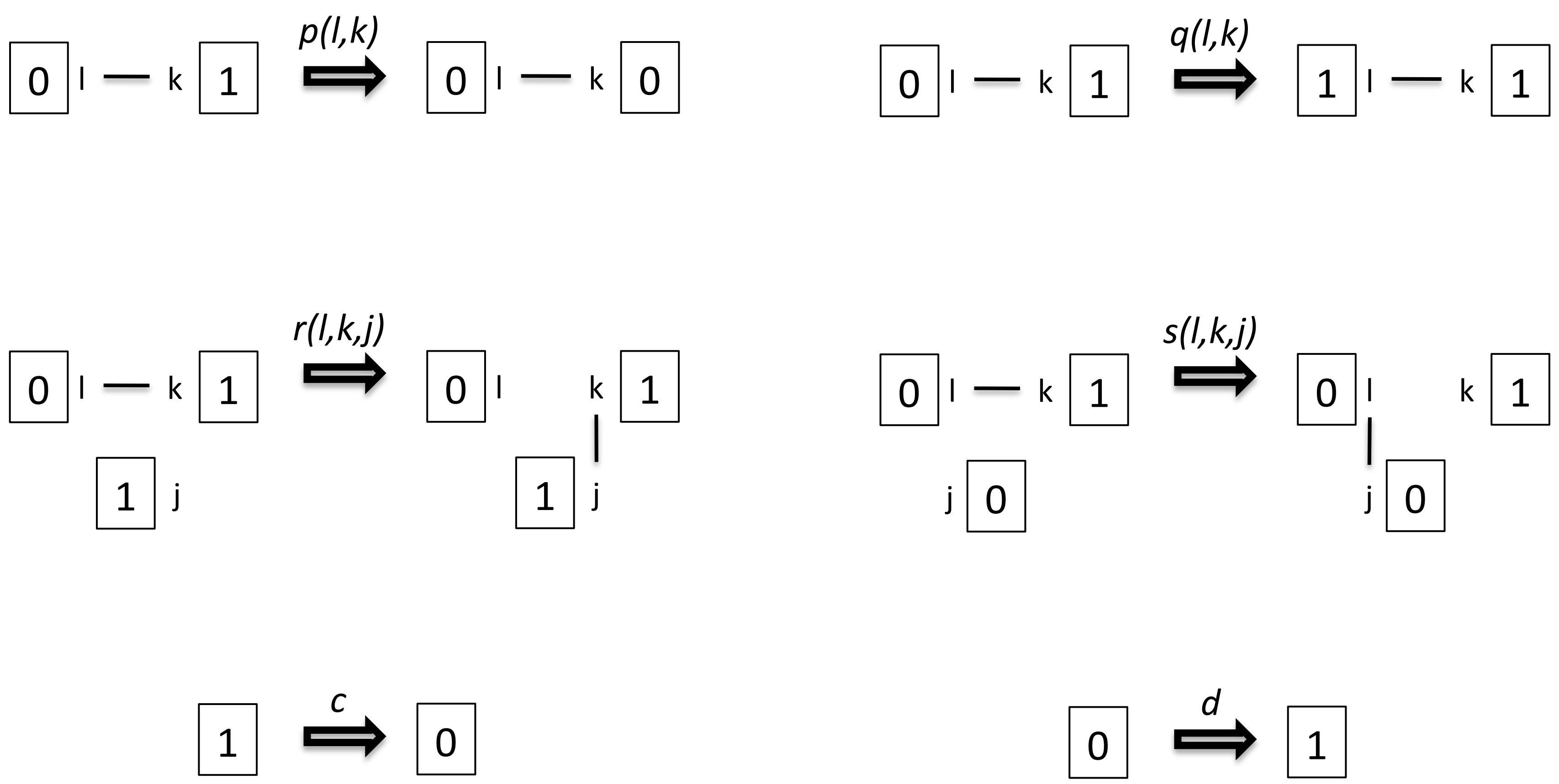


- Add labelled sites to act as ports to control degree, support rigidity
- Use diagrammatic notation

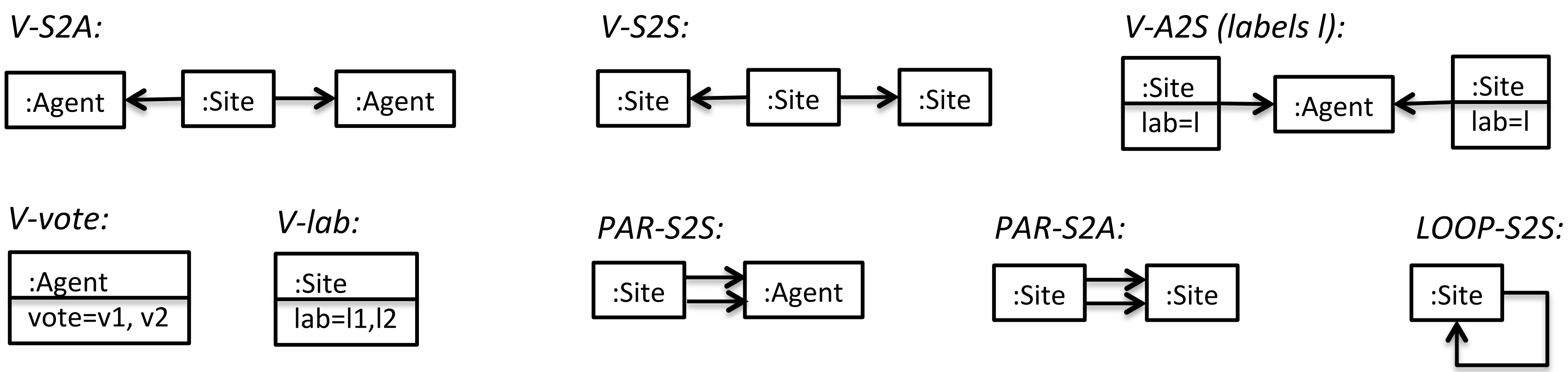




2. Explicit Model: Rules



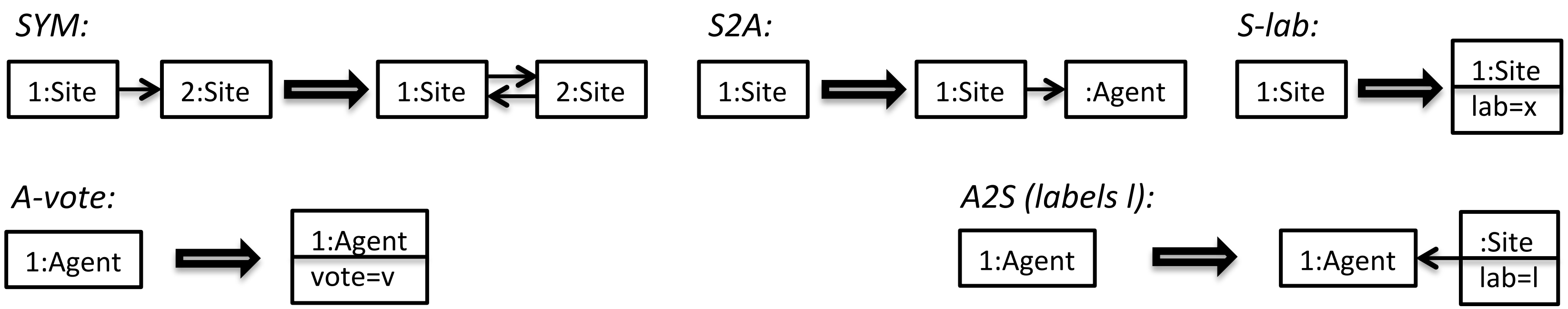
2. (Negative) Constraints on Patterns



Given adhesive ambient *category C of structures*, forbidden substructures S specify *full subcategory PC of patterns*

Taking as forbidden substructures all *non-rigid atomic structures* guarantees rigidity of *PC*

2. (Positive) Constraints on States



Patterns can be incomplete, states cannot

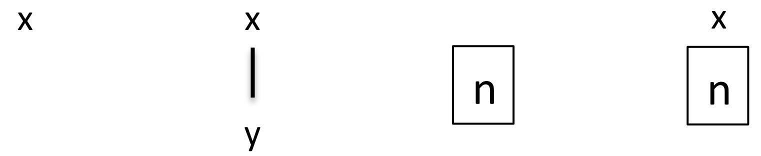
Category of states *SC full subcategory of PC*

3. Select Minimal Permissible Rules

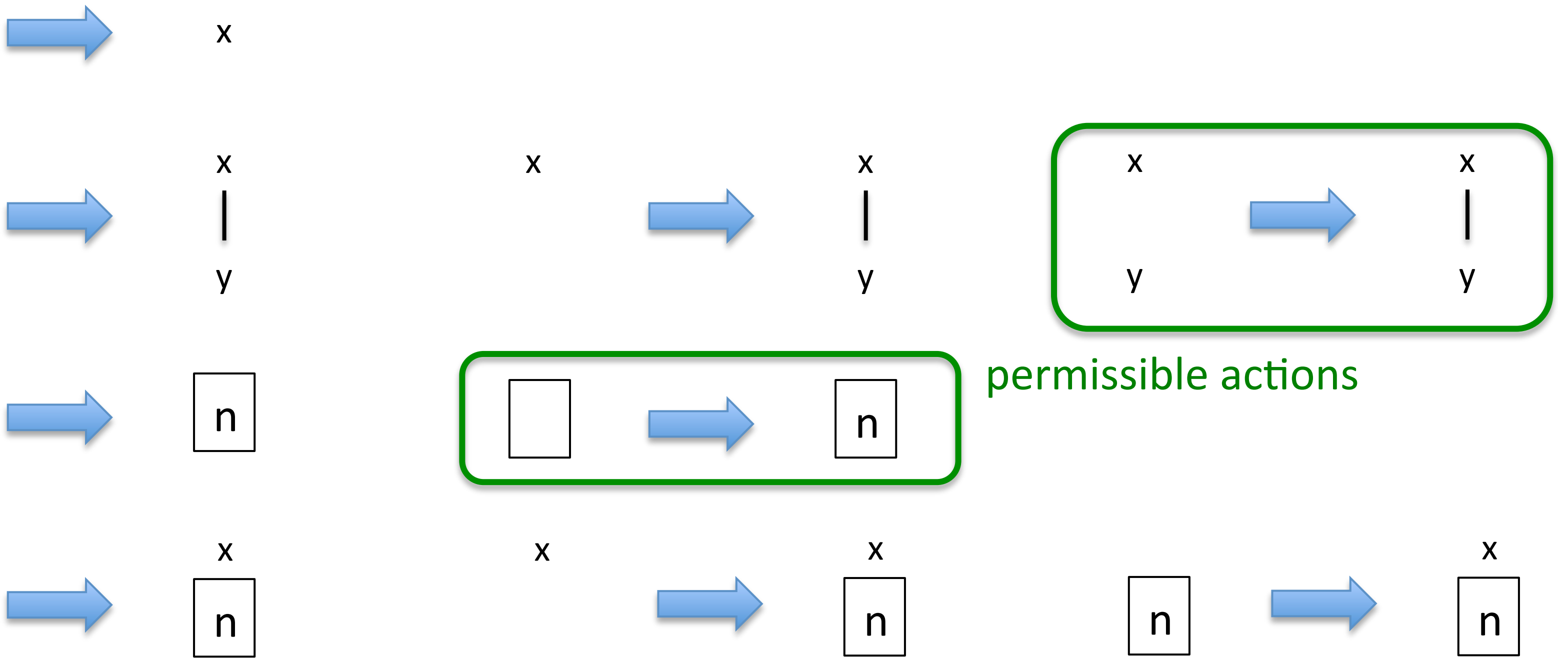
(creation only, delete actions symmetrical)

- Atomic graphs

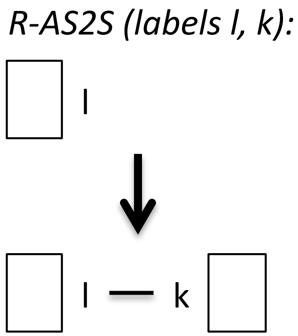
  x, y, z in {a, b, c} distinct

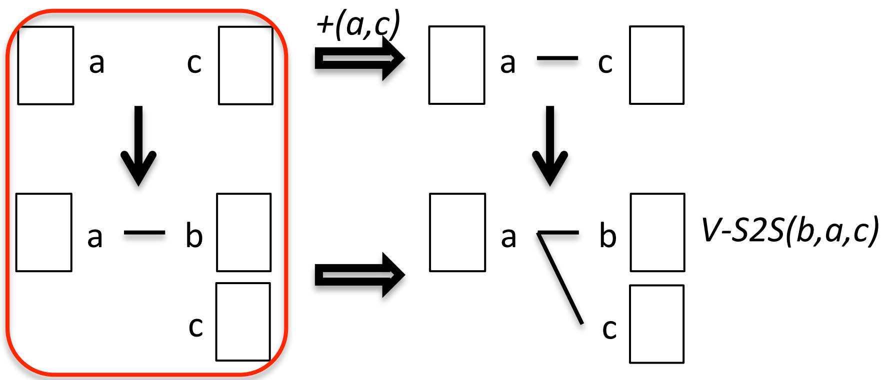


- Minimal creation rules:



4. Derive Constraints for Matches

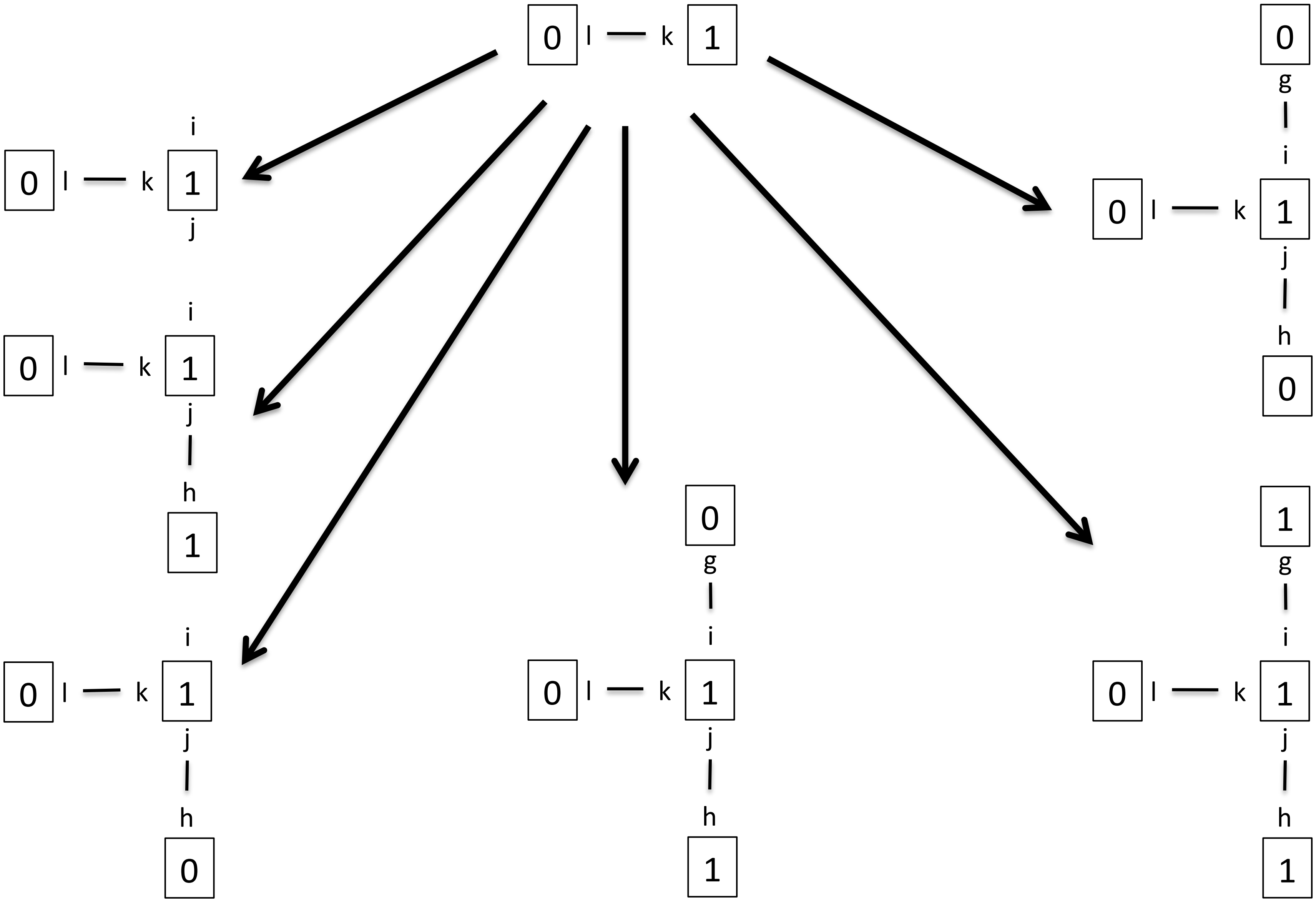


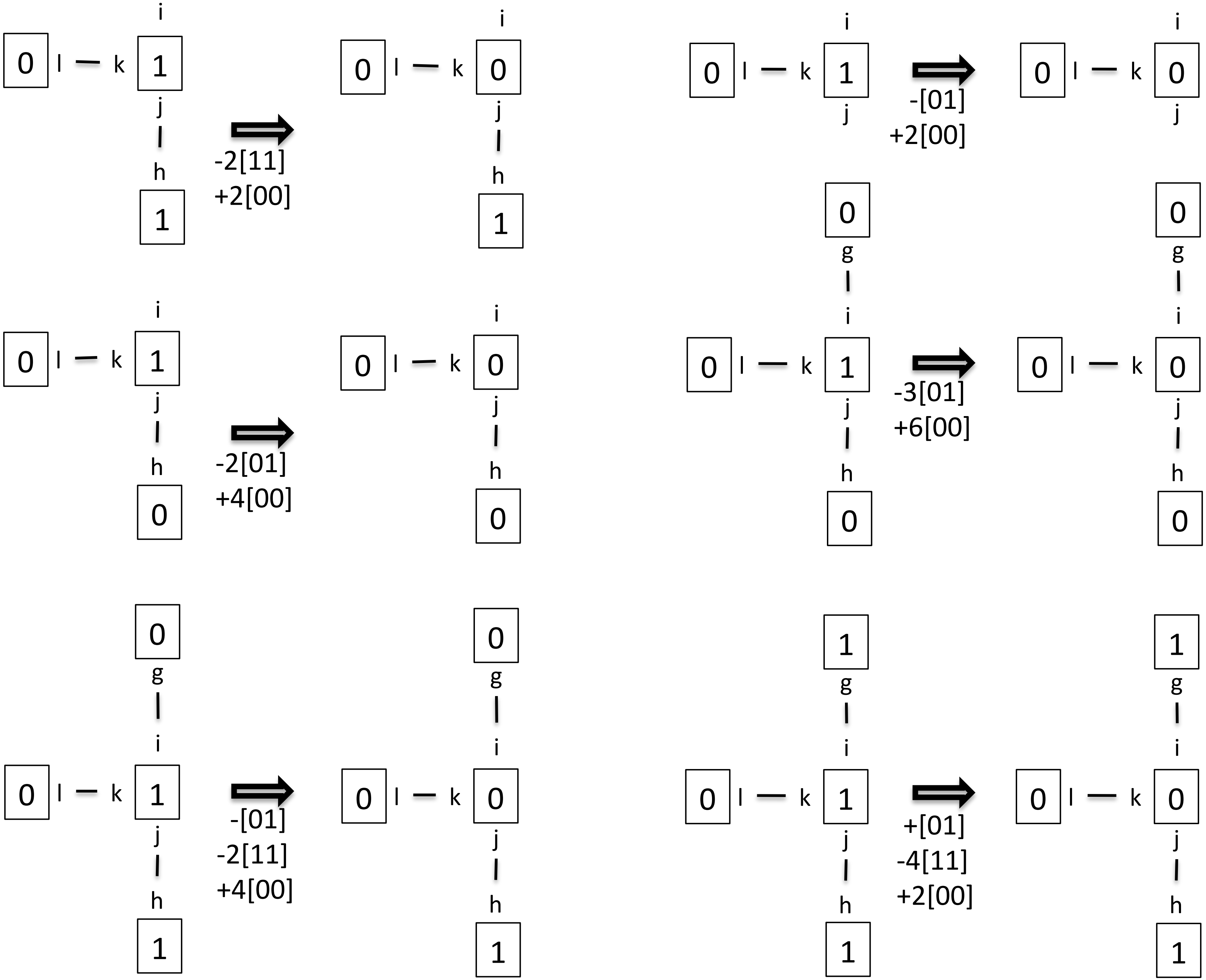


Path category, so that matches are *open maps*.

- ensure that negative pattern constrains are preserved
- control embedding into context

5. Petrify




-2[11]
+2[00]
-[01]
+2[00]
-2[01]
+4[00]
-3[01]
+6[00]
-[01]
-2[11]
+4[00]
+[01]
-4[11]
+2[00]

Methodology

1. Define explicit model: type graph and rules
2. Define constraints: patterns and states
3. Select minimal permissible rules
4. Derive constraints for matches
5. Petrify

Conclusion

We can now analyse the resulting PT net by simulation, deriving differential equations, using it for parameter fitting, etc.

Under meaningful restrictions this works in general adhesive categories.

Questions:

- How to systematically rigidify an existing non-rigid model?
- How to handle attributes other than by instantiation?