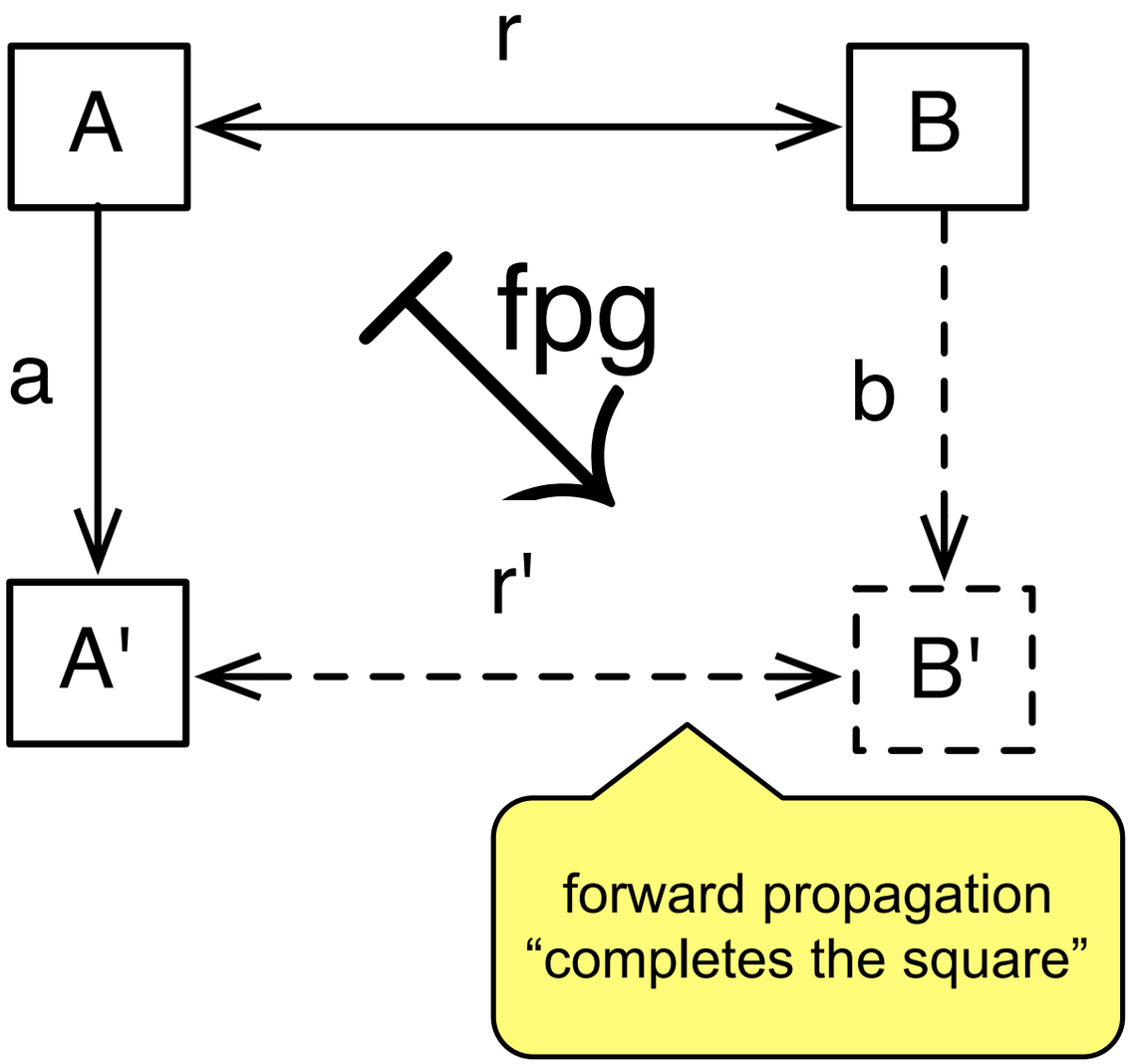
BX with Triple Graph Grammars

# PART 2: SYMMETRIC DELTA LENSES

# Delta Propagation

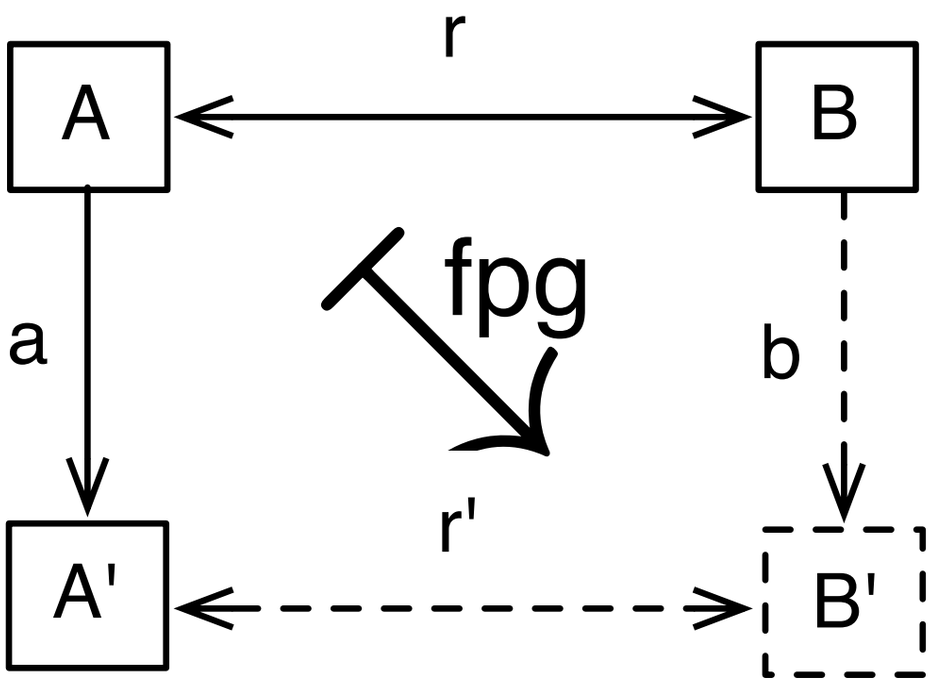
Nodes are models, arrows are deltas, dashed outline indicates derived elements

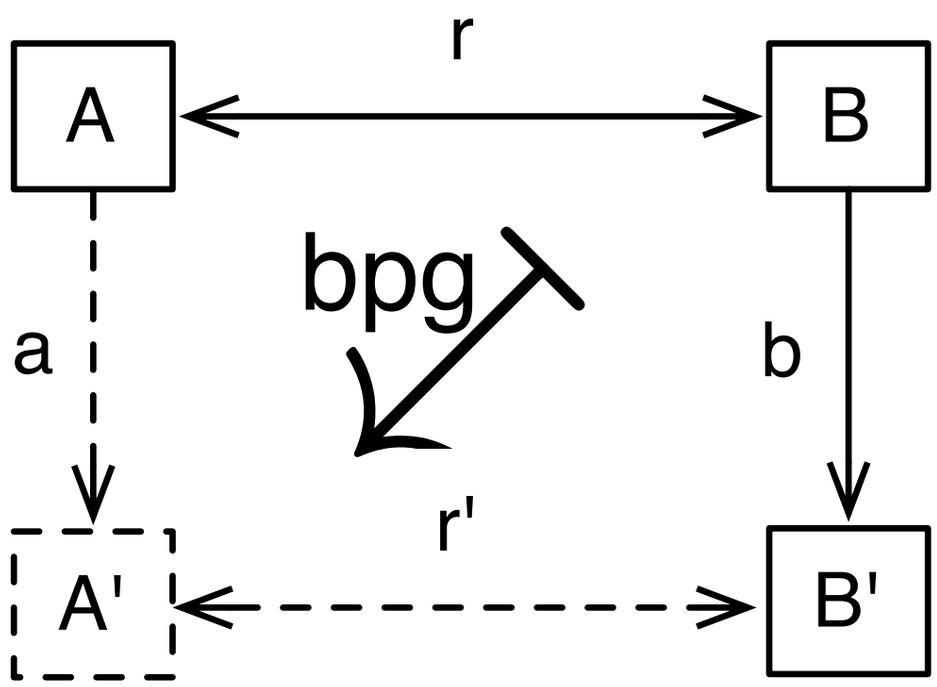



Note: This is NOT a pushout square!

# Delta Propagation

Nodes are models,
arrows are deltas,
dashed outline indicates
derived elements

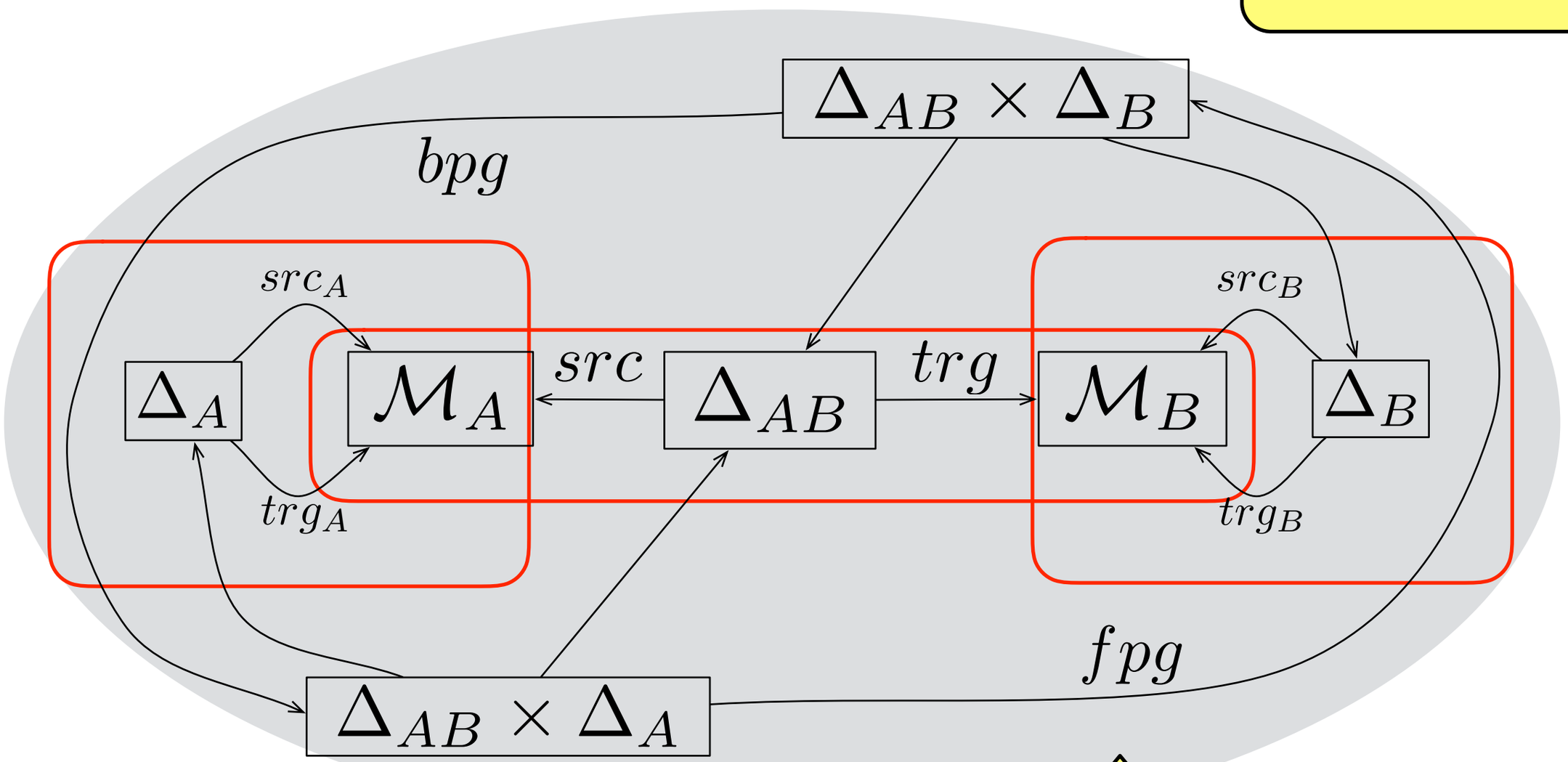



this (**model synchronisation**) is of course a simplification — we'll have to deal with **concurrent** delta propagation one day (**model integration**)

# Symmetric Delta Lenses (SDL)

Nodes are sets, arrows are total functions




… with all incidence conditions indicated on previous slide

$$\mathbf{A} \xleftarrow{\mathbf{R}} \mathbf{B}$$

pair of fpg and bpg functions forms a "**lens**" shape (with some imagination)

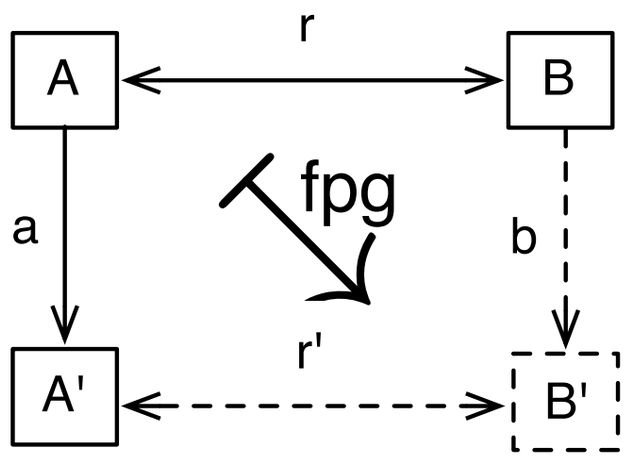
# Specifying SDLs

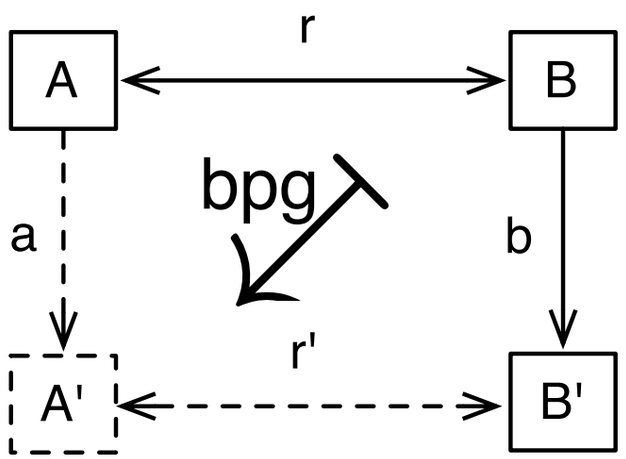
$$\mathbf{A} \xleftrightarrow{\mathbf{R}} \mathbf{B}$$

given a triple space, how do we specify an SDL?

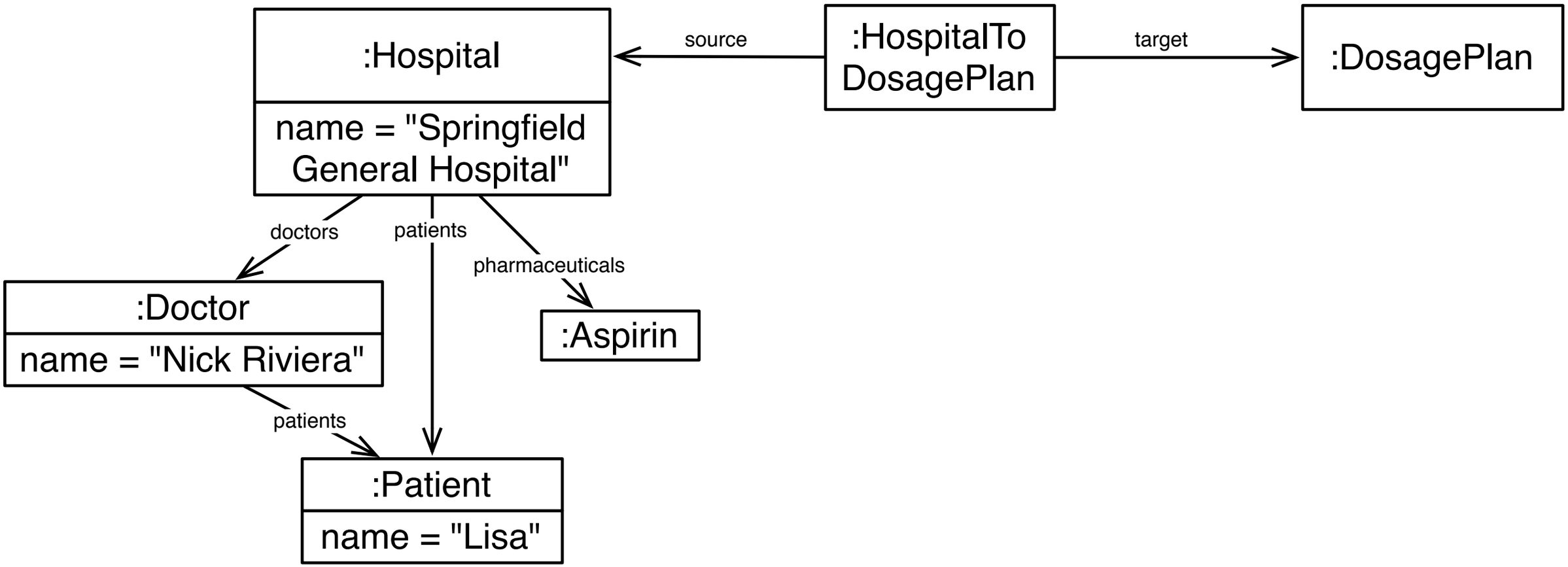
**Idea 1:**

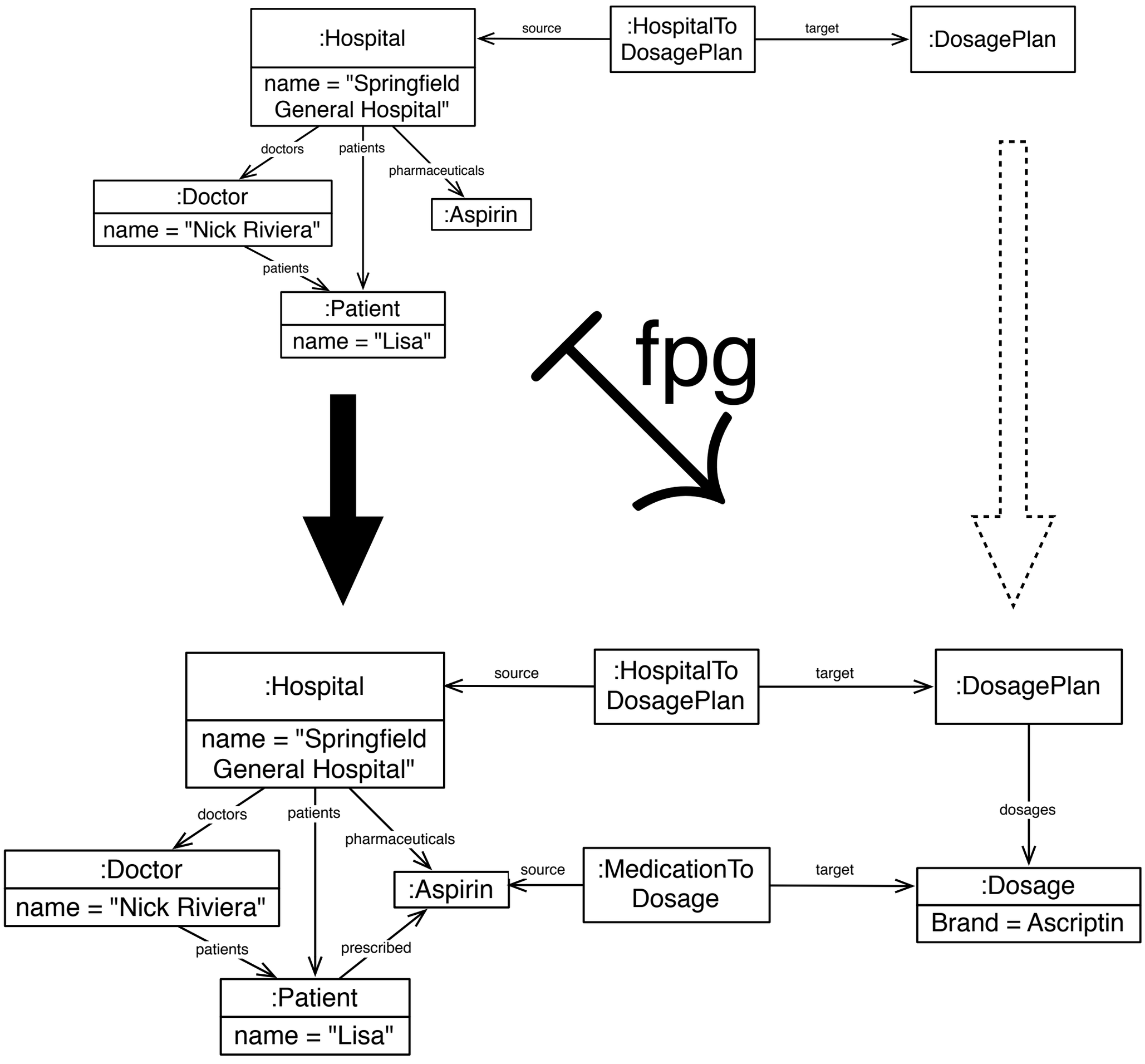
Enumerate all squares:



# Exhaustive Enumeration



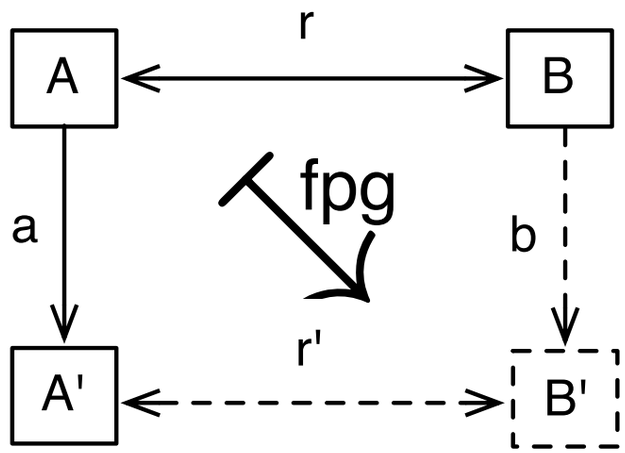

:Hospital
name = "Springfield General Hospital"
source
:HospitalTo DosagePlan
target
:DosagePlan
doctors
patients
pharmaceuticals
:Doctor
name = "Nick Riviera"
:Aspirin
patients
:Patient
name = "Lisa"
fpg
:Hospital
name = "Springfield General Hospital"
source
:HospitalTo DosagePlan
target
:DosagePlan
doctors
patients
pharmaceuticals
dosages
:Doctor
name = "Nick Riviera"
:Aspirin
source
:MedicationTo Dosage
target
:Dosage
Brand = Ascriptin
patients
prescribed
:Patient
name = "Lisa"

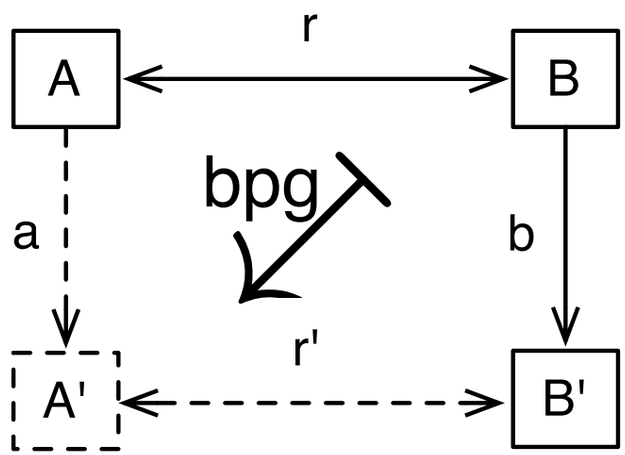
# Exhaustive enumeration

**Idea 1:**

Enumerate all squares:

not really feasible…
but why not?



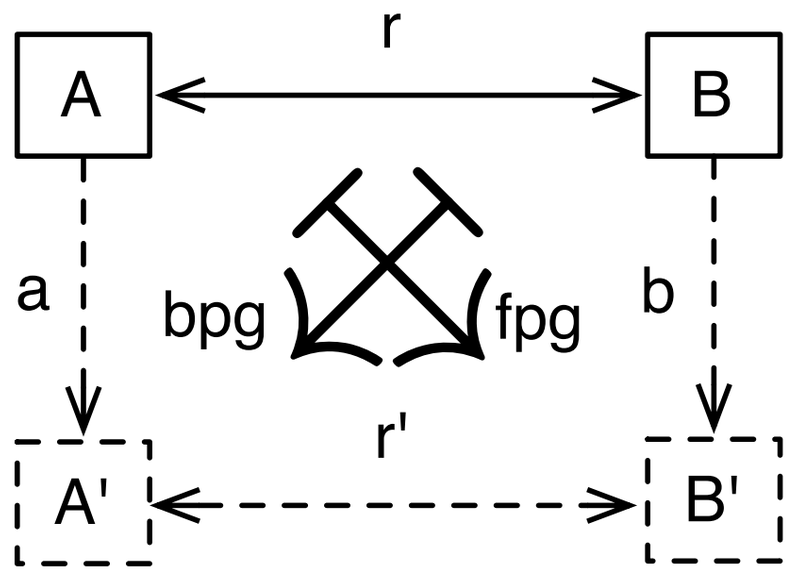



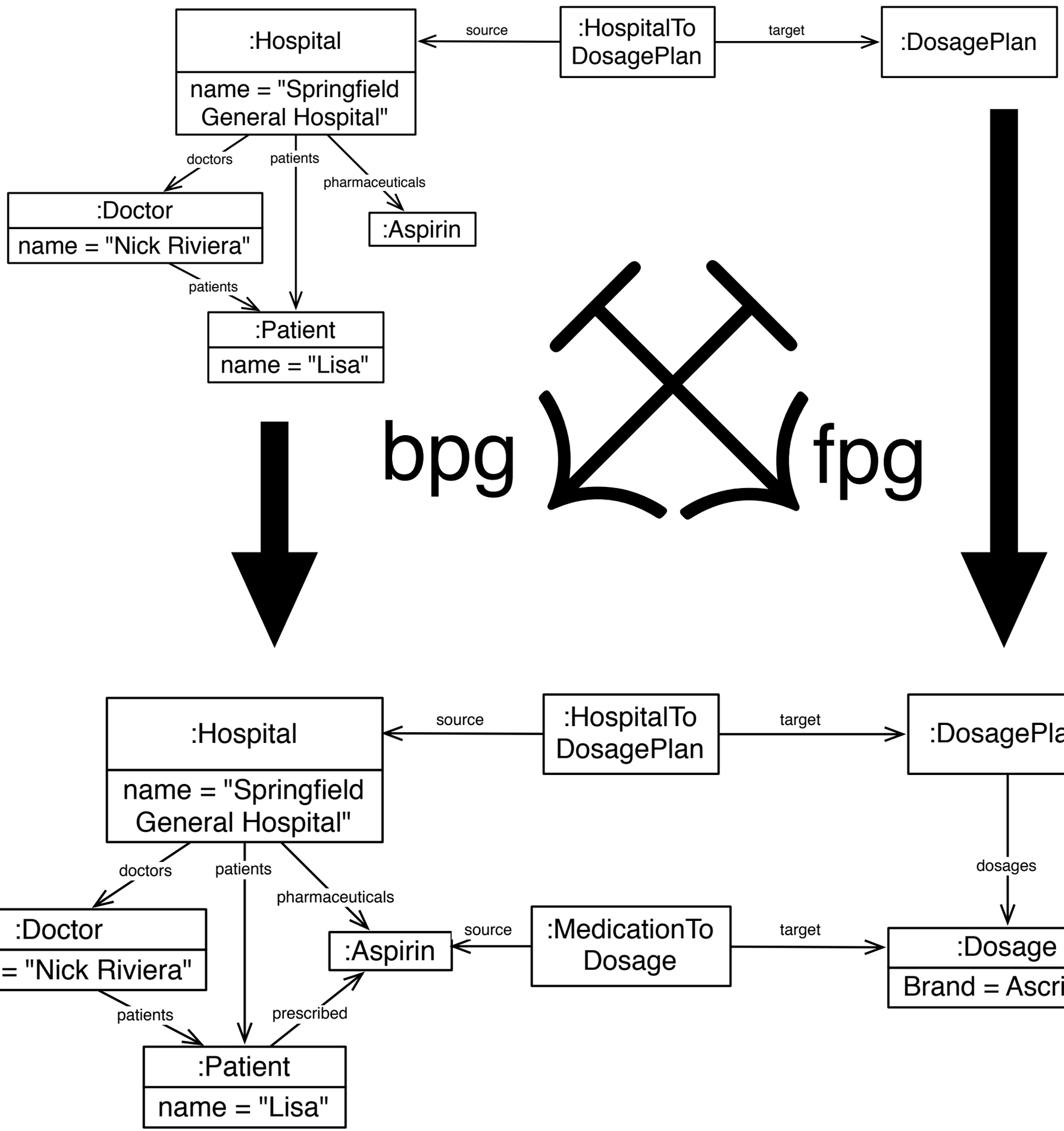

we typically have infinitely many such squares

# **Simultaneous**, exhaustive enumeration

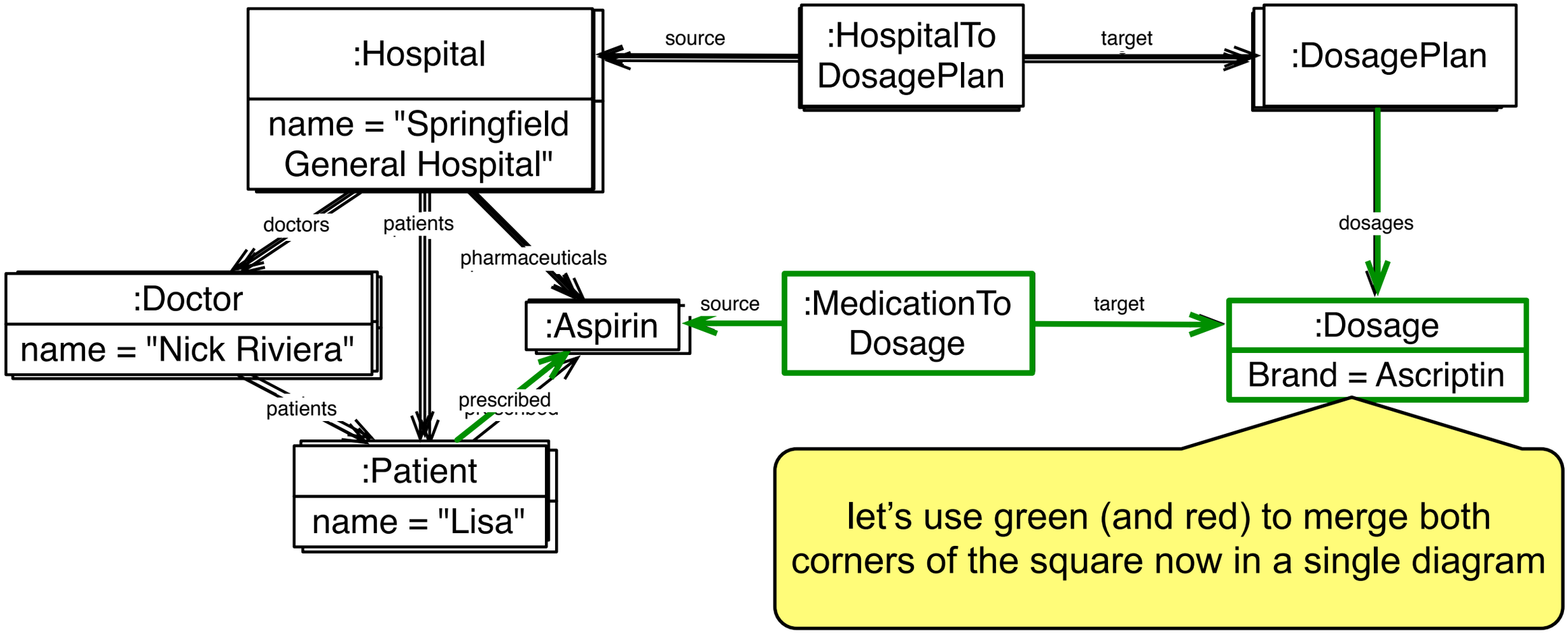
**Idea 2:**

Enumerate all squares representing combined **fpg** and **bpg** squares




:Hospital
name = "Springfield General Hospital"
source
:HospitalTo DosagePlan
target
:DosagePlan
doctors
patients
pharmaceuticals
:Doctor
name = "Nick Riviera"
:Aspirin
patients
:Patient
name = "Lisa"
bpg
fpg
:Hospital
name = "Springfield General Hospital"
source
:HospitalTo DosagePlan
target
:DosagePlan
doctors
patients
pharmaceuticals
dosages
:Doctor
name = "Nick Riviera"
:Aspirin
source
:MedicationTo Dosage
target
:Dosage
Brand = Ascriptin
patients
prescribed
:Patient
name = "Lisa"

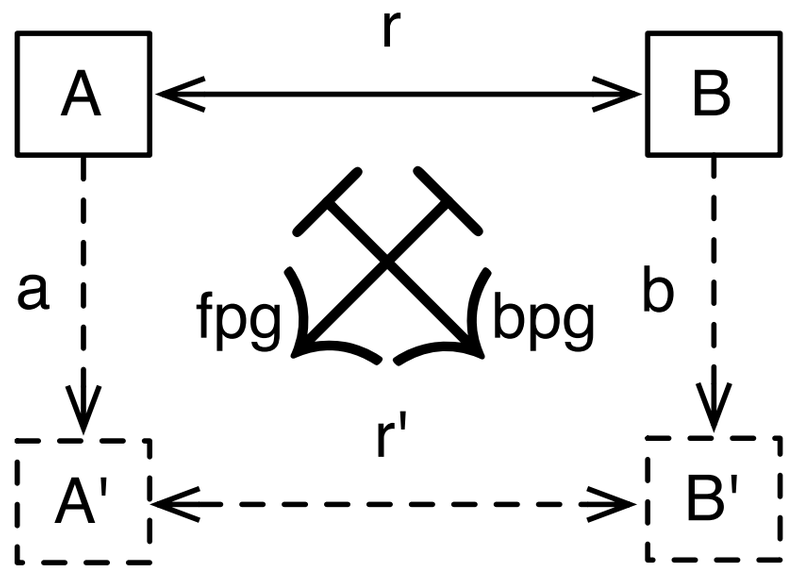
# **Simultaneous**, exhaustive enumeration



# **Simultaneous**, exhaustive enumeration

**Idea 2:**

Enumerate all squares representing combined **fpg** and **bpg** squares




still infeasible, but quite a nice idea…

Why?

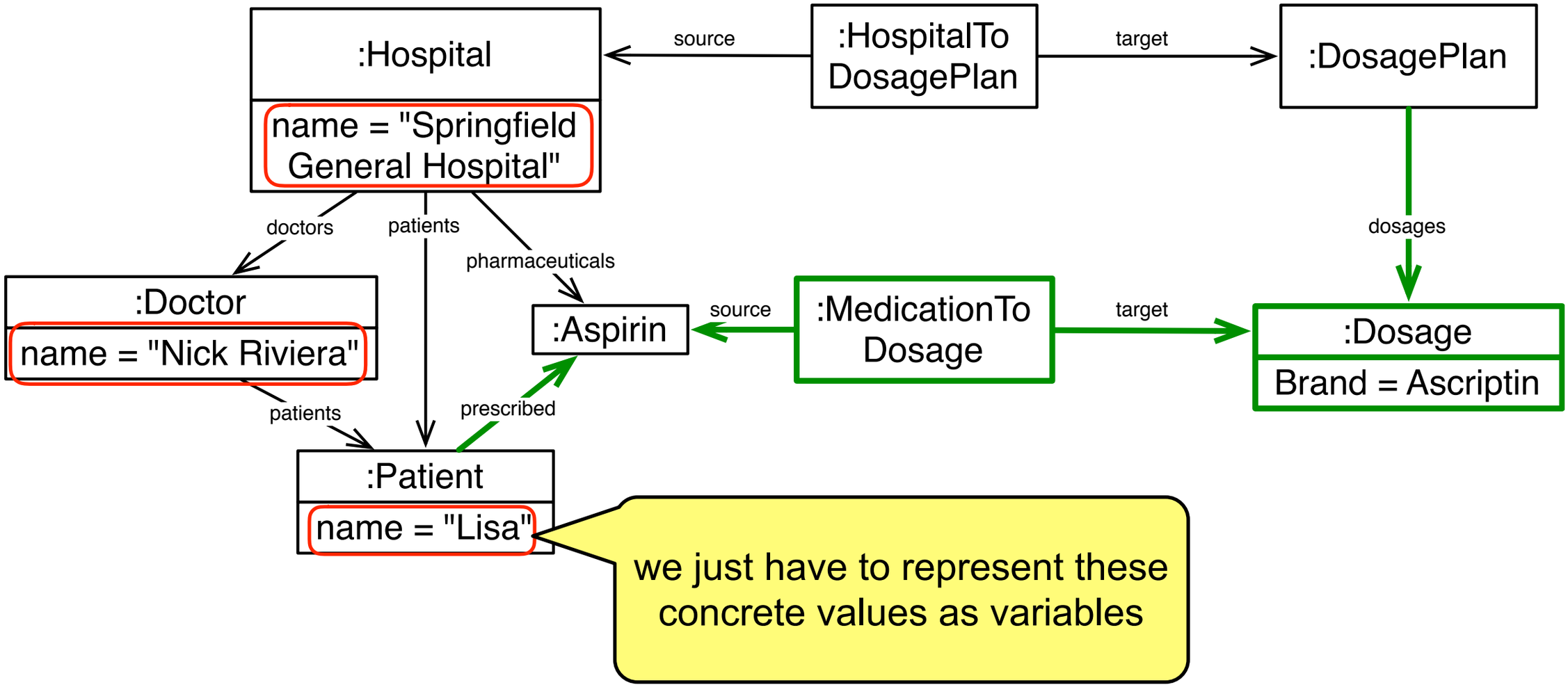
- promotes “symmetrical” thinking and avoids favouring either **fpg** or **bpg**
- easier to enforce “good” lens specifications
- we obviously only have to enumerate half of all squares (still typically infinitely many)

# Simultaneous **rules**

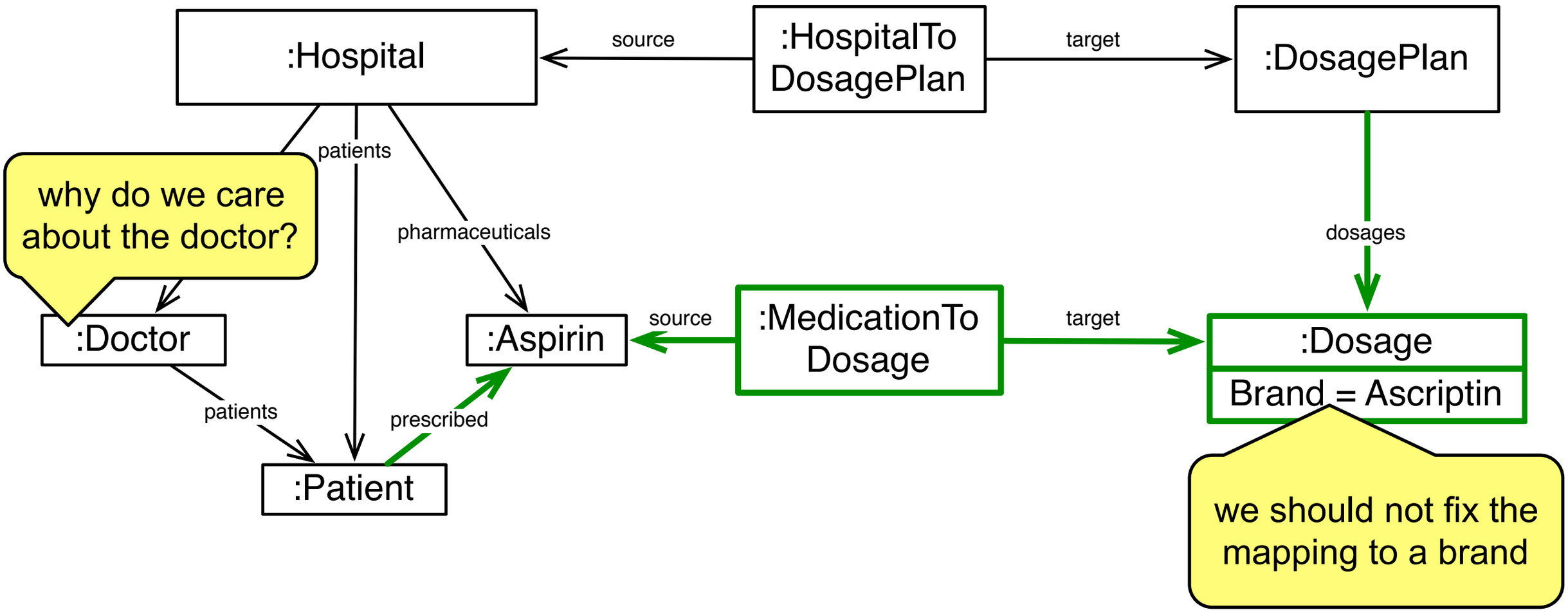
**Idea 3:**

specify infinitely many deltas using finitely many **rules** (precondition and postcondition graph **patterns**)

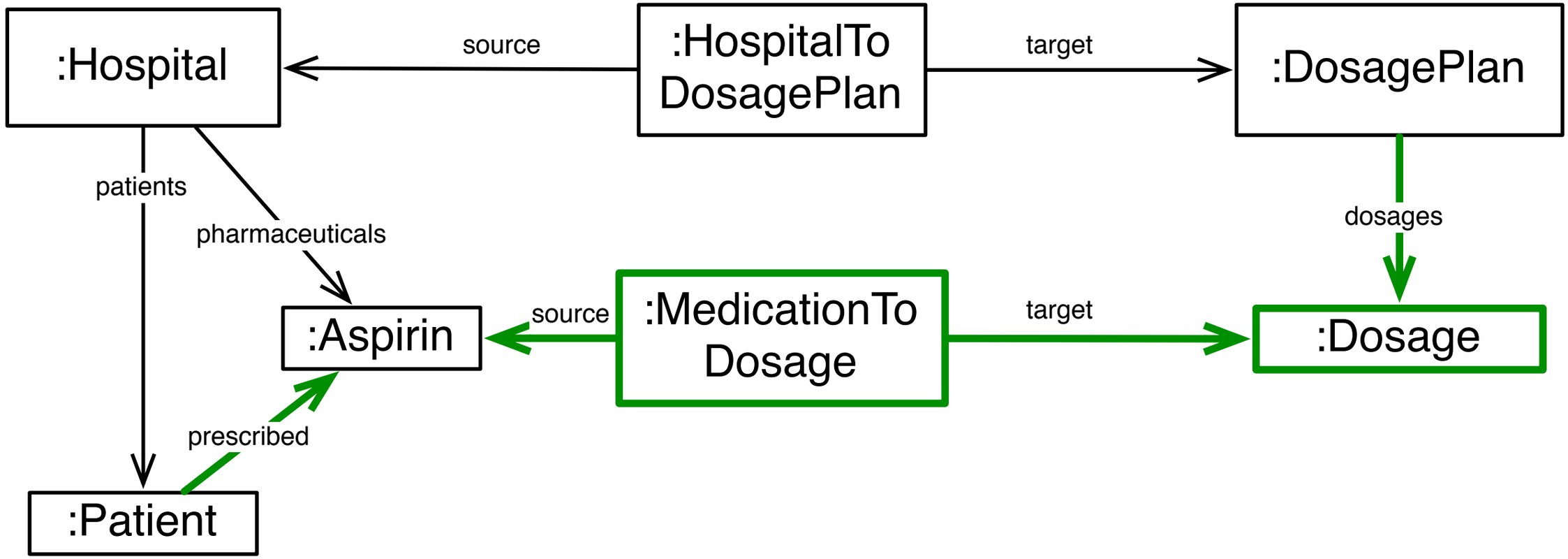
# Simultaneous **rules**



# Simultaneous **rules**



# Simultaneous **rules**



# Simultaneous **rules**

**Idea 3:**

specify infinitely many deltas using finitely many **rules**
(precondition and postcondition graph **patterns**)

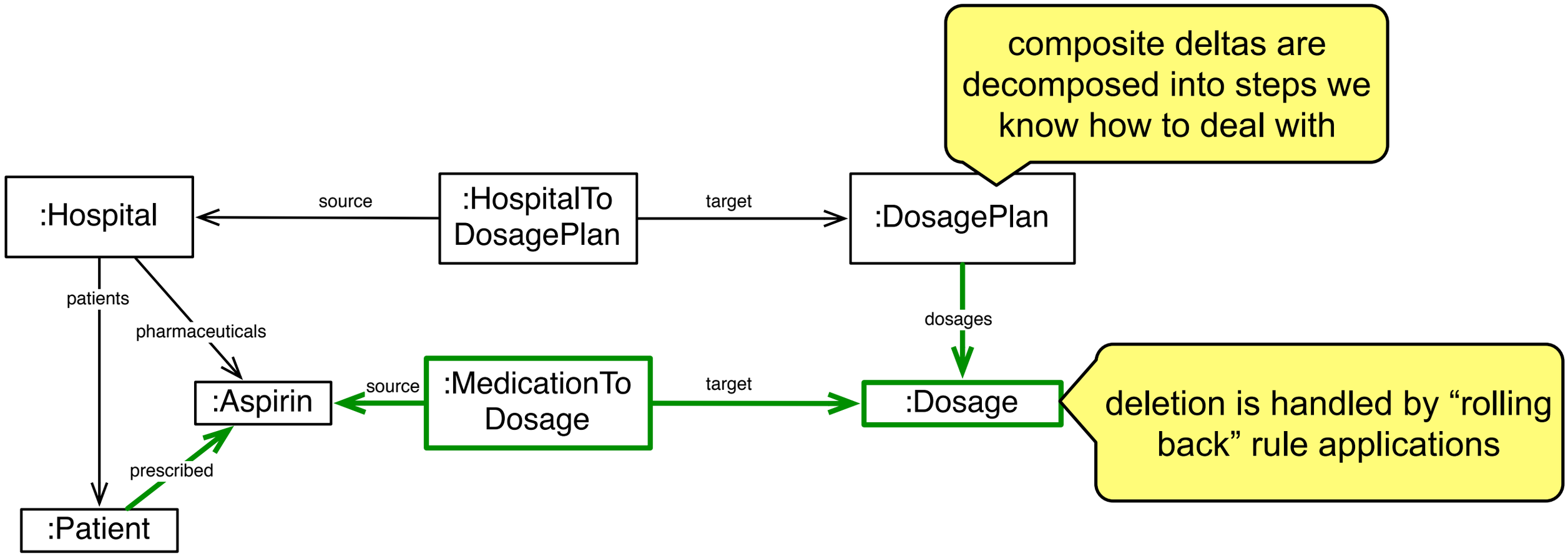
very important idea, as
we've finally made the jump
to a **finite** specification

specifying **all** deltas this way
is still a lot of work …

# Simultaneous, **monotonic** rules

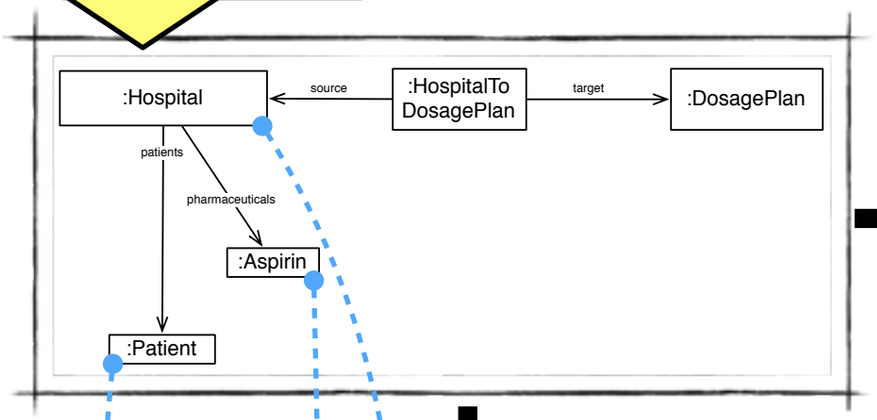
**Idea 4:**

only specify monotonic rules, i.e., only describing purely creating deltas



# Concrete deltas are derived via **rule application**

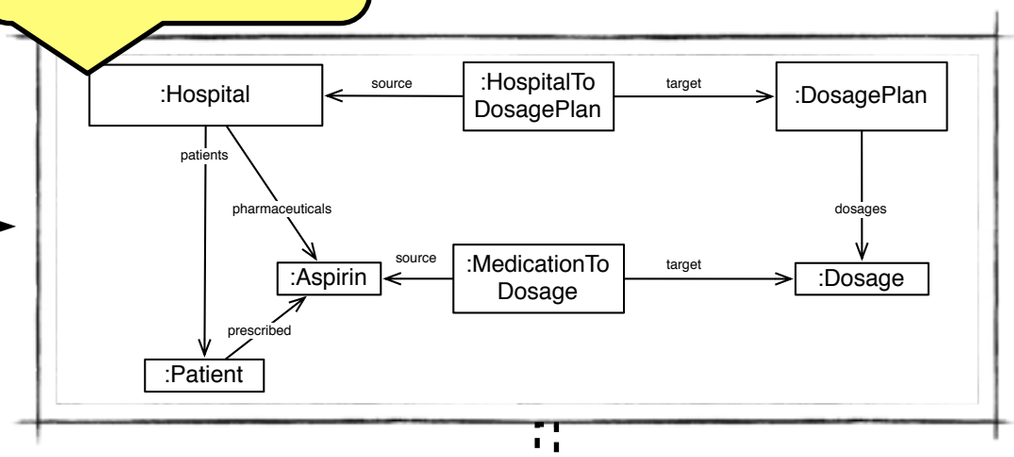
precondition (LHS)




p

rule (production)

postcondition (RHS)

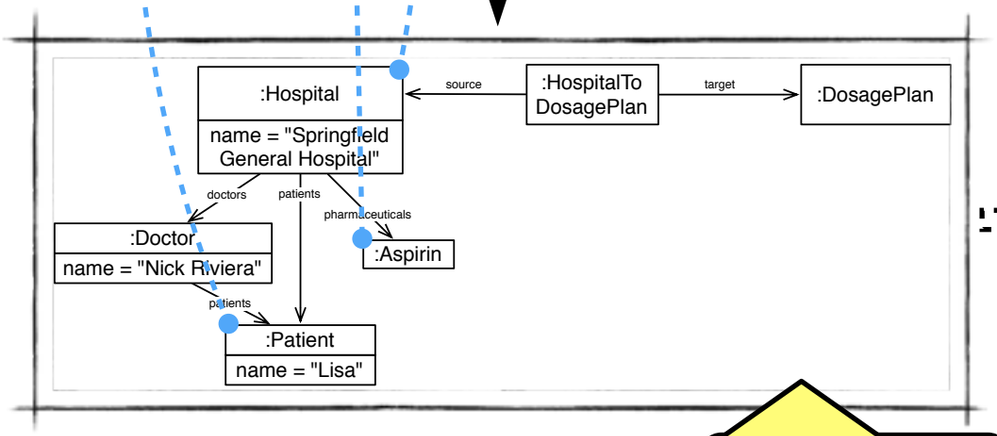



m

match (morphism)

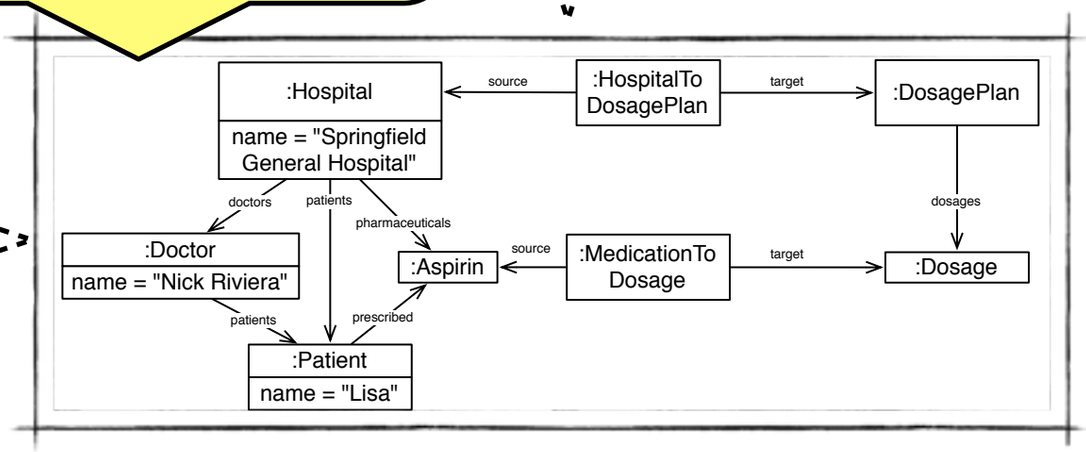
result is constructed via a disjoint union of the RHS and host graph, and a subsequent gluing of all elements with common image in the LHS

m’




p’

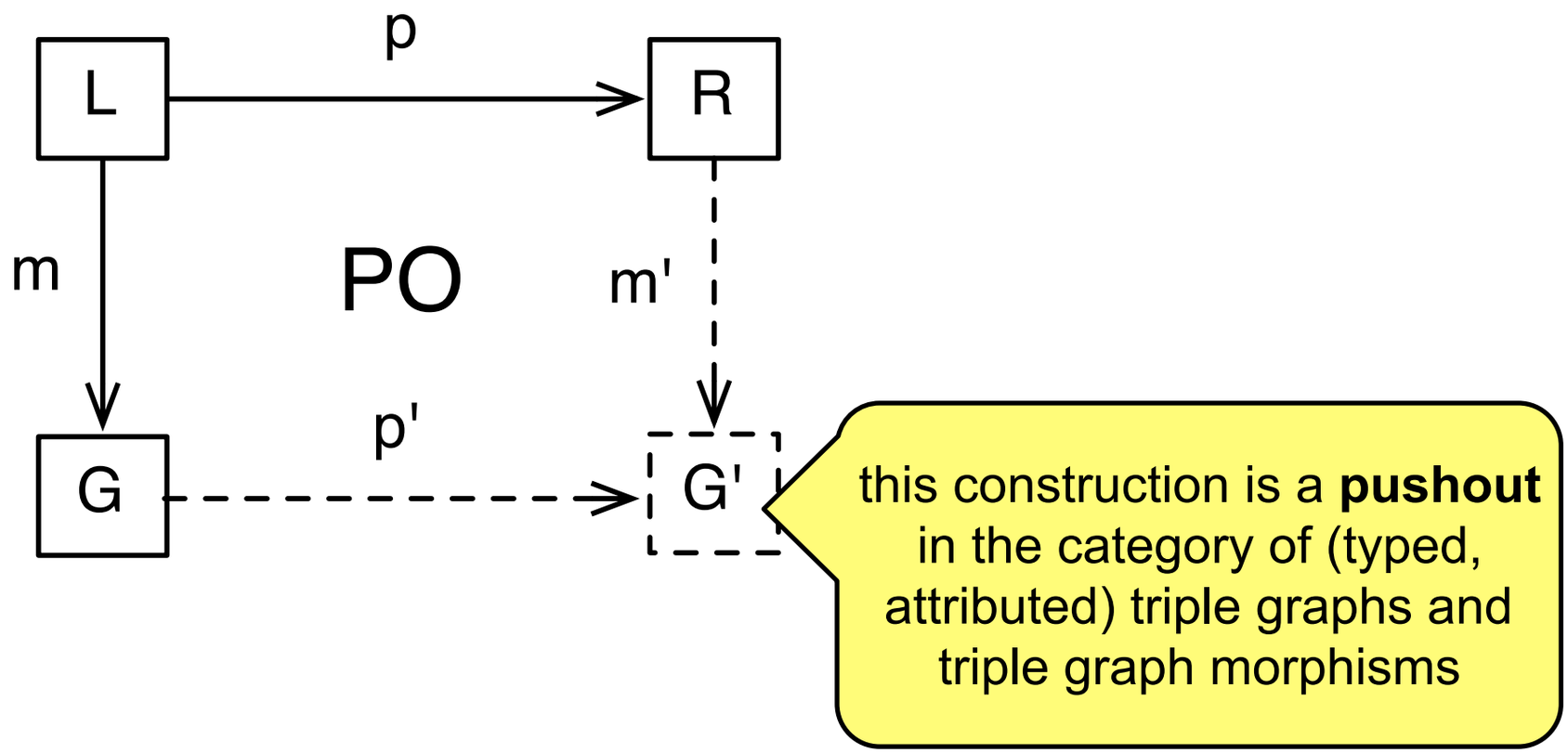
described creating delta




host graph

# Concrete deltas are derived via **rule application**

Nodes are triple graphs, arrows are triple graph morphisms




this construction is a **pushout** in the category of (typed, attributed) triple graphs and triple graph morphisms

**all you need to know:**

rule application is formal, always possible, and is unique up to isomorphisms

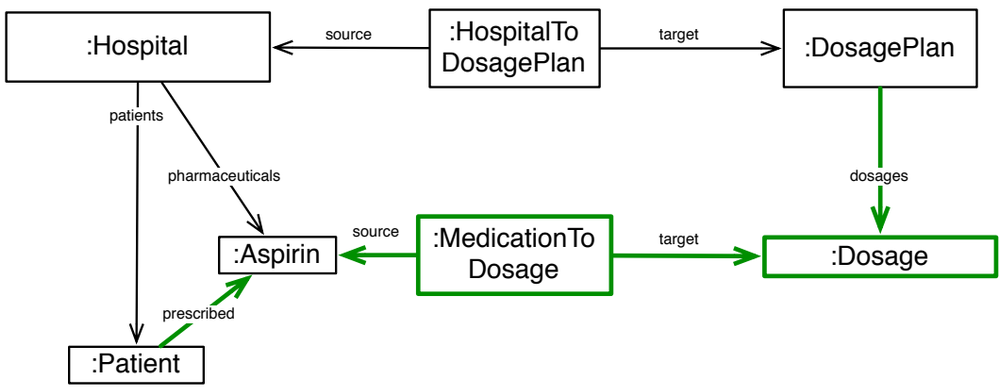
# **Implicit** ignore rules

**Idea 5:**

derive some “boring” rules by convention, i.e.,
assume they are specified implicitly

p1:


p2:




for every element in the metamodel, that is not created by any rule

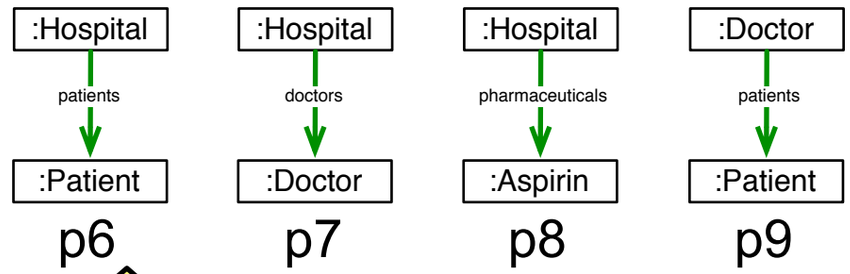
+

derive a minimal rule to create every object

p3: :Patient

p4: :Doctor

p5: :Aspirin




and a minimal rule to create every link

these rules are called **ignore** rules as they are only in one domain

# From Triple Graph Grammars to Lenses

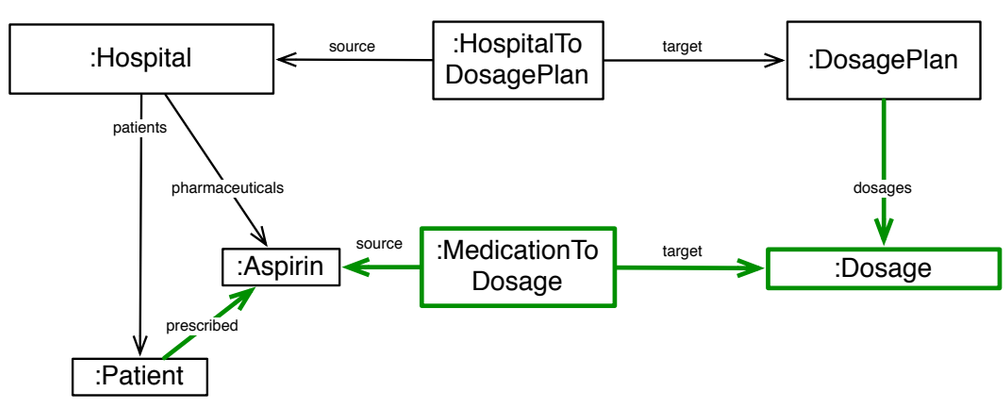
MediSoft ⟷ (R) MediSupply

user supplies a **triple space** (via a triple of metamodels)

+

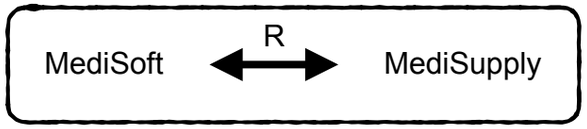
p1:


p2:

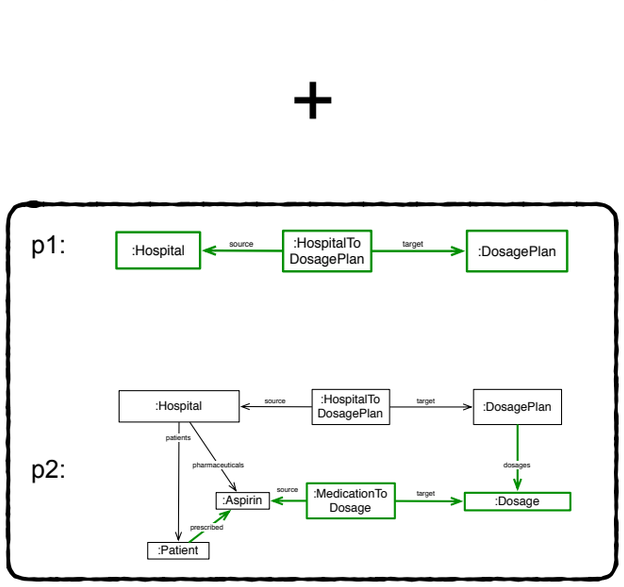



… and a finite set of monotonic, simultaneous triple rules, i.e., a **triple graph grammar**

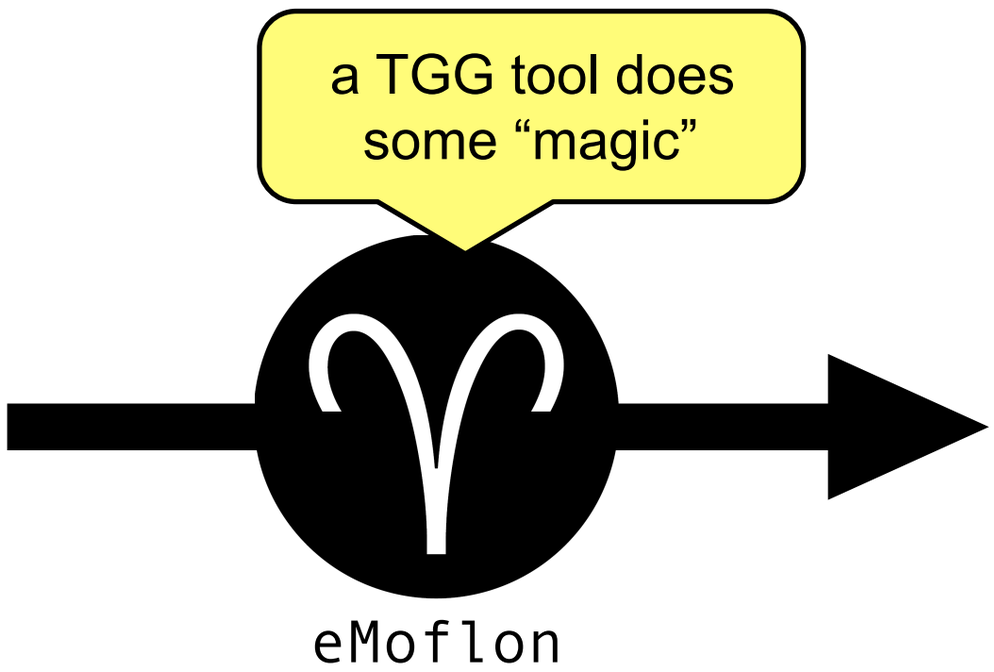
# From Triple Graph Grammars to Lenses

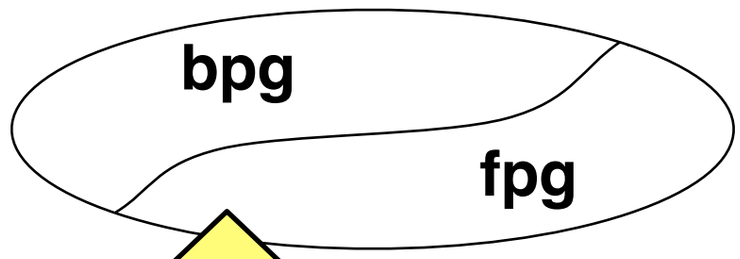



+




a TGG tool does some “magic”




and produces a symmetric delta lens!

**coming up:** we’ll take a closer look at the “magic” involved and some formal properties used to control it

# Task 2: Your first Triple Graph Grammar

1. Install VirtualBox from www.virtualbox.org
2. Download this VM: https://db.tt/gYgQMShZ
3. Open VM, start Eclipse with shortcut on desktop
4. Choose `workspaces/task2` as your workspace
5. Follow instructions from https://db.tt/g9uGurIs