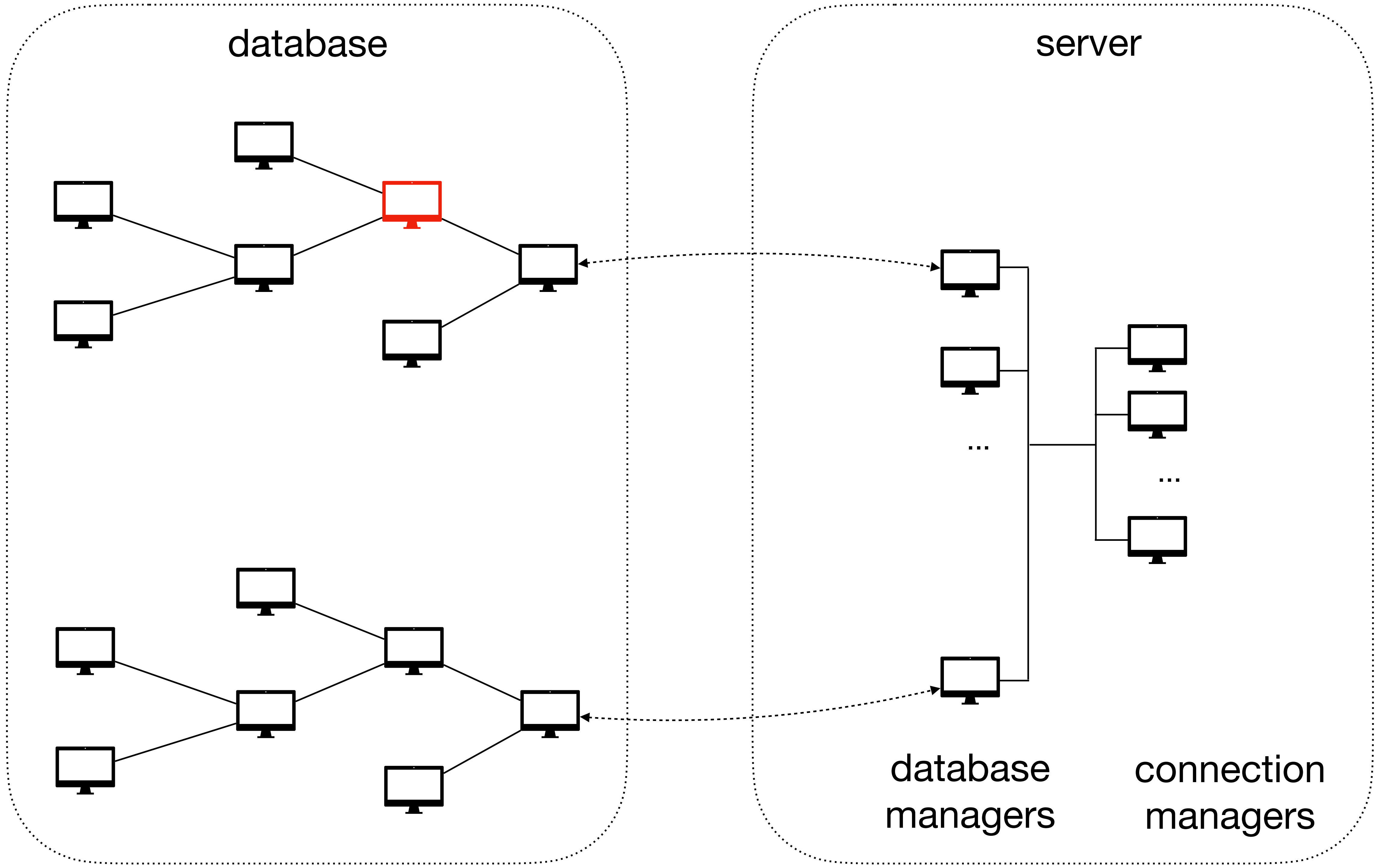
Self-Adapting Networks

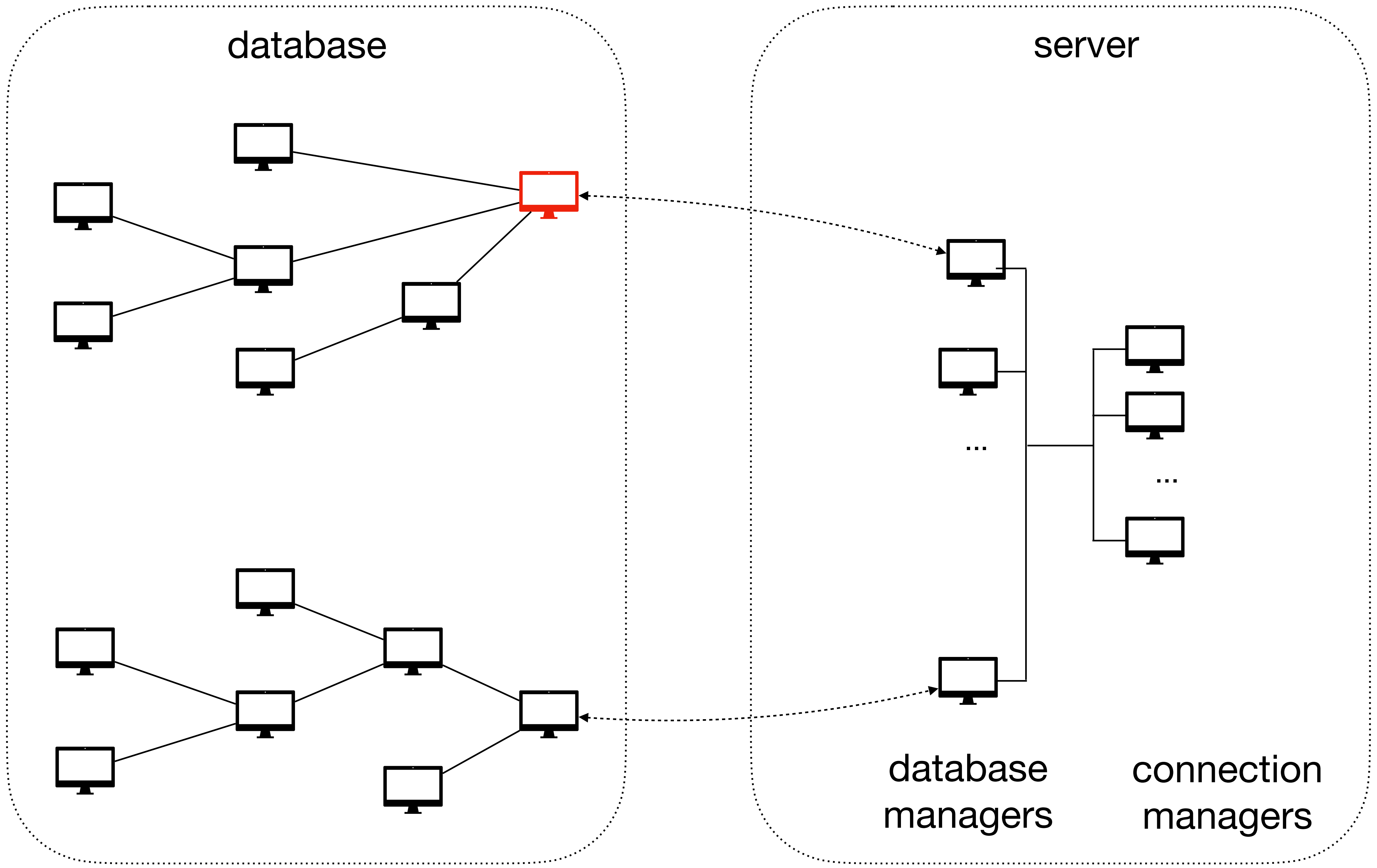
Radu Iosif (CNRS, University of Grenoble, VERIMAG)
joint work with Marius Bozga, Lucas Bueri (VERIMAG),
Joost-Pieter Katoen, Emma Ahrens (RWTH Aachen) and
Florian Zuleger (TU Wien)

Architectures and Reconfiguration



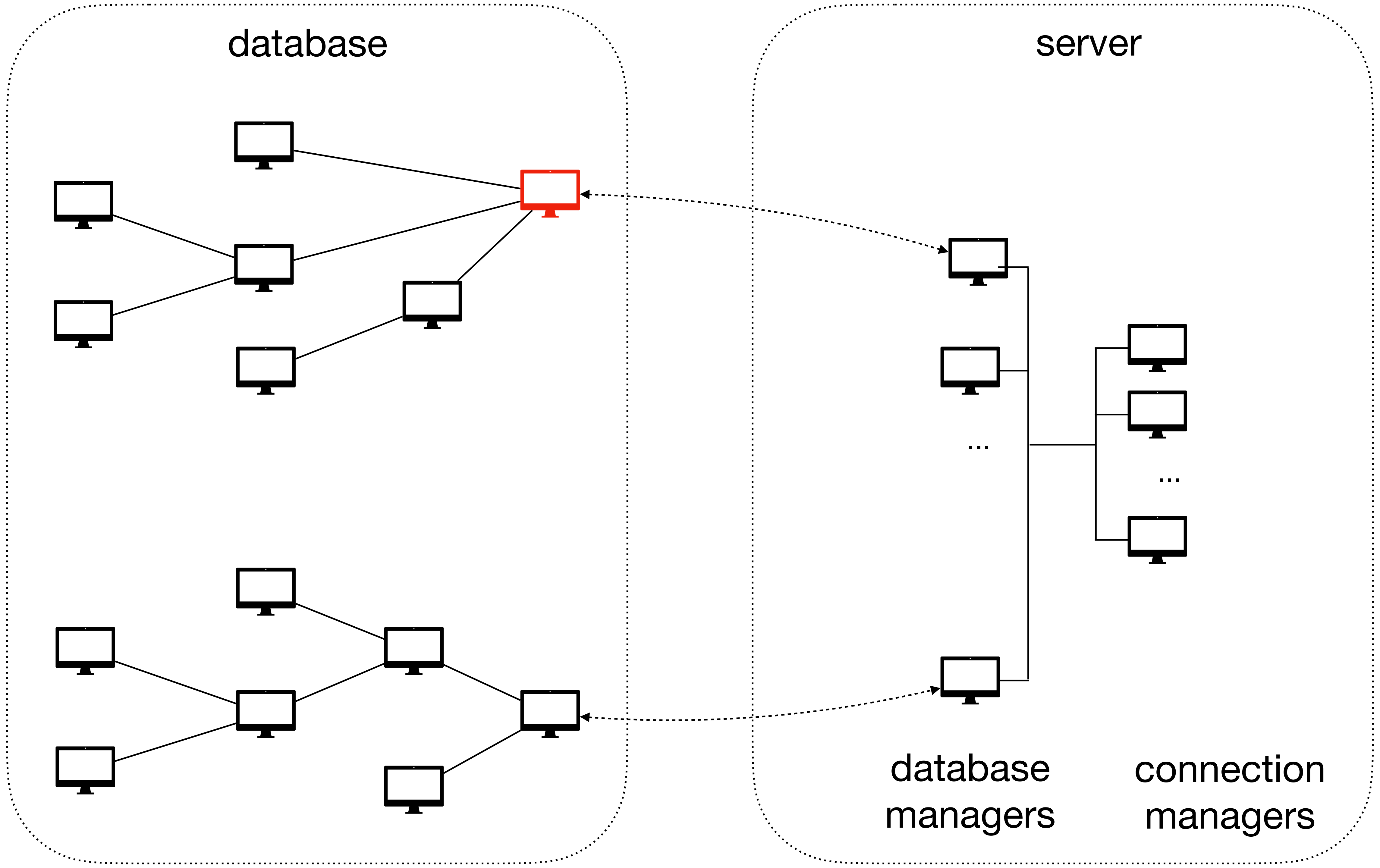
Architectural styles
(pipeline, tree, star, clique, etc.)

Architectures and Reconfiguration



Internal reconfiguration
(self-adapting networks)

Architectures and Reconfiguration



Internal reconfiguration
(self-adapting networks)

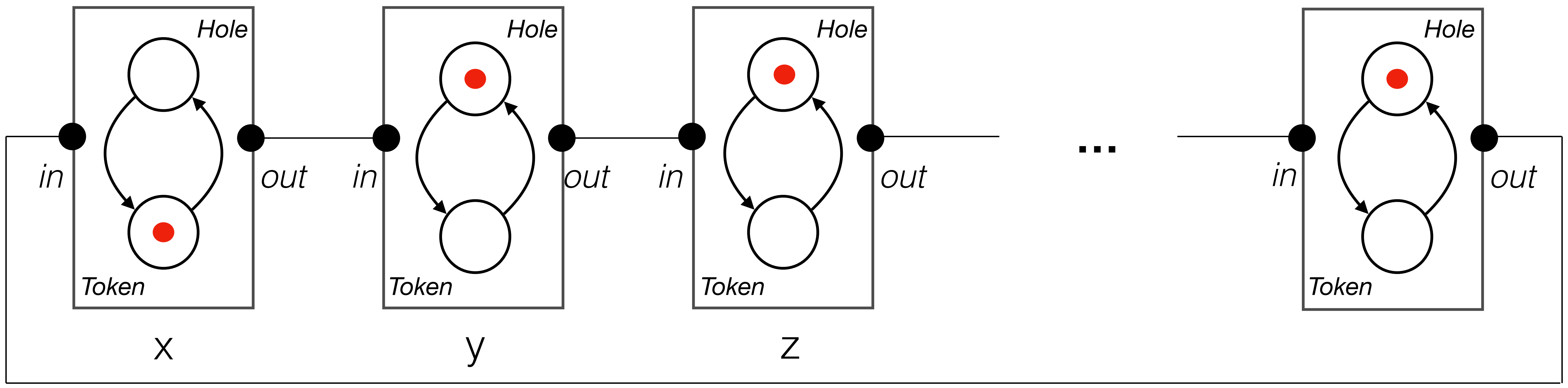
Internal vs external initiation of architectural changes

- self-adapting systems have internal initiation (guards)

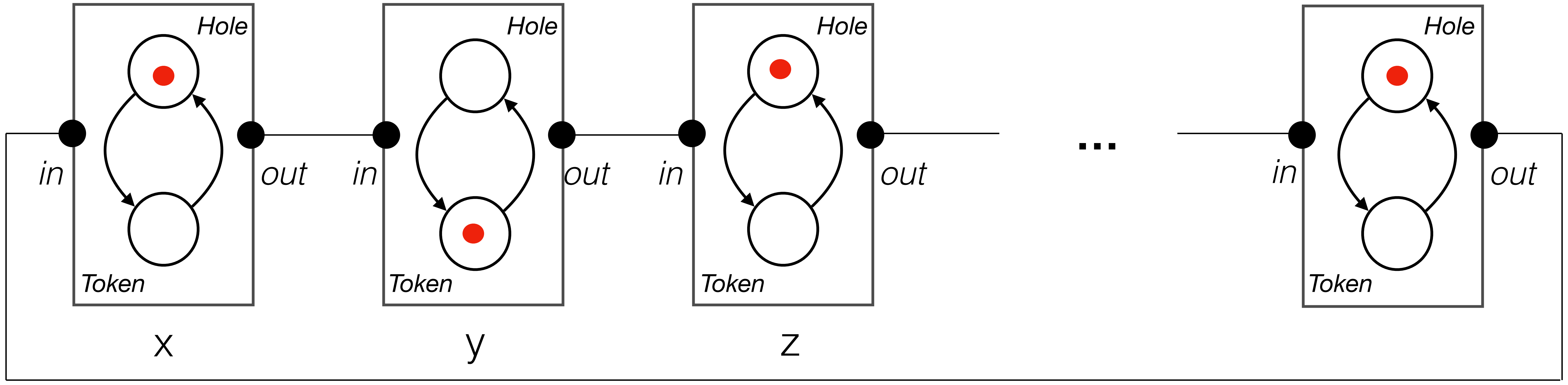
Centralized vs distributed management

- centralized (sequential) management: simpler to implement and supported by the majority of dynamic reconfiguration languages
- distributed (parallel) management: efficient and realistic but more challenging to model and reason about

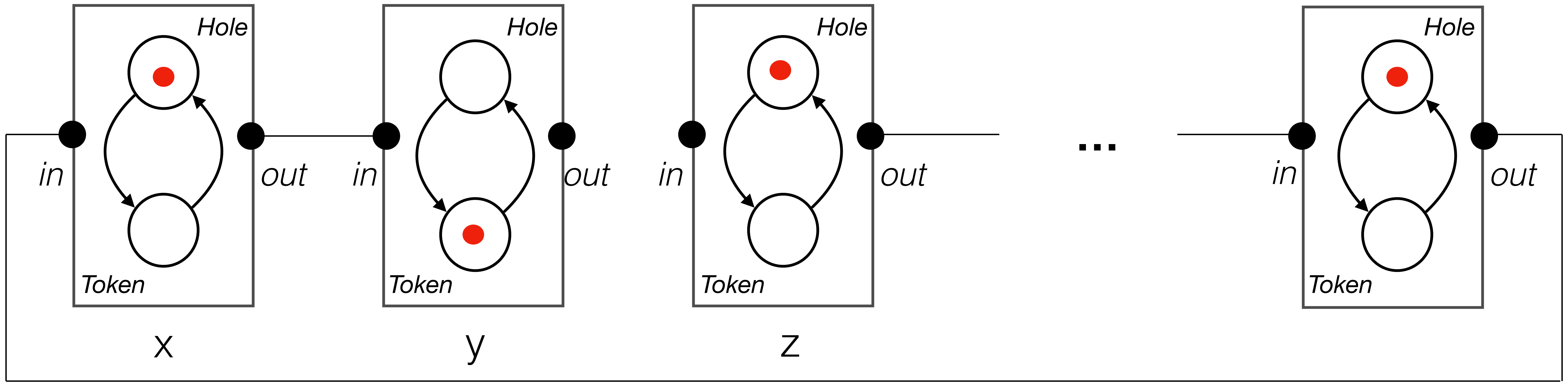
What can possibly go wrong?



What can possibly go wrong?



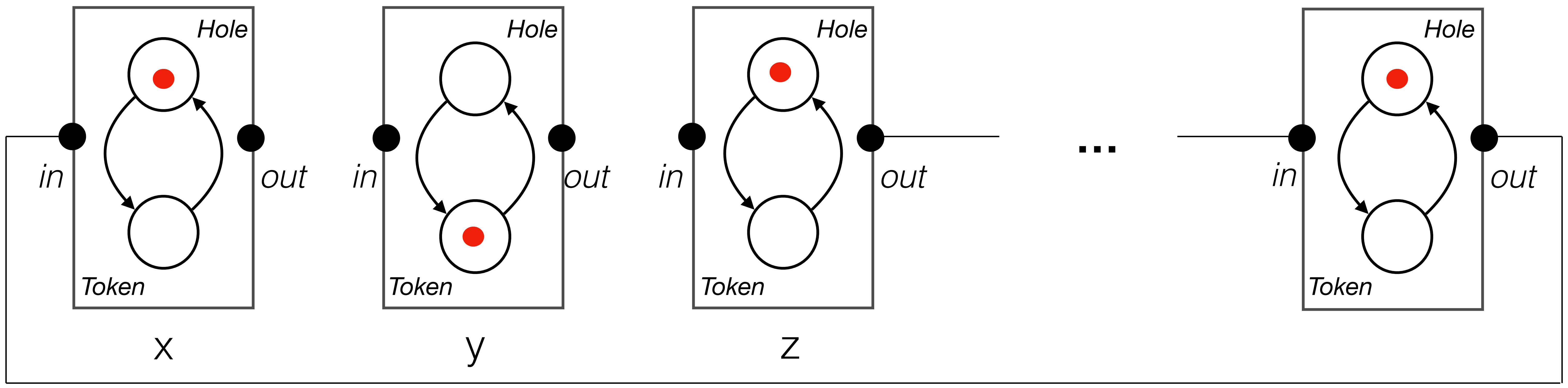
What can possibly go wrong?



reconfiguration program {

```
disconnect(y.out, z.in);
```

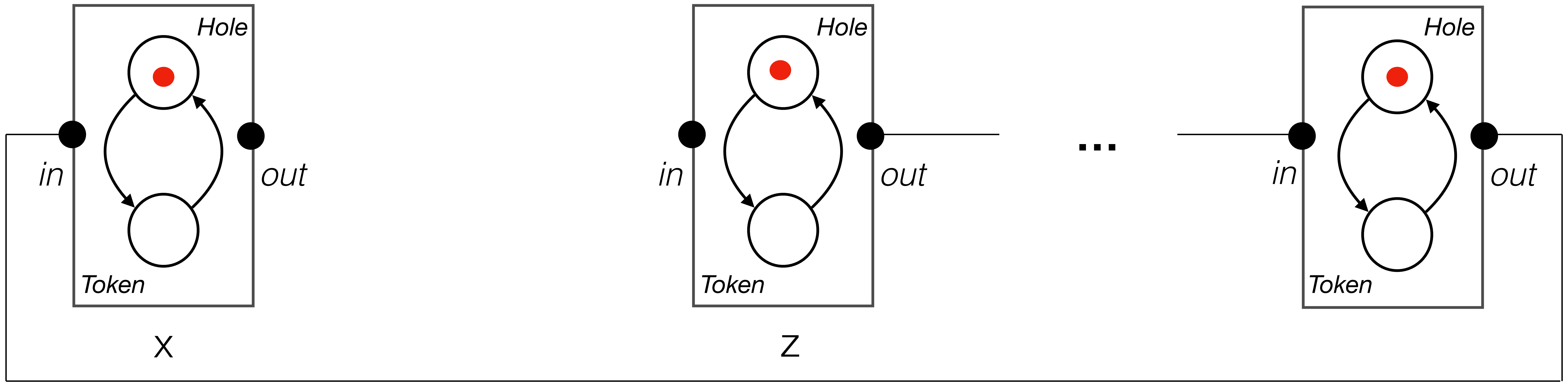
What can possibly go wrong?



reconfiguration program

```
disconnect(y.out, z.in);
disconnect(x.out, y.in);
```

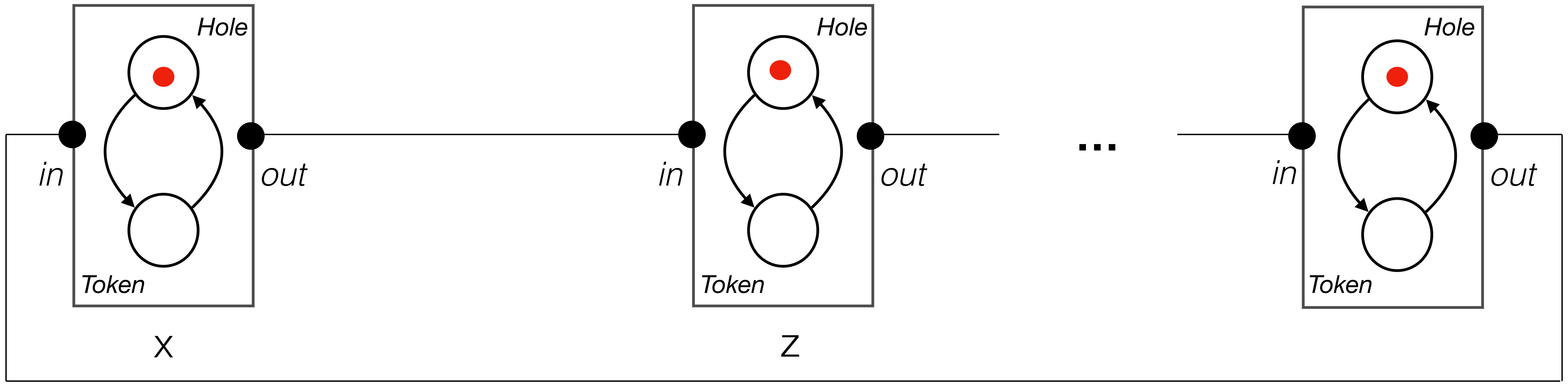
What can possibly go wrong?



reconfiguration program {

```
disconnect(y.out, z.in);
disconnect(x.out, y.in);
delete(y);
```

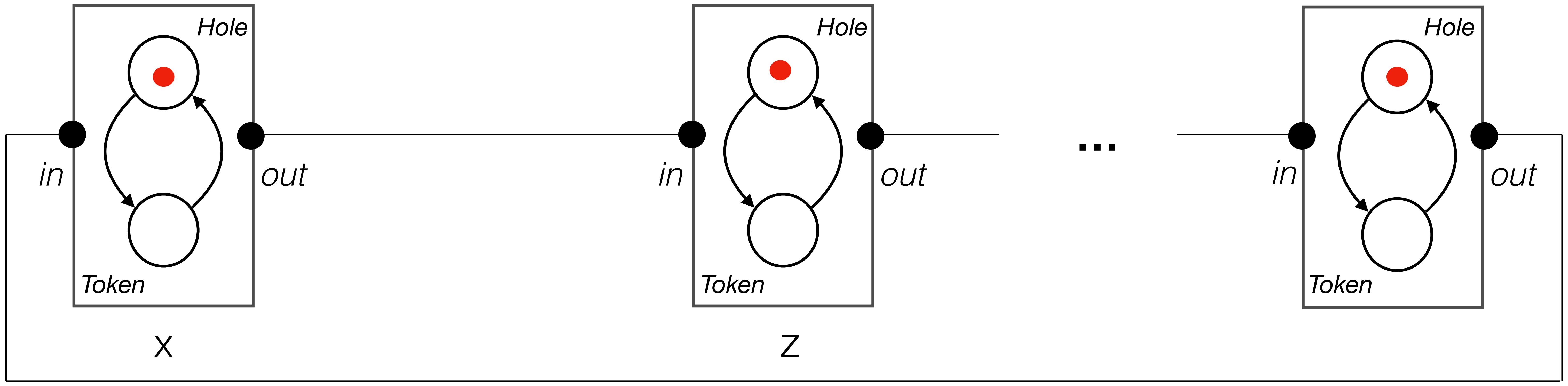
What can possibly go wrong?



reconfiguration program {

```
disconnect(y.out, z.in);
disconnect(x.out, y.in);
delete(y);
connect(x.out, z.in);
```

What can possibly go wrong?

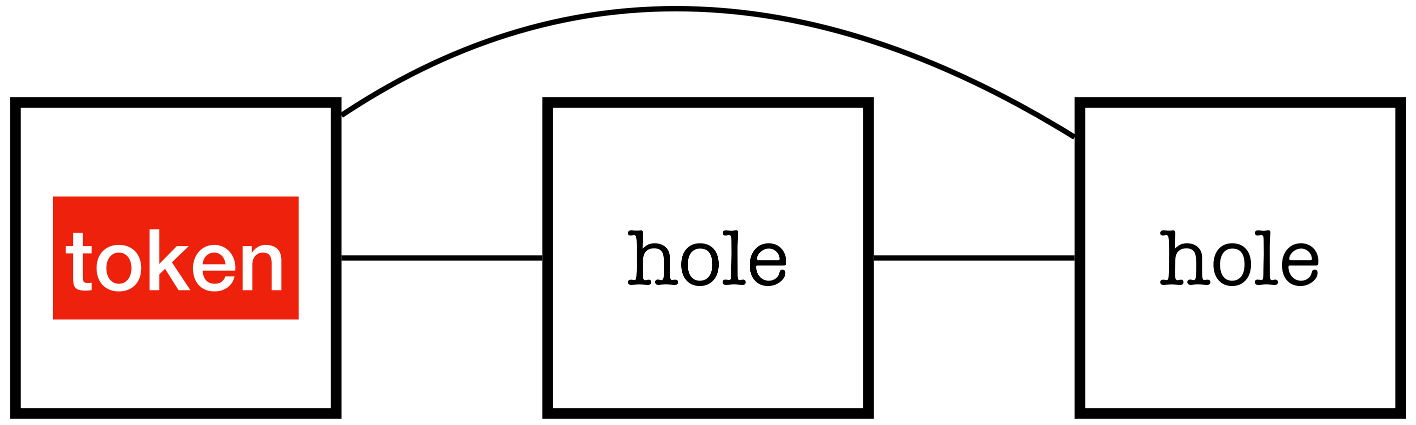


reconfiguration program

```
disconnect(y.out, z.in);
disconnect(x.out, y.in);
delete(y);
connect(x.out, z.in);
```

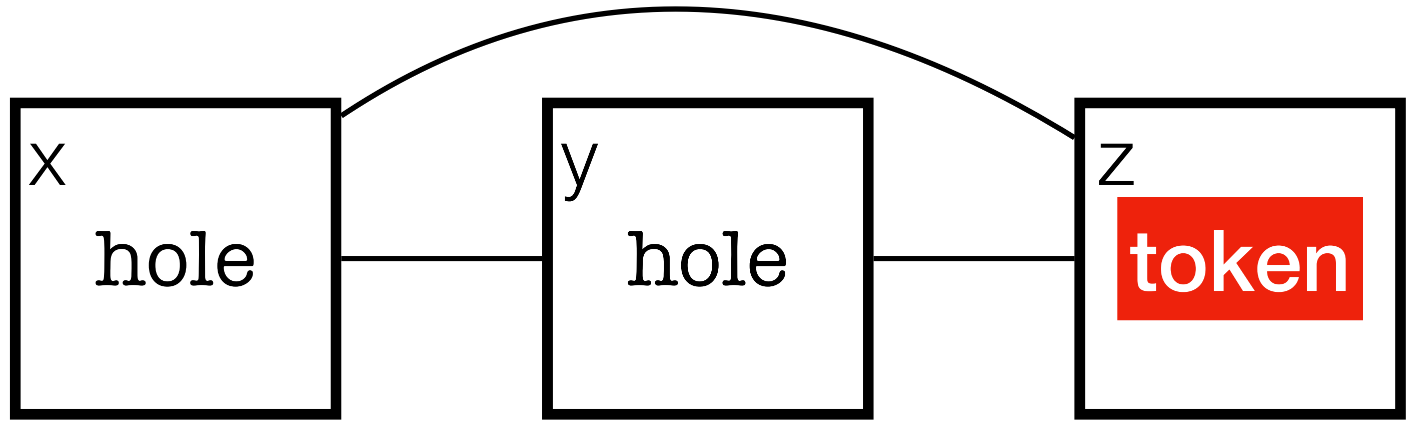
→ deadlock

Network Configurations

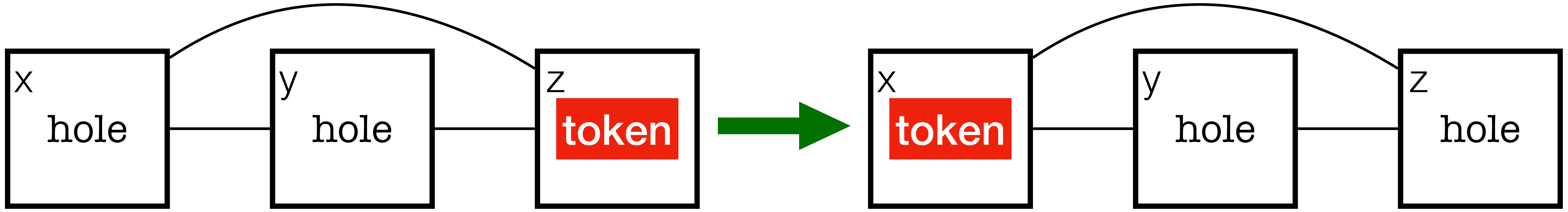


A configuration is a network with a snapshot of the states of each component

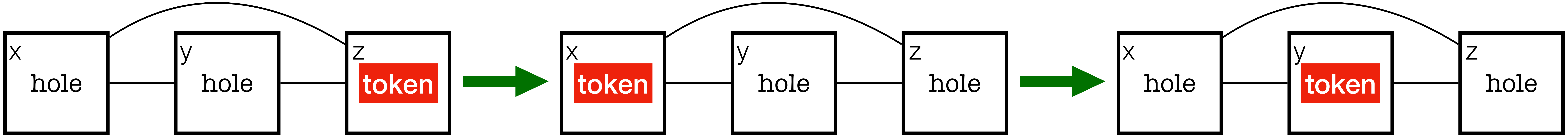
Havoc vs Reconfiguration Actions



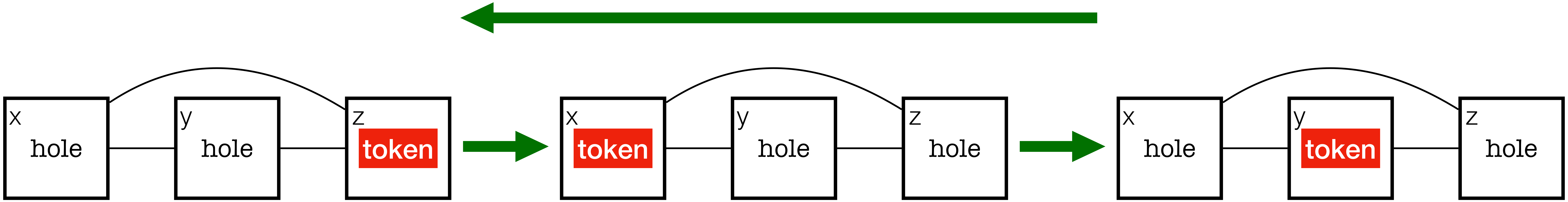
Havoc vs Reconfiguration Actions



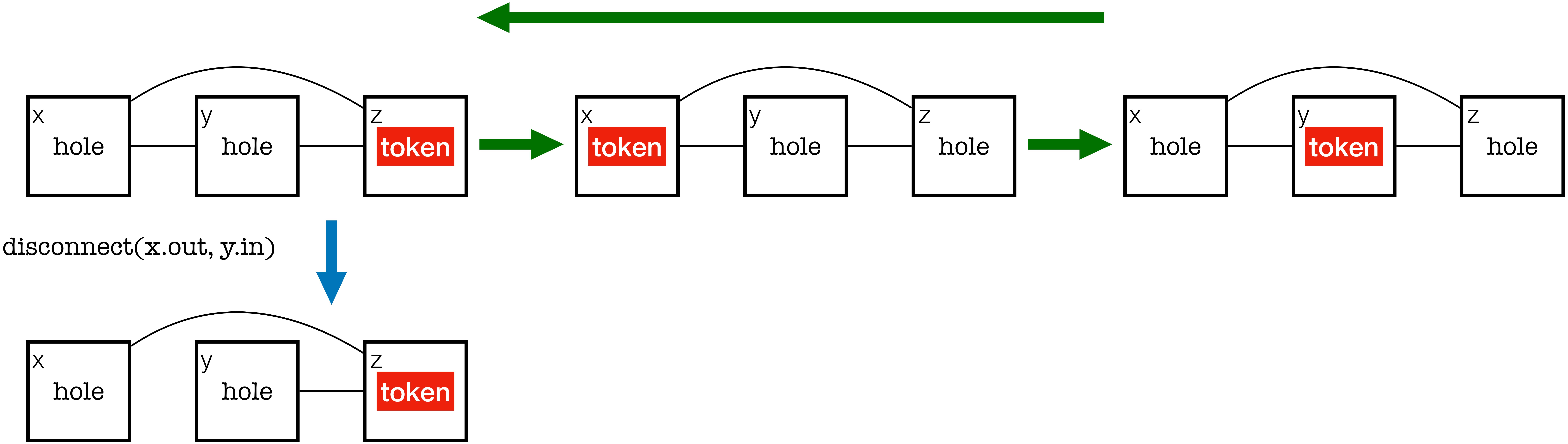
Havoc vs Reconfiguration Actions



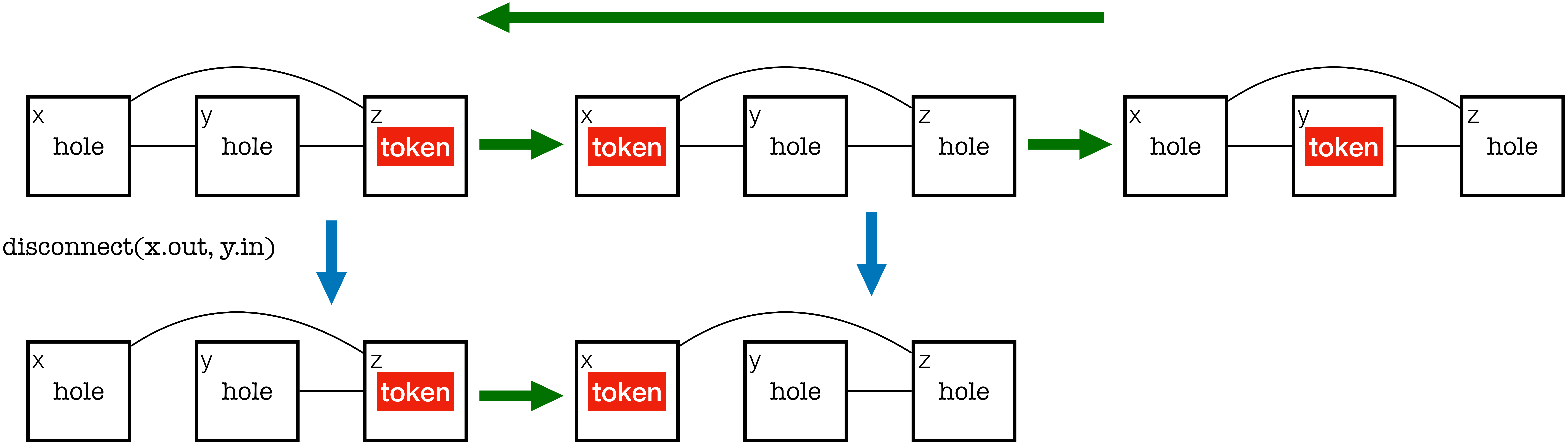
Havoc vs Reconfiguration Actions



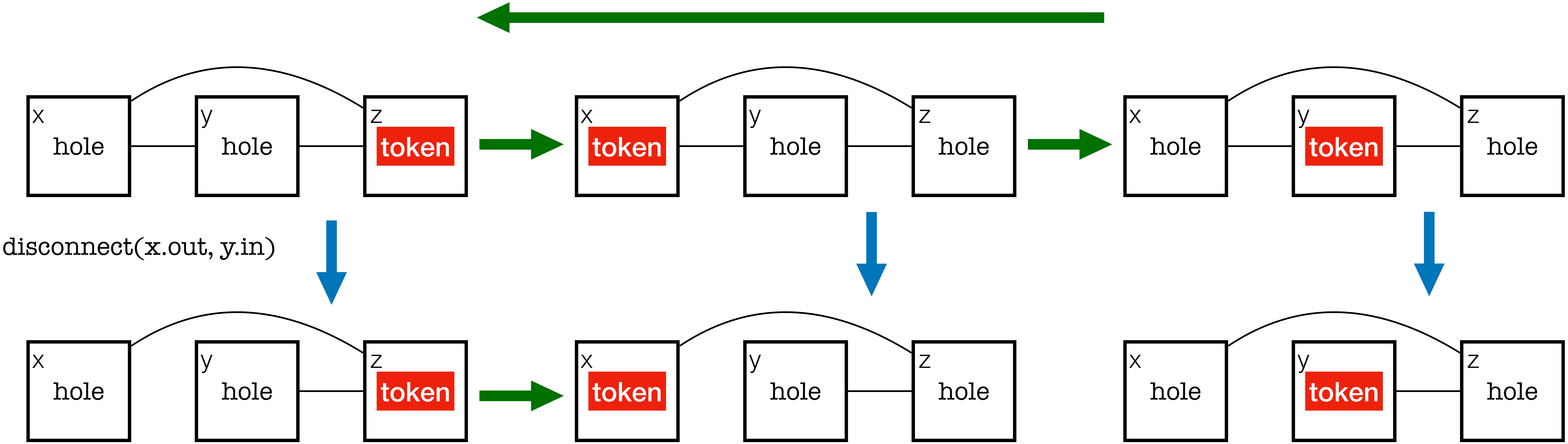
Havoc vs Reconfiguration Actions



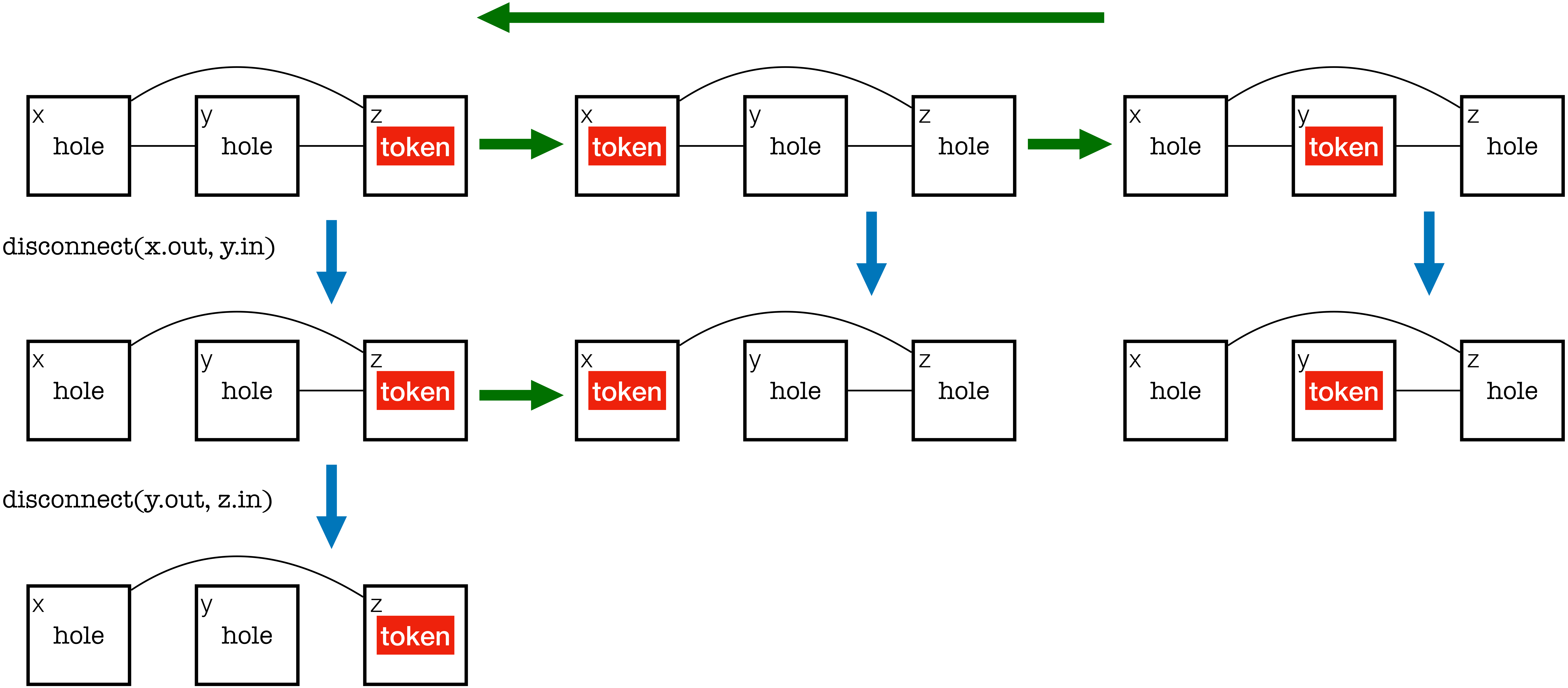
Havoc vs Reconfiguration Actions



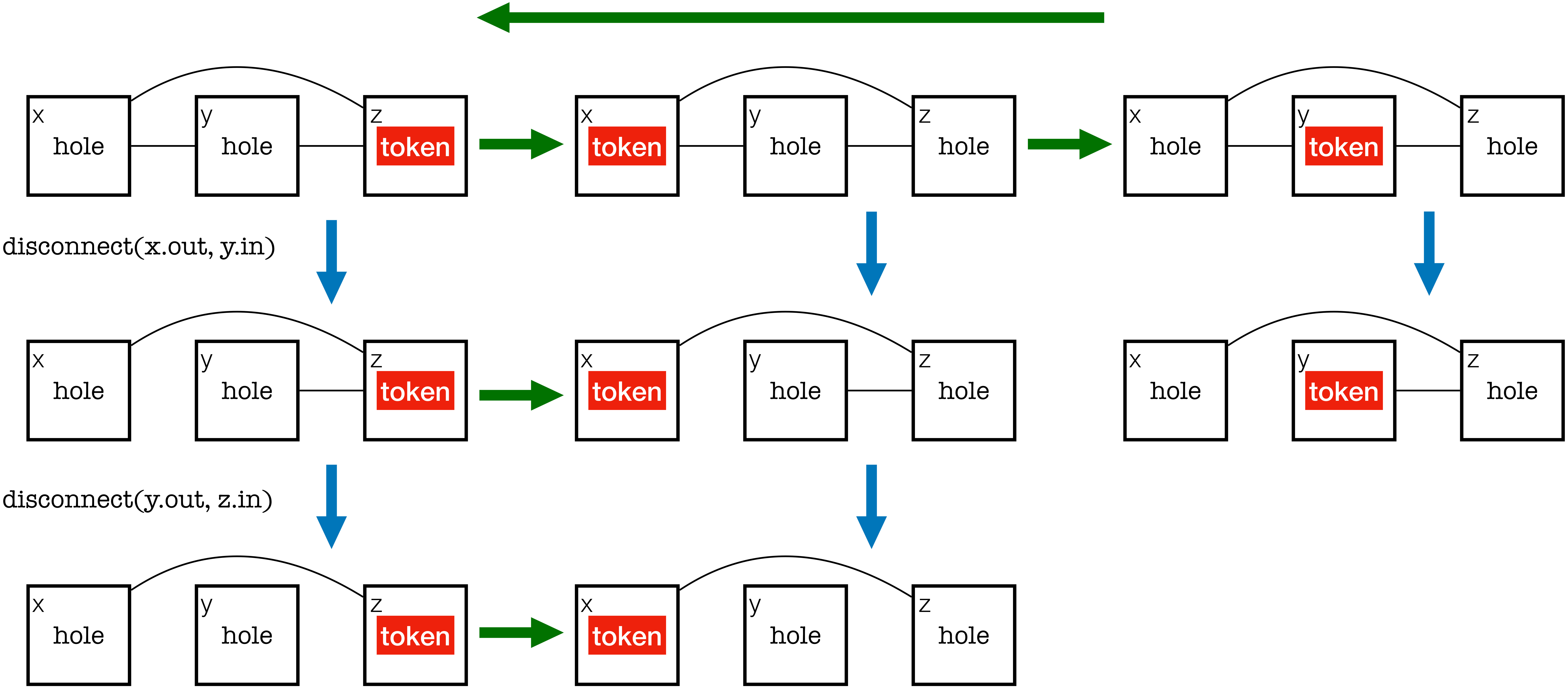
Havoc vs Reconfiguration Actions



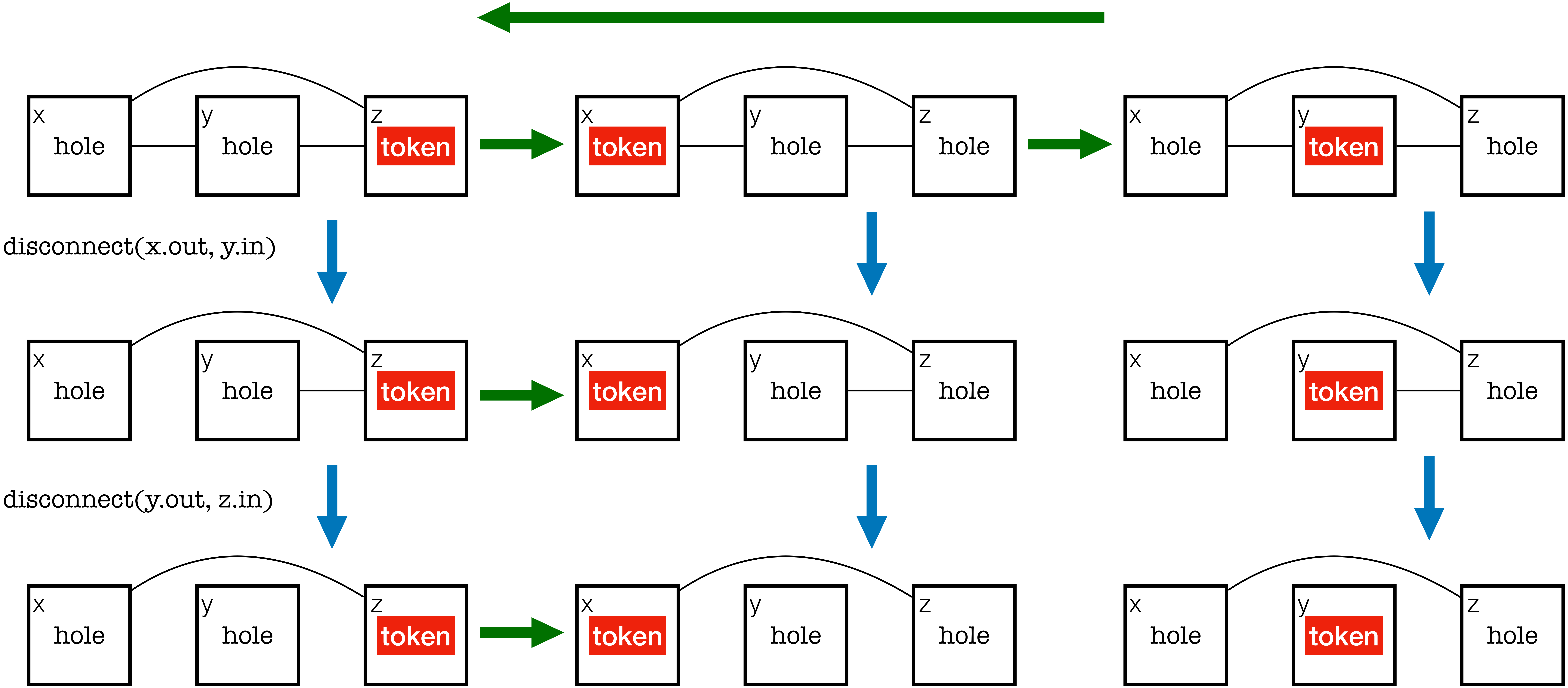
Havoc vs Reconfiguration Actions



Havoc vs Reconfiguration Actions



Havoc vs Reconfiguration Actions



Self-Adapting Networks are Infinite-state Systems

- Transition systems with unbounded number of configurations:
  - new components can be added, yielding increasingly complex reachability graphs

Self-Adapting Networks are Infinite-state Systems

- Transition systems with unbounded number of configurations:
  - new components can be added, yielding increasingly complex reachability graphs

- Two orthogonal types of actions that interleave:
  - reconfiguration actions change the architecture of a system
  - havoc actions are state changes caused by firing interactions

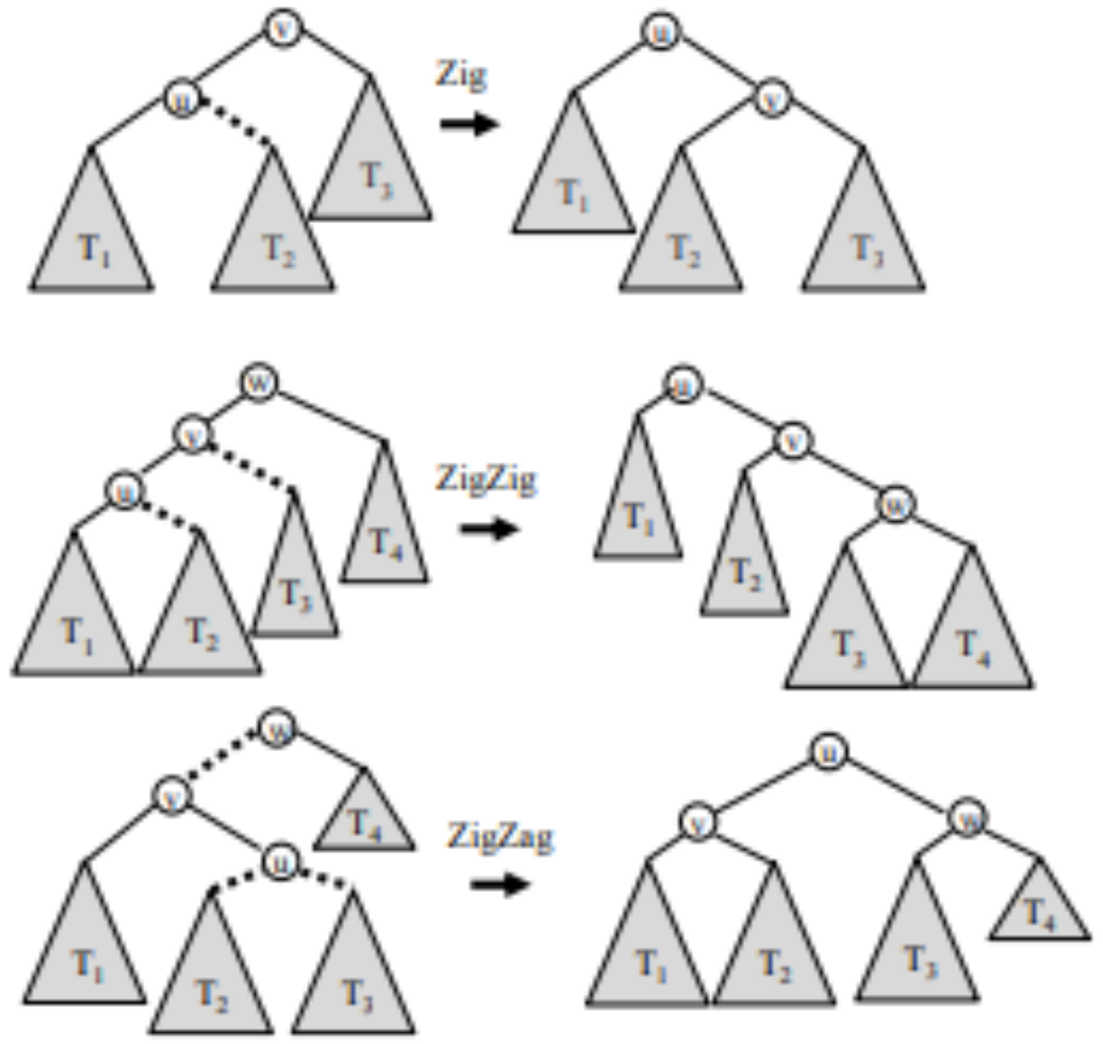
Self-Adapting Networks are Infinite-state Systems

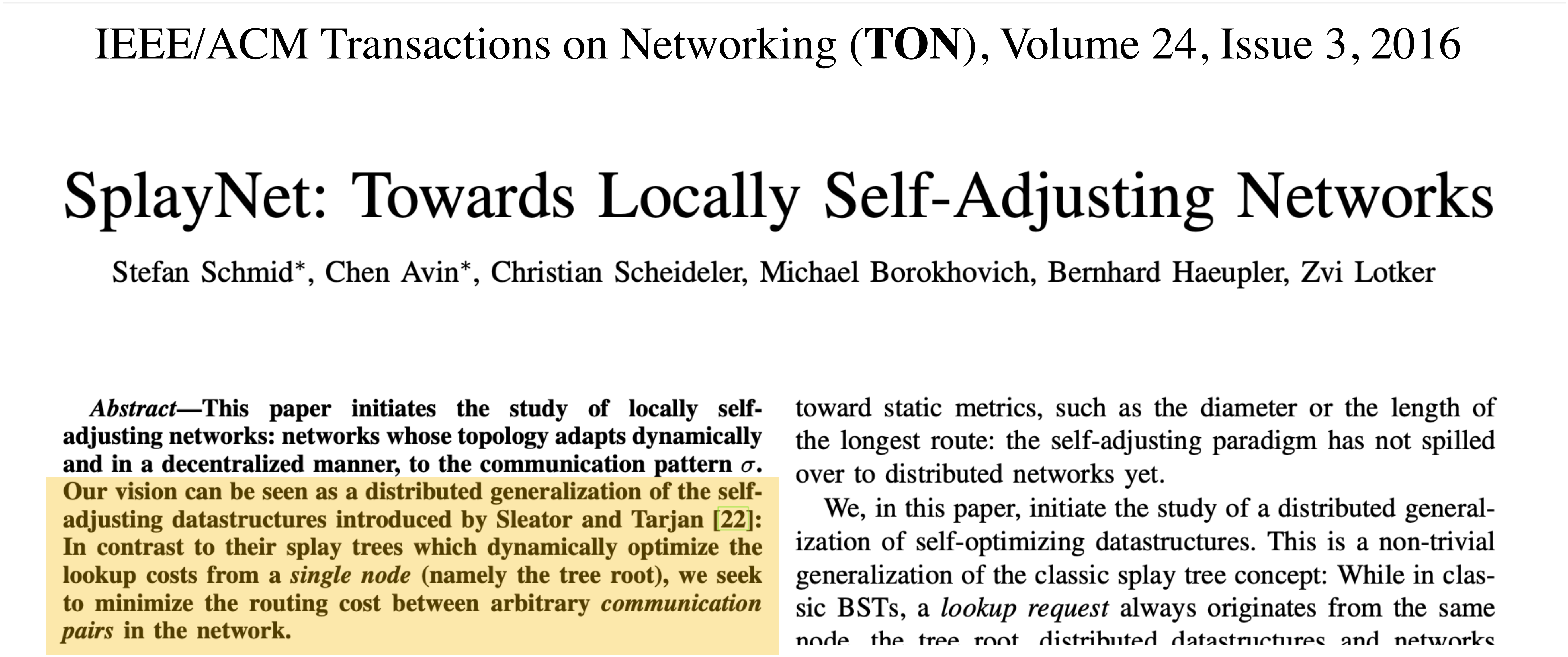
- Transition systems with unbounded number of configurations:
  - new components can be added, yielding increasingly complex reachability graphs

- Two orthogonal types of actions that interleave:
  - reconfiguration actions change the architecture of a system
  - havoc actions are state changes caused by firing interactions

- The correctness proofs combine:
  - reconfiguration rules using local reasoning scale up via compositionality [Ahrens, Bozga, I, Katoen, OOPSLA'22]
  - havoc invariants using regular model checking techniques [Bozga, Bueri, I, CONCUR'22]
  - proving safety of assertions using parametric model checking techniques [Bozga, I, Sifakis, TCS' 23]

Architectures and Datastructures





IEEE/ACM Transactions on Networking (**TON**), Volume 24, Issue 3, 2016

SplayNet: Towards Locally Self-Adjusting Networks

Stefan Schmid*, Chen Avin*, Christian Scheideler, Michael Borokhovich, Bernhard Haeupler, Zvi Lotker

***Abstract*—This paper initiates the study of locally self-adjusting networks: networks whose topology adapts dynamically and in a decentralized manner, to the communication pattern σ. Our vision can be seen as a distributed generalization of the self-adjusting datastructures introduced by Sleator and Tarjan [22]: In contrast to their splay trees which dynamically optimize the lookup costs from a *single node* (namely the tree root), we seek to minimize the routing cost between arbitrary *communication pairs* in the network.**

toward static metrics, such as the diameter or the length of the longest route: the self-adjusting paradigm has not spilled over to distributed networks yet.

We, in this paper, initiate the study of a distributed generalization of self-optimizing datastructures. This is a non-trivial generalization of the classic splay tree concept: While in classic BSTs, a *lookup request* always originates from the same node, the tree root, distributed datastructures and networks

- Network architectures are similar to the datastructures used in programming
- Used to design efficient routing algorithms that minimize internal traffic in datacenters
- We aim at proving correctness of self-adapting networks using a Configuration Logic (CL)

A Logic of Configurations (CL)

emp	the empty network

A Logic of Configurations (CL)

emp	the empty network
[x]@q	a single node in state q and no interactions

A Logic of Configurations (CL)

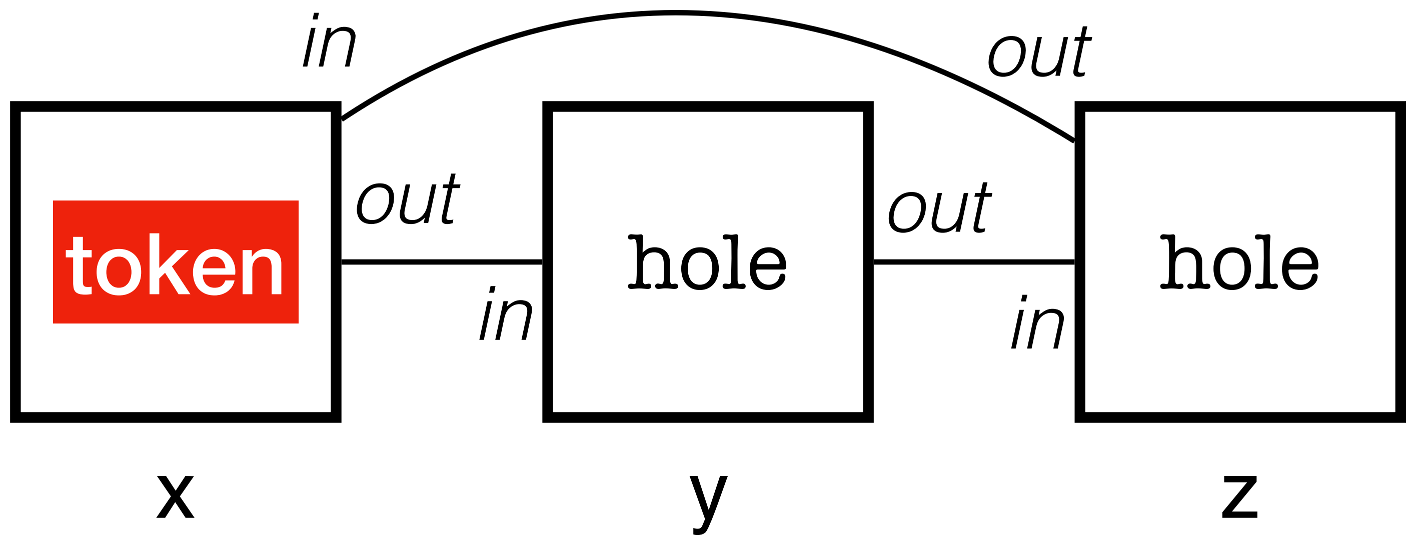
emp	the empty network
[x]@q	a single node in state q and no interactions
$\langle x_1.p_1$ ...., $x_n.p_n\rangle$	a single interaction and no nodes

A Logic of Configurations (CL)

emp	the empty network
[x]@q	a single node in state q and no interactions
$\langle x_1.p_1 \ldots., x_n.p_n \rangle$	a single interaction and no nodes
$\phi_1 * \phi_2$	union of disjoint networks

A Logic of Configurations (CL)

emp	the empty network
$[x]@q$	a single node in state q and no interactions
$\langle x_1.p_1 \ldots., x_n.p_n \rangle$	a single interaction and no nodes
$\phi_1 * \phi_2$	union of disjoint networks



$[x]@token * \langle x.out,y.in\rangle * [y]@hole * \langle y.out,z.in\rangle * [z]@hole * \langle z.out, x.in\rangle$

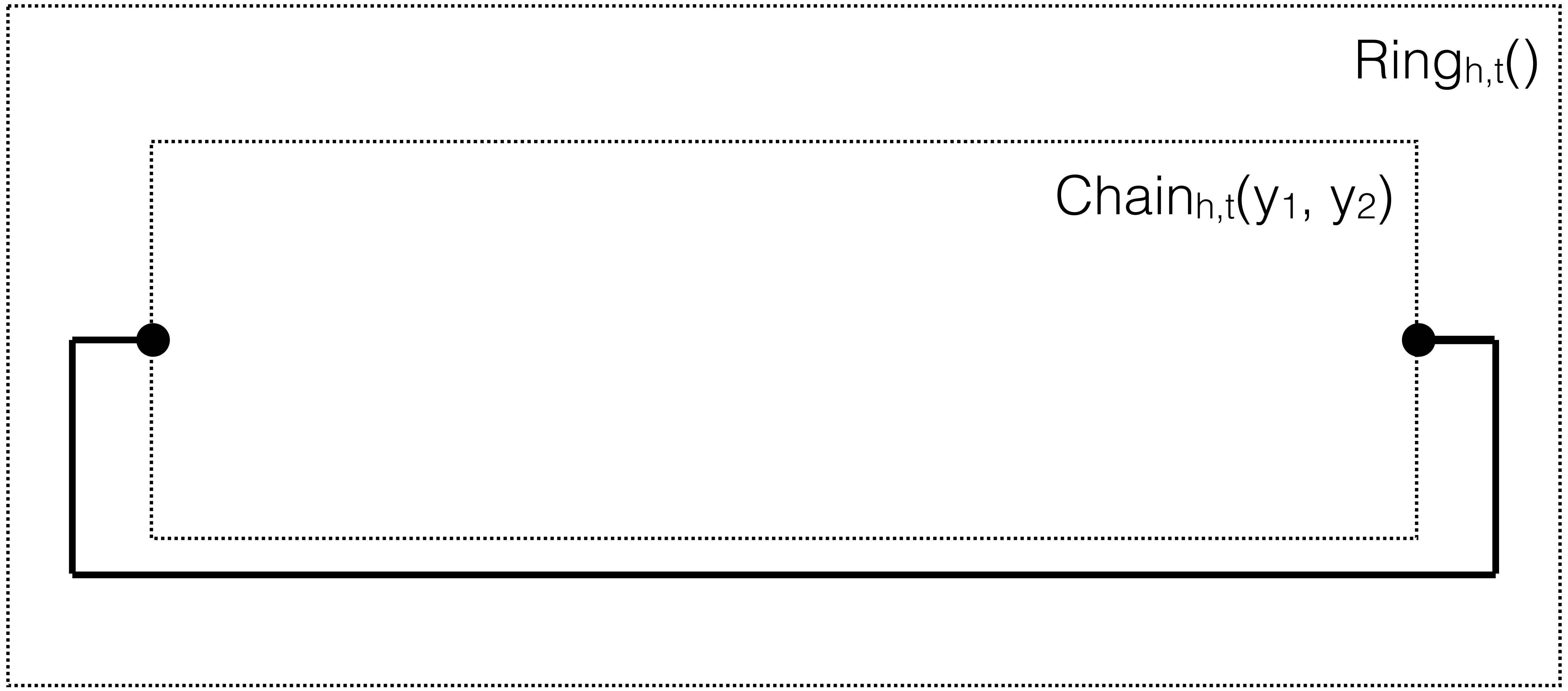
A Logic of Configurations (CL)

emp	the empty network
$[x]@q$	a single node in state q and no interactions
$\langle x_1.p_1 \text{ ...., } x_n.p_n\rangle$	a single interaction and no nodes
$\phi_1 * \phi_2$	separating conjunction (union of disjoint networks)
$\phi_1 \wedge \phi_2$	boolean conjunction
$\exists x \, . \, \phi$	existential quantification

Adding inductive definitions

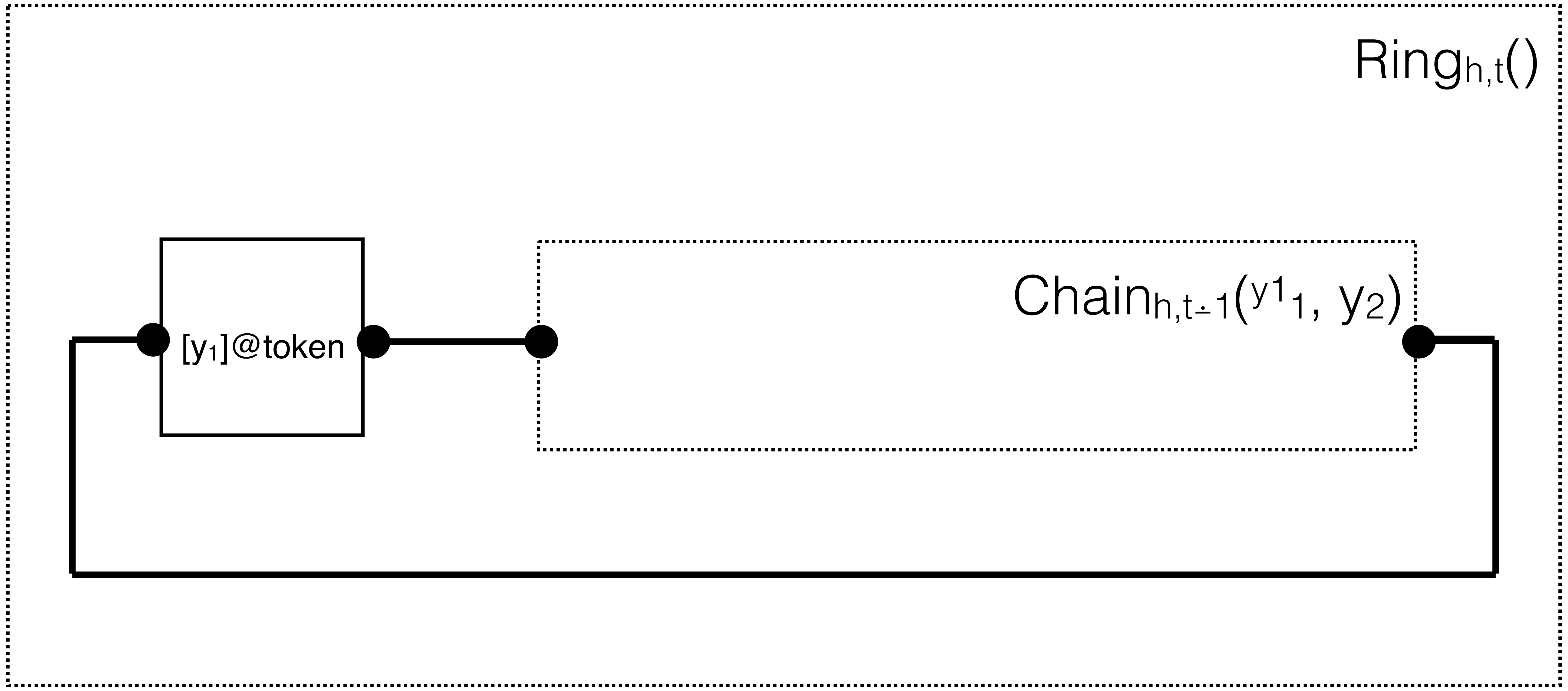
$\mathrm{Ring}_{h,t}()$

Adding inductive definitions



$Ring_{h,t}() \leftarrow \exists y_1 \exists y_2 .\ Chain_{h,t}(y_1, y_2) * \langle y_2.out, y_1.in \rangle$

Adding inductive definitions

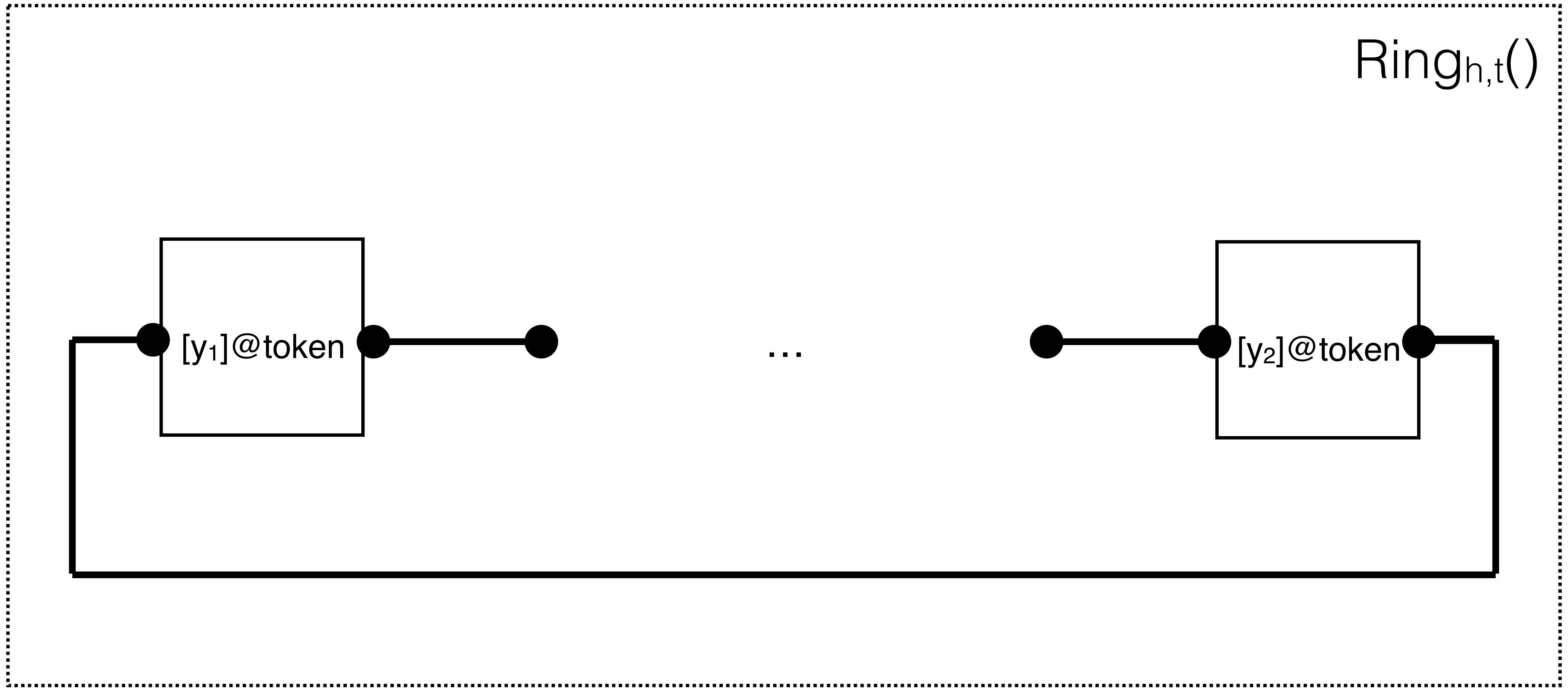


$\mathrm{Ring}_{h,t}() \leftarrow \exists y_1\ \exists y_2\ .\ \mathrm{Chain}_{h,t}(y_1, y_2) * \langle y_2.\mathrm{out}, y_1.\mathrm{in}\rangle$

$\mathrm{Chain}_{h,t}(x, y) \leftarrow \exists z\ .\ [x]@\mathrm{token} * \langle x.\mathrm{out}, z.\mathrm{in}\rangle * \mathrm{Chain}_{h,t\dot{-}1}(z, y)$

$\mathrm{Chain}_{h,t}(x, y) \leftarrow \exists z\ .\ [x]@\mathrm{hole} * \langle x.\mathrm{out}, z.\mathrm{in}\rangle * \mathrm{Chain}_{h\dot{-}1,t}(z, y),\ n\dot{-}1 \overset{\mathrm{def}}{=} \max(0,n\text{-}1)$

Adding inductive definitions



$\mathrm{Ring}_{h,t}() \leftarrow \exists y_1\, \exists y_2 .\ \mathrm{Chain}_{h,t}(y_1, y_2) * \langle y_2.\mathrm{out}, y_1.\mathrm{in}\rangle$

$\mathrm{Chain}_{h,t}(x, y) \leftarrow \exists z .\ [x]@\mathrm{token} * \langle x.\mathrm{out}, z.\mathrm{in}\rangle * \mathrm{Chain}_{h,t\dot{-}1}(z, y)$

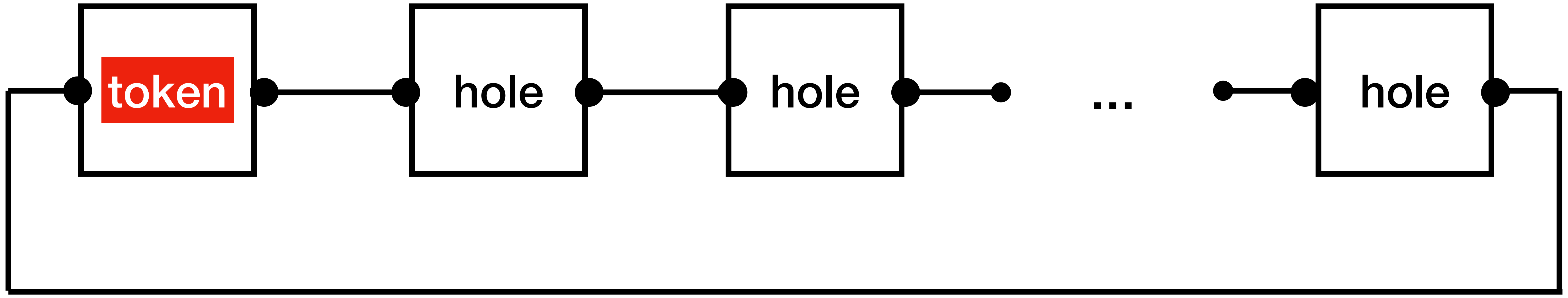
$\mathrm{Chain}_{h,t}(x, y) \leftarrow \exists z .\ [x]@\mathrm{hole} * \langle x.\mathrm{out}, z.\mathrm{in}\rangle * \mathrm{Chain}_{h\dot{-}1,t}(z, y),\ \ n\dot{-}1 \overset{\mathrm{def}}{=} \max(0,n-1)$

$\mathrm{Chain}_{0,1}(x,y) \leftarrow [x]@\mathrm{token} * x=y$ $\qquad\qquad$ $\mathrm{Chain}_{1,0}(x,y) \leftarrow [x]@\mathrm{hole} * x=y$

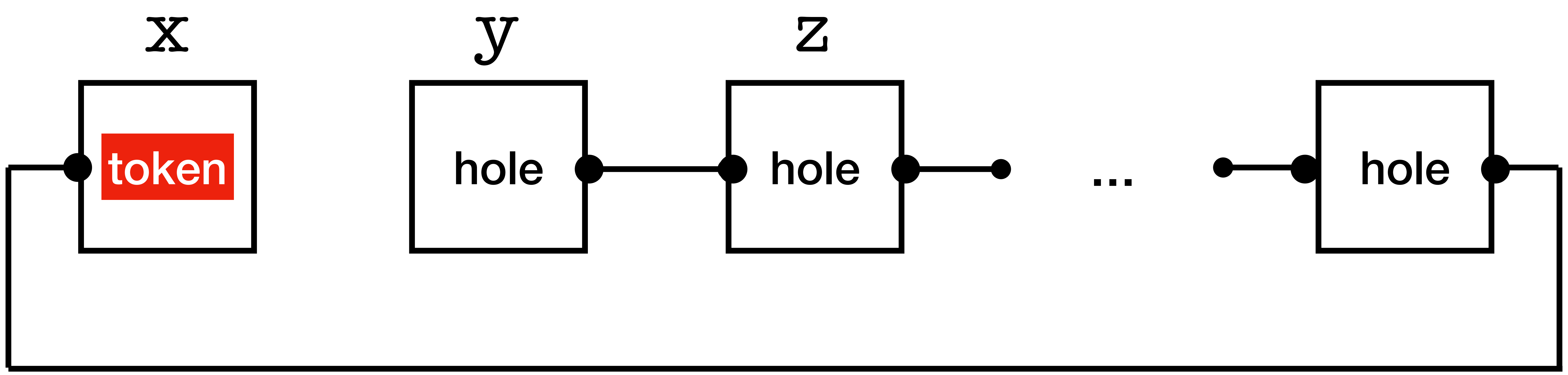
Programmed reconfigurability

- Sequential programming language based on:
  - primitives: new(x,q), delete(x), connect($x_1.p_1$, ..., $x_n.p_n$), disconnect($x_1.p_1$, ..., $x_n.p_n$)
  - conditional: with x_1, ..., x_n : ϕ do R od, where ϕ is a CL formula with no predicates
  - sequential composition (R_1; R_2), iteration (R^*) and nondeterministic choice ($R_1 + R_2$)

An example: token-ring node removal

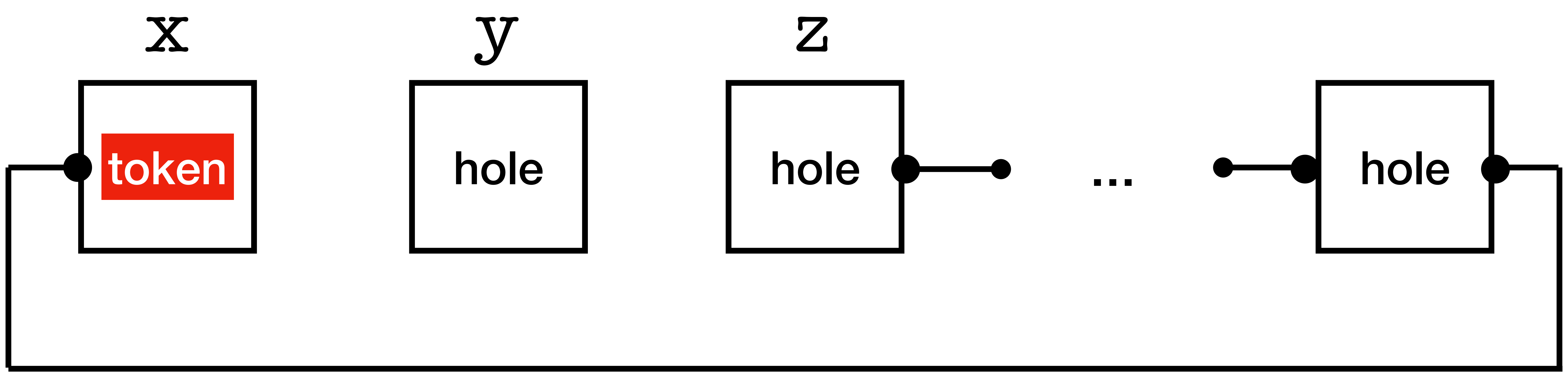


An example: token-ring node removal



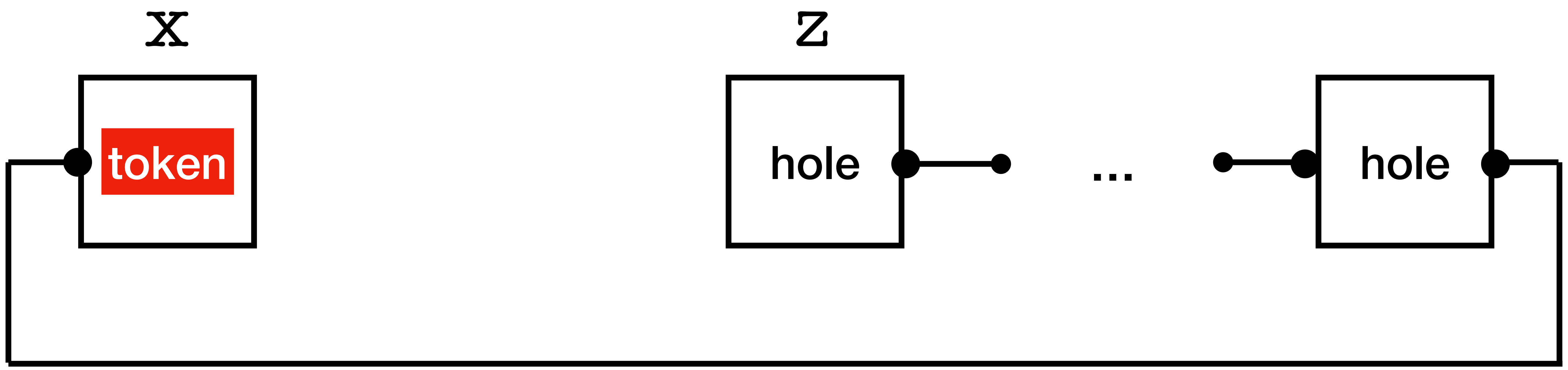
with x,y,z : ⟨x.out,y.in⟩ * [y]@hole * ⟨y.out,z.in⟩ do
  disconnect(x.out,y.in);

An example: token-ring node removal



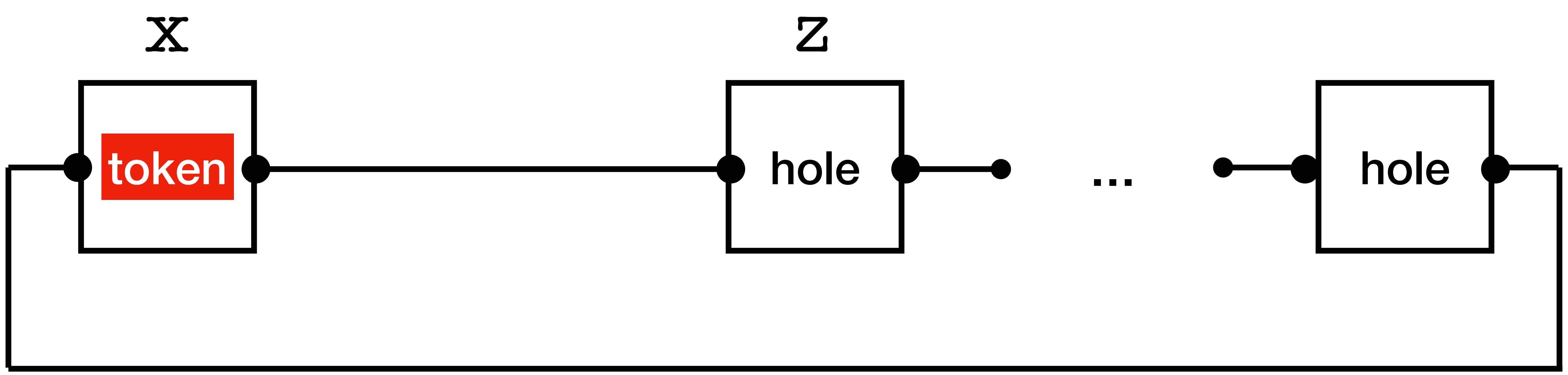
```
with x,y,z : ⟨x.out,y.in⟩ * [y]@hole * ⟨y.out,z.in⟩ do
  disconnect(x.out,y.in);
  disconnect(y.out,z.in);
```

An example: token-ring node removal



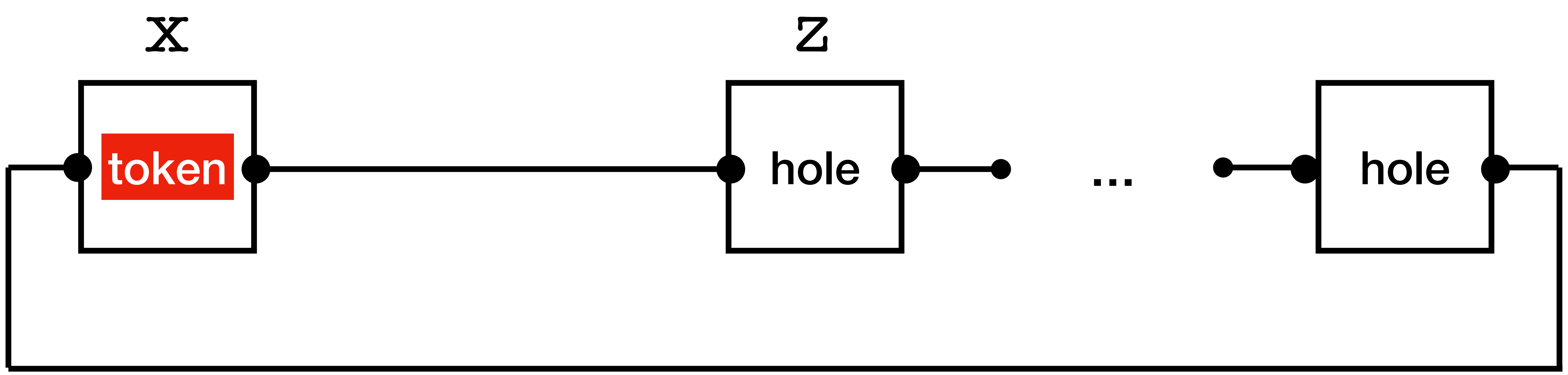
```
with x,y,z : ⟨x.out,y.in⟩ * [y]@hole* ⟨y.out,z.in⟩ do
  disconnect(x.out,y.in);
  disconnect(y.out,z.in);
  delete(y);
```

An example: token-ring node removal



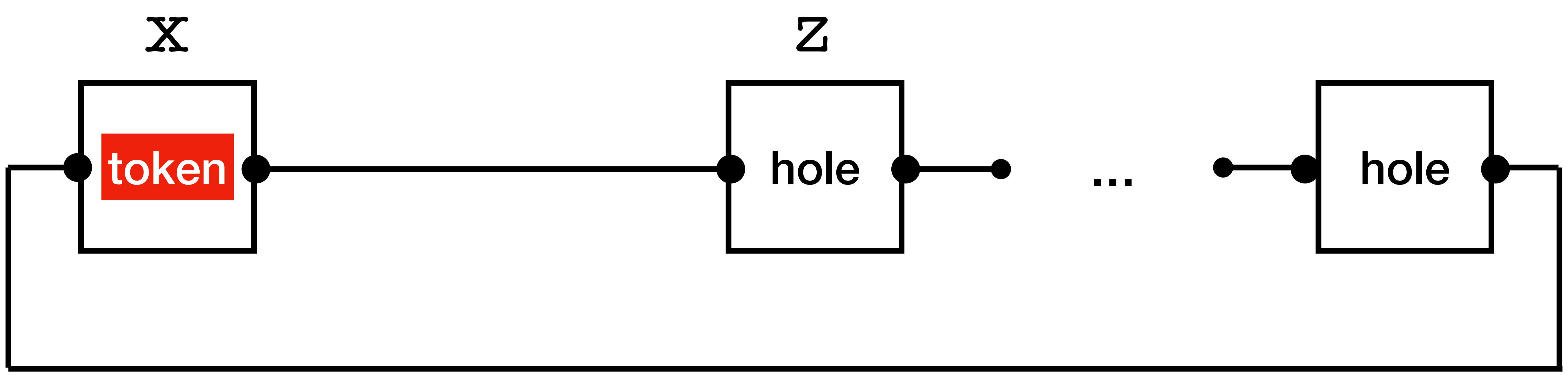
```
with x,y,z : ⟨x.out,y.in⟩ * [y]@hole* ⟨y.out,z.in⟩ do
  disconnect(x.out,y.in);
  disconnect(y.out,z.in);
  delete(y);
  connect(x.out,z.in);
od
```

An example: token-ring node removal



```
{ $Ring_{2,1}()$ }
  with x,y,z : ⟨x.out,y.in⟩ * [y]@hole* ⟨y.out,z.in⟩ do
    disconnect(x.out,y.in);
    disconnect(y.out,z.in);
    delete(y);
    connect(x.out,z.in);
  od
{ $Ring_{1,1}()$ }
```

An example: token-ring node removal



{ $\text{Ring}_{2,1}()$ }

```
with x,y,z : ⟨x.out,y.in⟩ * [y]@hole * ⟨y.out,z.in⟩ do
  disconnect(x.out,y.in);
  disconnect(y.out,z.in);
  delete(y);
  connect(x.out,z.in);
od
```

{ $\text{Ring}_{1,1}()$ } → safe

Local Reasoning

{emp} **new(x,q)** {[x]@q}

{[x]@q} **delete(x)** {emp}

{emp} **connect($x_1.p_1$, ..., $x_n.p_n$)** { ⟨$x_1.p_1$ …., $x_n.p_n$⟩ }

{ ⟨$x_1.p_1$ …., $x_n.p_n$⟩ } **disconnect($x_1.p_1$, ..., $x_n.p_n$)** {emp}

A local specification only mentions those resources that are necessary to avoid faulting

Local Reasoning

$\{\mathrm{emp}\}\ \mathbf{new}(\mathbf{x},\mathbf{q})\ \{[x]@q\}$

$\{[x]@q\}\ \mathbf{delete}(\mathbf{x})\ \{\mathrm{emp}\}$

$\{\mathrm{emp}\}\ \mathbf{connect}(\mathbf{x}_1.\mathbf{p}_1, \ldots, \mathbf{x}_n.\mathbf{p}_n)\ \{\ \langle x_1.p_1\ \ldots., x_n.p_n\rangle\ \}$

$\{\ \langle x_1.p_1\ \ldots., x_n.p_n\rangle\ \}\ \mathbf{disconnect}(\mathbf{x}_1.\mathbf{p}_1, \ldots, \mathbf{x}_n.\mathbf{p}_n)\ \{\mathrm{emp}\}$

$$\frac{\{\phi\}\ \mathrm{R}\ \{\Psi\}}{\{\phi\ {*}\ F\}\ \mathrm{R}\ \{\Psi\ {*}\ F\}}$$

if R is a local program and

$\mathrm{modifies}(\mathrm{R}) \cap \mathrm{fv}(F) = \varnothing$

A local specification only mentions those resources that are necessary to avoid faulting

The frame rule plugs a local specification into a global context

Which Reconfiguration Programs are Local?

Let Γ be the set of configurations

An action is a function $f : \Gamma \rightarrow \mathrm{pow}(\Gamma)^{T}$, where $\mathcal{S} \subseteq T$, $\forall \mathcal{S} \in \mathrm{pow}(\Gamma)$

Which Reconfiguration Programs are Local?

Let Γ be the set of configurations

An action is a function f : Γ→ pow(Γ)^{T}, where $\mathcal{S} \subseteq T$, $\forall \mathcal{S} \in$ pow(Γ)

An action f is local ⇔ $f(\gamma_1 * \gamma_2) \subseteq f(\gamma_1) * \{\gamma_2\}$

Which Reconfiguration Programs are Local?

Let Γ be the set of configurations

An action is a function f : Γ→ pow(Γ)$^{\top}$, where $\mathcal{S} \subseteq \top$, $\forall \mathcal{S} \in \mathrm{pow}(\Gamma)$

An action f is local ⇔ $f(\gamma_1 * \gamma_2) \subseteq f(\gamma_1) * \{\gamma_2\}$

- $\mathtt{new(x,q)}$, $\mathtt{delete(x)}$, $\mathtt{connect(x_1.p_1, \ldots, x_n.p_n)}$, $\mathtt{disconnect(x_1.p_1, \ldots, x_n.p_n)}$
- with $\mathtt{x_1, \ldots, x_n} : \phi$ do … od, where ϕ is a conjunction of equalities
- nondeterministic choices $\mathtt{R_1} + \mathtt{R_2}$ between local programs

Which Reconfiguration Programs are Local?

Let Γ be the set of configurations

An action is a function $f : \Gamma \rightarrow pow(\Gamma)^{T}$, where $\mathcal{S} \subseteq T$, $\forall \mathcal{S} \in pow(\Gamma)$

An action f is local $\Leftrightarrow f(\gamma_1 * \gamma_2) \subseteq f(\gamma_1) * \{\gamma_2\}$

- new(x,q), delete(x), connect($x_1.p_1, \ldots, x_n.p_n$), disconnect($x_1.p_1, \ldots, x_n.p_n$)
- with $x_1, \ldots, x_n : \phi$ do … od, where ϕ is a conjunction of equalities
- nondeterministic choices $R_1 + R_2$ between local programs

Non-local programs:

- sequential compositions $R_1; R_2$
- with $x_1, \ldots, x_n : \phi$ do … od, where ϕ contains node/interaction atoms

Sequential Composition

$$\frac{\{\phi\}\ R_1\ \{\theta\} \qquad \{\theta\}\ R_2\ \{\Psi\}}{\{\phi\}\ R_1;\ R_2\ \{\Psi\}}$$

Sequential Composition

$$\frac{\{\phi\}\ R_1\ \{\theta\} \qquad \{\theta\}\ R_2\ \{\Psi\}}{\{\phi\}\ R_1;\ R_2\ \{\Psi\}}$$

θ is havoc invariant

A formula ϕ is havoc invariant ⇔ *for each model ɣ of ϕ and each state change ɣ →* ɣ' corresponding to firing one or more interactions enabled in ɣ, ɣ' is a model of ϕ*

Conditional Rule

$$\frac{\{\phi \wedge (\theta * \mathrm{true})\}\ \mathtt{R}\ \{\Psi\}}{\{\phi\}\ \mathtt{with}\ \mathbf{x}{:}\theta\ \mathtt{do}\ \mathtt{R}\ \mathtt{od}\ \{\exists \mathbf{x}\ .\ \Psi\}} \qquad \mathrm{fv}(\phi) \cap \mathbf{x} = \varnothing$$

The premiss introduces both boolean and separating conjunction

Conditional Rule

$$\frac{\{\theta * F\}\ \mathtt{R}\ \{\Psi\}}{\{\phi\}\ \mathtt{with}\ \mathbf{x}{:}\theta\ \mathtt{do}\ \mathtt{R}\ \mathtt{od}\ \{\exists \mathbf{x}\ .\ \Psi\}}\quad \mathrm{fv}(\phi) \cap \mathbf{x} = \varnothing$$

The premiss introduces both boolean and separating conjunction

The boolean conjunction can be eliminated by solving a frame inference problem:

Find the strongest formula (if one exists) F such that $\phi \models \theta * F$

Back to the proof

```
{ $\text{Ring}_{2,1}()$ }

with x,y,z : ⟨x.out,y.in⟩ * [y]@hole* ⟨y.out,z.in⟩ do

  disconnect(x.out,y.in);

  disconnect(y.out,z.in);

  delete(y);

  connect(x.out,z.in);

od

{ $\text{Ring}_{1,1}()$ }
```

Back to the proof

```
{ $\mathrm{Ring}_{2,1}()$ }
{ $\mathrm{Chain}_{2,1}(x,z)$ *  ⟨z.out,x.in⟩  }
with x,y,z : ⟨x.out,y.in⟩ * [y]@hole* ⟨y.out,z.in⟩ do

  disconnect(x.out,y.in);

  disconnect(y.out,z.in);

  delete(y);

  connect(x.out,z.in);

od

{ $\mathrm{Ring}_{1,1}()$ }
```

Back to the proof

```
{ $\mathrm{Ring}_{2,1}()$ }
{ $\mathrm{Chain}_{2,1}(x,z)$ * ⟨z.out,x.in⟩ }
with x,y,z : ⟨x.out,y.in⟩ * [y]@hole * ⟨y.out,z.in⟩ do
{ ⟨x.out,y.in⟩ * [y]@hole* ⟨y.out,z.in⟩ * $\mathrm{Chain}_{1,1}(z,x)$ }
  disconnect(x.out,y.in);

  disconnect(y.out,z.in);

  delete(y);

  connect(x.out,z.in);

od

{ $\mathrm{Ring}_{1,1}()$ }
```

Back to the proof

```
{ Ring$_{2,1}$() }
{ Chain$_{2,1}$(x,z) * ⟨z.out,x.in⟩ }
with x,y,z : ⟨x.out,y.in⟩ * [y]@hole* ⟨y.out,z.in⟩ do
{ ⟨x.out,y.in⟩ * [y]@hole* ⟨y.out,z.in⟩ * Chain$_{1,1}$(z,x) }
   disconnect(x.out,y.in);
{ [y]@hole* ⟨y.out,z.in⟩ * Chain$_{1,1}$(z,x) }
   disconnect(y.out,z.in);

   delete(y);

   connect(x.out,z.in);

od

{ Ring$_{1,1}$() }
```

{ ⟨x.out,y.in⟩ } disconnect(x.out,y.in) { emp }

Back to the proof

```
{ $Ring_{2,1}()$ }
{ $Chain_{2,1}(x,z)$ * ⟨z.out,x.in⟩ }
with x,y,z : ⟨x.out,y.in⟩ * [y]@hole* ⟨y.out,z.in⟩ do
{ ⟨x.out,y.in⟩ * [y]@hole* ⟨y.out,z.in⟩ * $Chain_{1,1}(z,x)$ }
  disconnect(x.out,y.in);
{ [y]@hole * ⟨y.out,z.in⟩ * $Chain_{1,1}(z,x)$ }
  disconnect(y.out,z.in);

  delete(y);

  connect(x.out,z.in);

od

{ $Ring_{1,1}()$ }
```

{ ⟨x.out,y.in⟩ } disconnect(x.out,y.in) { emp }

{ ⟨y.out,z.in⟩ } disconnect(y.out,z.in) { emp }

Back to the proof

```
{ $\mathrm{Ring}_{2,1}()$ }
{ $\mathrm{Chain}_{2,1}(x,z)$ * ⟨z.out,x.in⟩ }
with x,y,z : ⟨x.out,y.in⟩ * [y]@hole * ⟨y.out,z.in⟩ do
{ ⟨x.out,y.in⟩ * [y]@hole* ⟨y.out,z.in⟩ * $\mathrm{Chain}_{1,1}(z,x)$ }
  disconnect(x.out,y.in);
{ [y]@hole * ⟨y.out,z.in⟩ * $\mathrm{Chain}_{1,1}(z,x)$ }
  disconnect(y.out,z.in);
{ [y]@hole * $\mathrm{Chain}_{1,1}(z,x)$ }
  delete(y);

  connect(x.out,z.in);

od

{ $\mathrm{Ring}_{1,1}()$ }
```

{ ⟨x.out,y.in⟩ } disconnect(x.out,y.in) { emp }

{ ⟨y.out,z.in⟩ } disconnect(y.out,z.in) { emp }

Back to the proof

```
{ Ring$_{2,1}$() }
{ Chain$_{2,1}$(x,z) * ⟨z.out,x.in⟩ }
with x,y,z : ⟨x.out,y.in⟩ * [y]@hole * ⟨y.out,z.in⟩ do
{ ⟨x.out,y.in⟩ * [y]@hole* ⟨y.out,z.in⟩ * Chain$_{1,1}$(z,x) }
  disconnect(x.out,y.in);
{ [y]@hole * ⟨y.out,z.in⟩ * Chain$_{1,1}$(z,x) }
  disconnect(y.out,z.in);
{ [y]@hole * Chain$_{1,1}$(z,x) }
  delete(y);

  connect(x.out,z.in);

od

{ Ring$_{1,1}$() }
```

{ ⟨x.out,y.in⟩ } disconnect(x.out,y.in) { emp }

{ ⟨y.out,z.in⟩ } disconnect(y.out,z.in) { emp }

Back to the proof

{ Ring$_{2,1}$() }

{ Chain$_{2,1}$(x,z) * ⟨z.out,x.in⟩ }

with x,y,z : ⟨x.out,y.in⟩ * [y]@hole * ⟨y.out,z.in⟩ do

{ ⟨x.out,y.in⟩ * [y]@hole* ⟨y.out,z.in⟩ * Chain$_{1,1}$(z,x) }

  disconnect(x.out,y.in);

{ [y]@hole * ⟨y.out,z.in⟩ * Chain$_{1,1}$(z,x) }

  disconnect(y.out,z.in);

{ [y]@hole * Chain$_{1,1}$(z,x) }

  delete(y);

{ Chain$_{1,1}$(z,x) }

  connect(x.out,z.in);

od

{ Ring$_{1,1}$() }

{ ⟨x.out,y.in⟩ } disconnect(x.out,y.in) { emp }

{ ⟨y.out,z.in⟩ } disconnect(y.out,z.in) { emp }

{ [y] } delete(y) { emp }

Back to the proof

```
{ $\mathrm{Ring}_{2,1}()$ }
{ $\mathrm{Chain}_{2,1}(x,z) * \langle z.out,x.in \rangle$ }
with x,y,z : $\langle x.out,y.in \rangle * [y]@hole * \langle y.out,z.in \rangle$ do
{ $\langle x.out,y.in \rangle * [y]@hole * \langle y.out,z.in \rangle * \mathrm{Chain}_{1,1}(z,x)$ }
   disconnect(x.out,y.in);
{ $[y]@hole * \langle y.out,z.in \rangle * \mathrm{Chain}_{1,1}(z,x)$ }
   disconnect(y.out,z.in);
{ $[y]@hole * \mathrm{Chain}_{1,1}(z,x)$ }
   delete(y);
{ $\mathrm{Chain}_{1,1}(z,x)$ }
   connect(x.out,z.in);
{ $\mathrm{Chain}_{1,1}(z,x) * \langle x.out,z.in \rangle$ }
od
{ $\exists x \exists z.\mathrm{Chain}_{1,1}(z,x) * \langle z.out,x.in \rangle$ }
{ $\mathrm{Ring}_{1,1}()$ }
```

{ $\langle x.out,y.in \rangle$ } disconnect(x.out,y.in) { emp }

{ $\langle y.out,z.in \rangle$ } disconnect(y.out,z.in) { emp }

{ [y] } delete(y) { emp }

{ emp } connect(x.out,z.in) { $\langle x.out,z.in \rangle$ }

Back to the proof

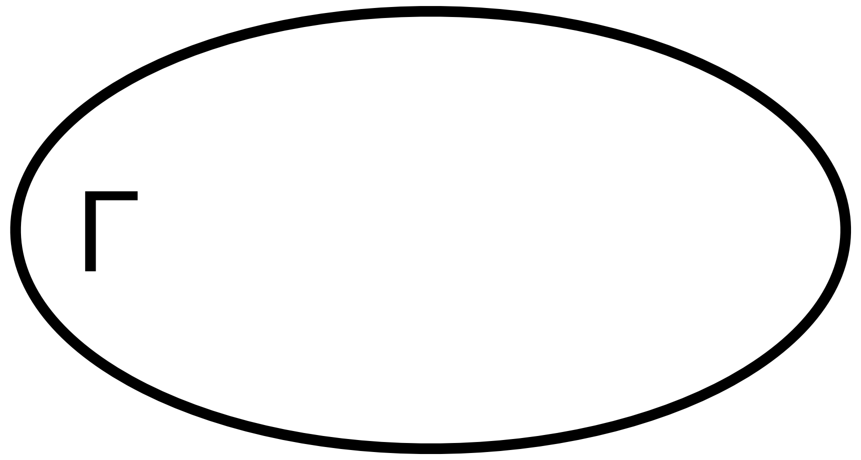
```
{ $Ring_{2,1}()$ }
{ $Chain_{2,1}(x,z)$ * ⟨z.out,x.in⟩ }
with x,y,z : ⟨x.out,y.in⟩ * [y]@hole* ⟨y.out,z.in⟩ do
{ ⟨x.out,y.in⟩ * [y]@hole* ⟨y.out,z.in⟩ * $Chain_{1,1}(z,x)$ }
   disconnect(x.out,y.in);
{ [y]@hole * ⟨y.out,z.in⟩ * $Chain_{1,1}(z,x)$ }
   disconnect(y.out,z.in);
{ [y]@hole * $Chain_{1,1}(z,x)$ }
   delete(y);
{ $Chain_{1,1}(z,x)$ }
   connect(x.out,z.in);
{ $Chain_{1,1}(z,x)$ * ⟨x.out,z.in⟩ }
od
{ ∃x∃z.$Chain_{1,1}(z,x)$ * ⟨z.out,x.in⟩ }
{ $Ring_{1,1}()$ }
```

Back to the proof

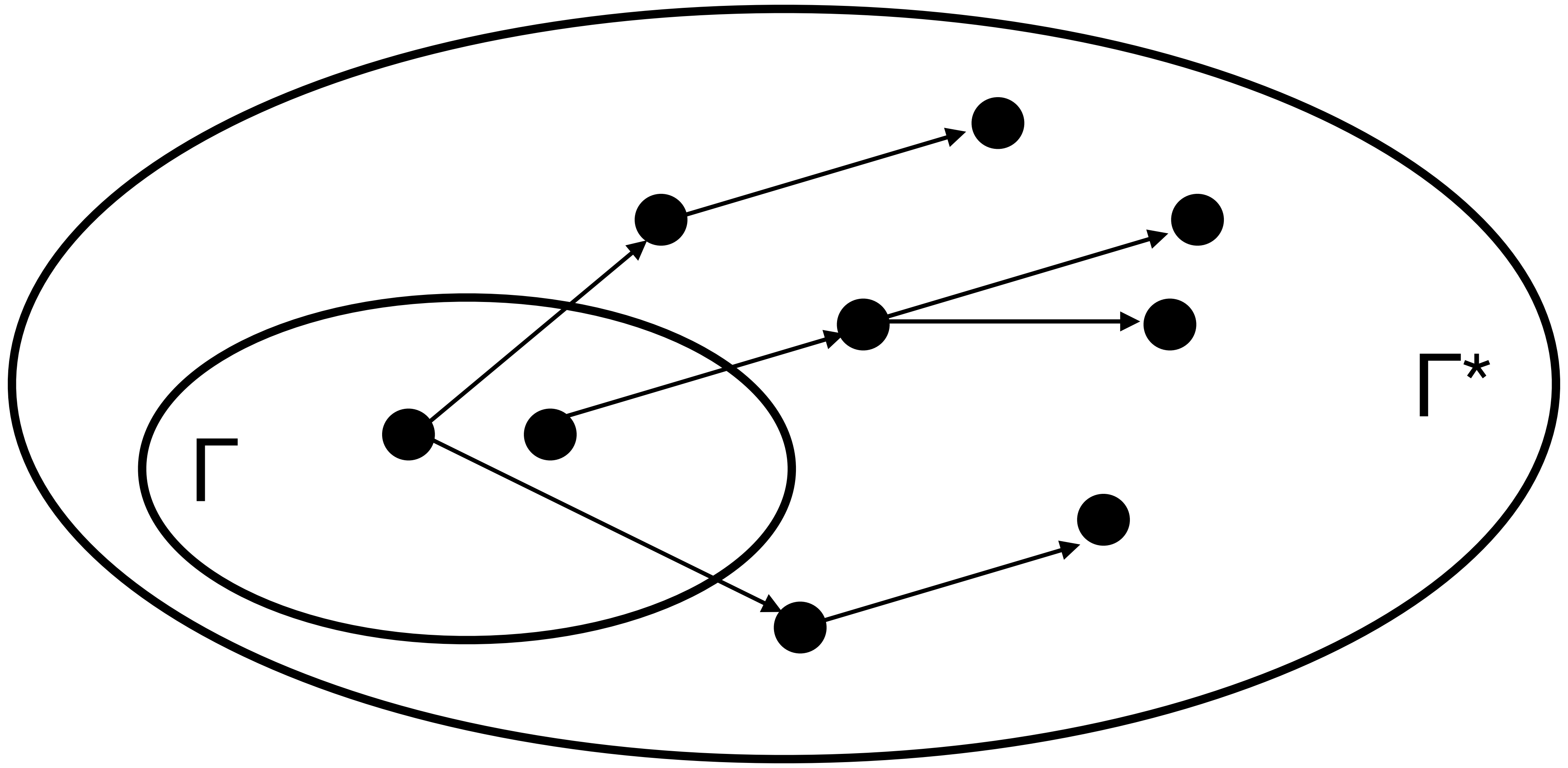
```
{ $\text{Ring}_{2,1}()$ }
{ $\text{Chain}_{2,1}(x,z)$ * ⟨z.out,x.in⟩ }
with x,y,z : ⟨x.out,y.in⟩ * [y]@hole* ⟨y.out,z.in⟩ do
{ ⟨x.out,y.in⟩ * [y]@hole* ⟨y.out,z.in⟩ * $\text{Chain}_{1,1}(z,x)$ }
   disconnect(x.out,y.in);
{ [y]@hole * ⟨y.out,z.in⟩ * $\text{Chain}_{1,1}(z,x)$ }
   disconnect(y.out,z.in);
{ [y]@hole * $\text{Chain}_{1,1}(z,x)$ }
   delete(y);
{ $\text{Chain}_{1,1}(z,x)$ }
   connect(x.out,z.in);
{ $\text{Chain}_{1,1}(z,x)$ * ⟨x.out,z.in⟩ }
od
{ $\exists x \exists z.\text{Chain}_{1,1}(z,x)$ * ⟨z.out,x.in⟩ }
{ $\text{Ring}_{1,1}()$ }
```

havoc invariant?

Checking Havoc Invariance

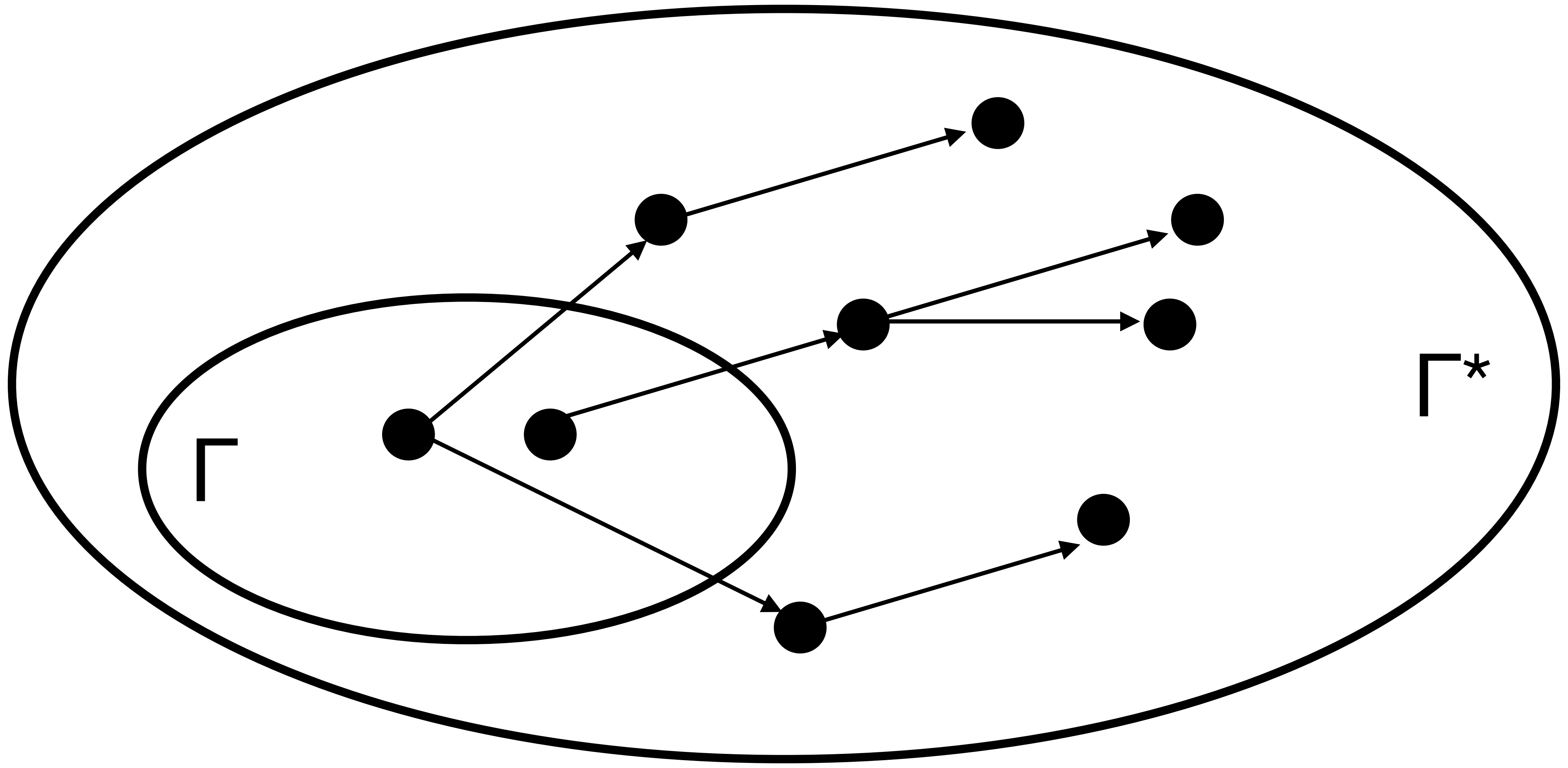


Checking Havoc Invariance

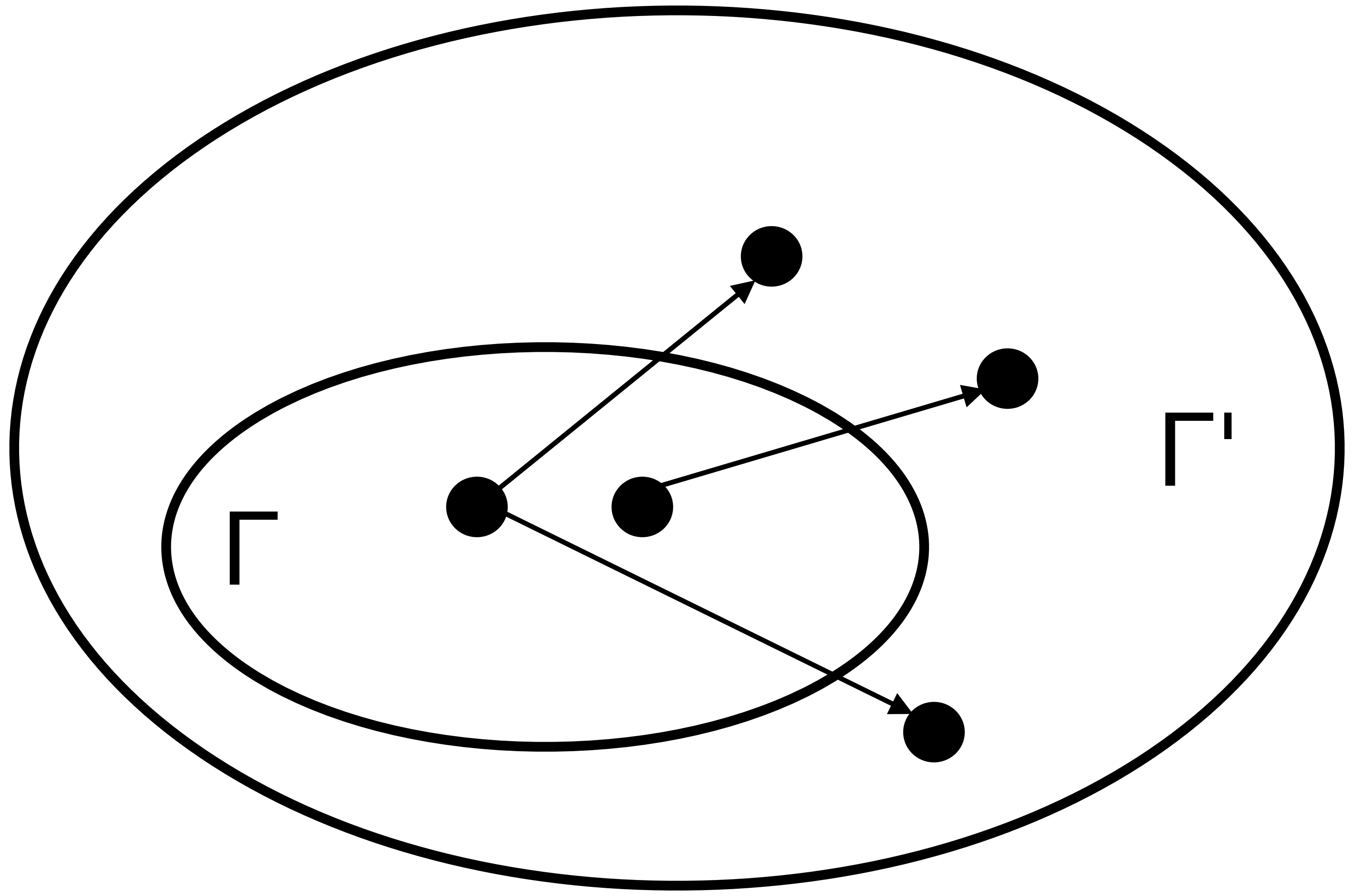


Γ* ⊆ Γ ?

Checking Havoc Invariance



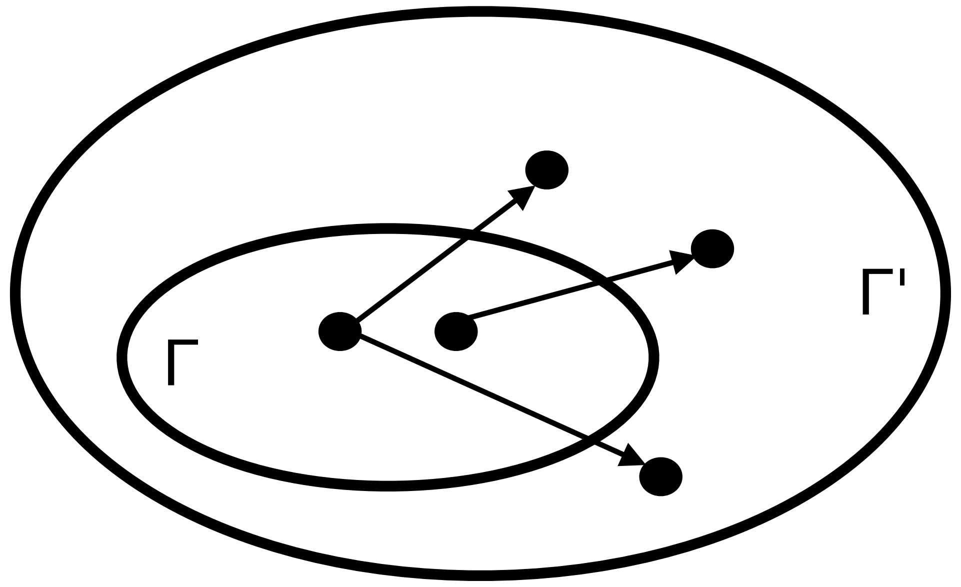
Γ* ⊆ Γ ?



Γ' ⊆ Γ ?

Checking Havoc Invariance

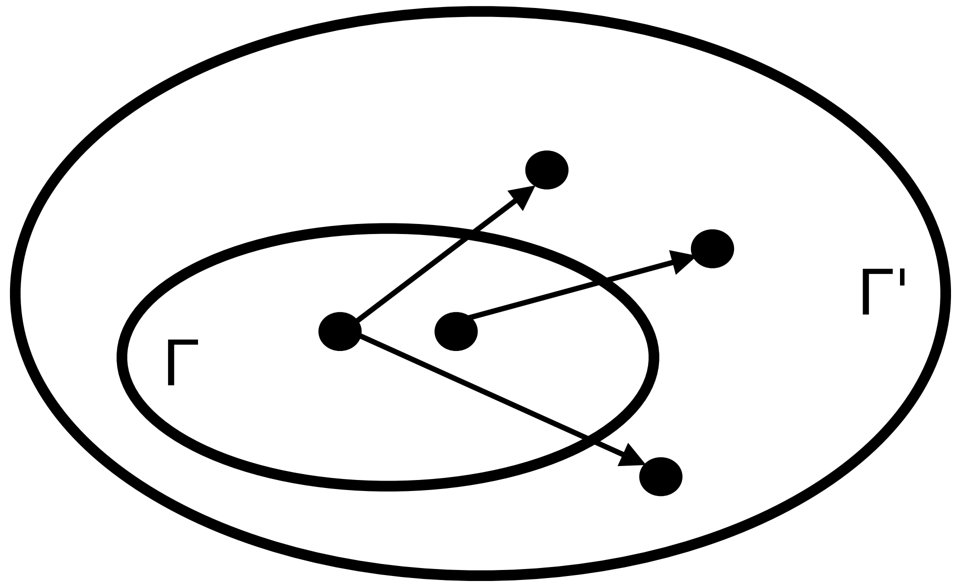
(Δ,A) describes Γ ⇢ (Δ',A') describes Γ'



$\Gamma' \subseteq \Gamma$?

Checking Havoc Invariance

(Δ,A) describes Γ ⋯⋯→ (Δ',A') describes Γ'



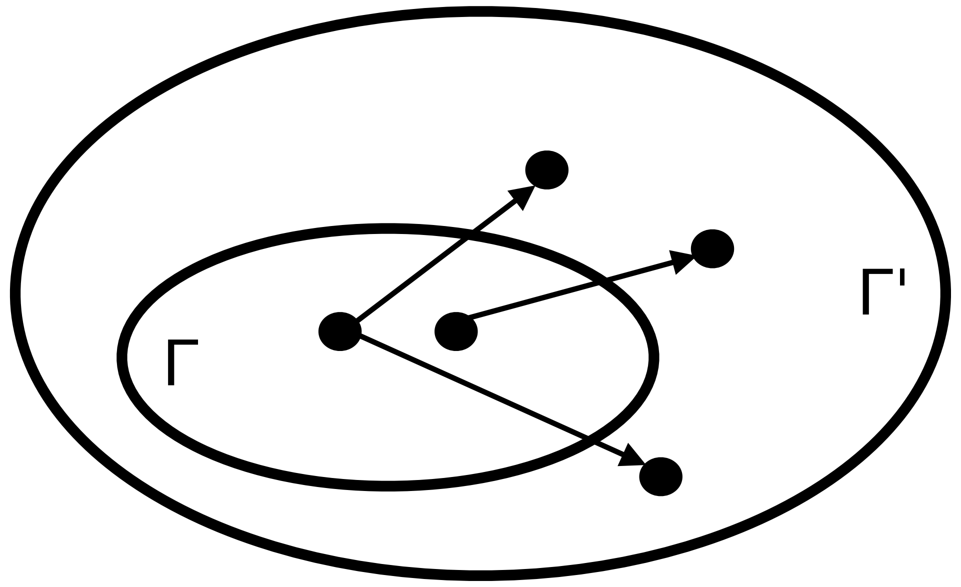
Γ' ⊆ Γ ?

Configurations are encoded as unfolding trees labeled with CL formulae

Checking Havoc Invariance

(Δ,A) describes Γ

(Δ',A') describes Γ'



Γ' ⊆ Γ ?

$\mathscr{A}_{\Delta,A}$ tree automaton

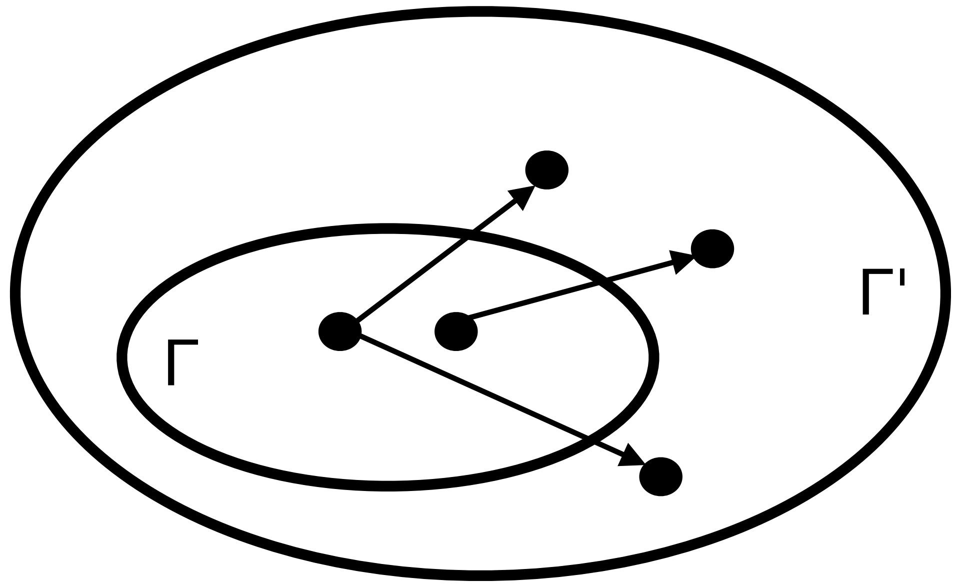
recognizing the unfolding trees of Δ for the formula $A(x_1 \ldots x_n)$

Configurations are encoded as unfolding trees labeled with CL formulae

Checking Havoc Invariance

(Δ,A) describes Γ

(Δ',A') describes Γ'



Γ' ⊆ Γ ?

$\mathcal{A}_{\Delta,A}$ tree automaton

recognizing the unfolding trees of Δ for the formula $A(x_1 \ldots x_n)$

$\mathcal{T}$ tree transducer

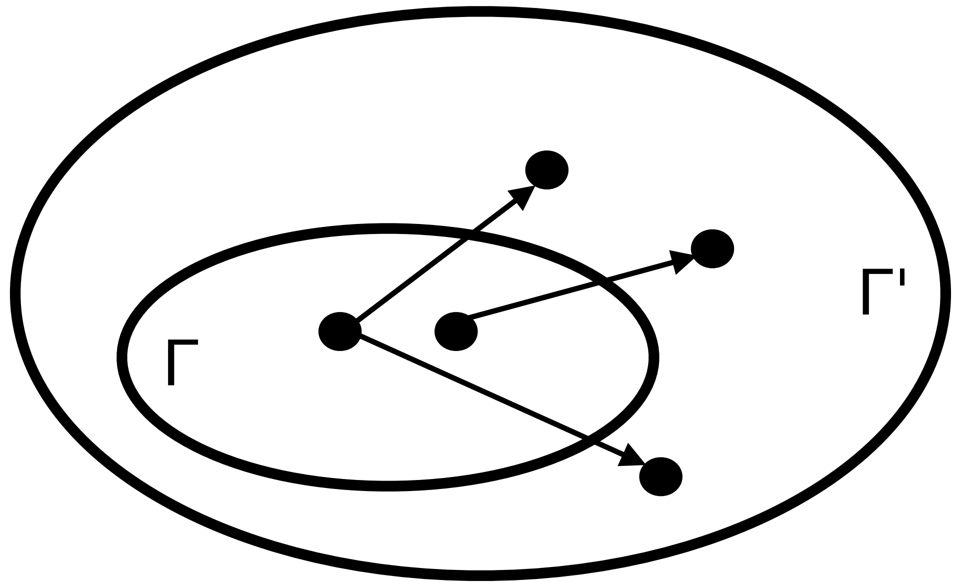
encoding the effect of one havoc step

Configurations are encoded as unfolding trees labeled with CL formulae

Checking Havoc Invariance

(Δ,A) describes Γ

(Δ',A') describes Γ'



Γ' ⊆ Γ ?

$\mathscr{A}_{\Delta,A}$ tree automaton

recognizing the unfolding trees of Δ for the formula A(x_1 ... x_n)

$\mathscr{T}$ tree transducer

encoding the effect of one havoc step

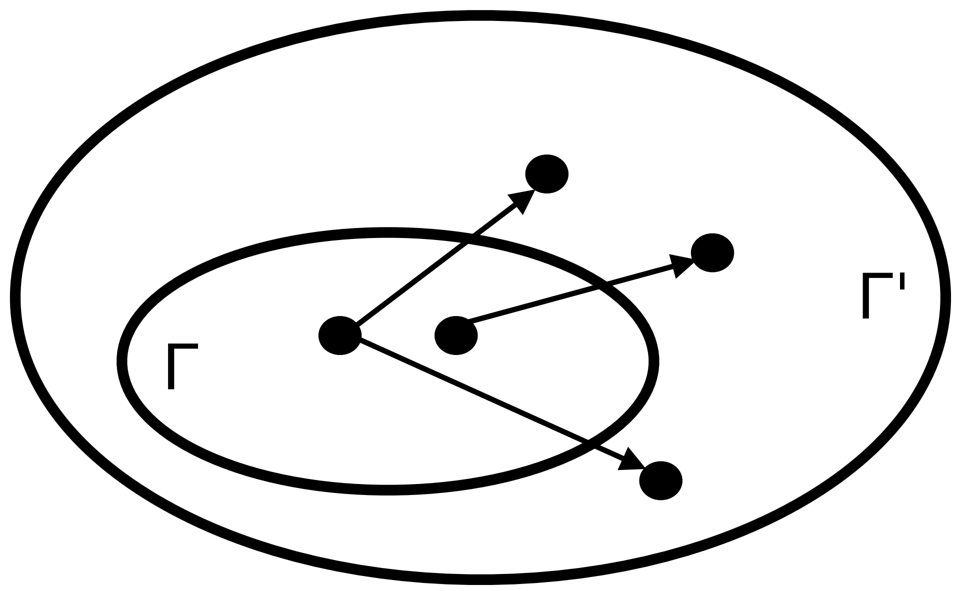
$\mathscr{T}$ ($\mathscr{A}_{\Delta,A}$)

Configurations are encoded as unfolding trees labeled with CL formulae

Checking Havoc Invariance

(Δ,A) describes Γ

(Δ',A') describes Γ'



Γ' ⊆ Γ ?

$\mathcal{A}_{\Delta,A}$ tree automaton

recognizing the unfolding trees of Δ for the formula $A(x_1 \ldots x_n)$

$\mathcal{T}$ tree transducer

encoding the effect of one havoc step

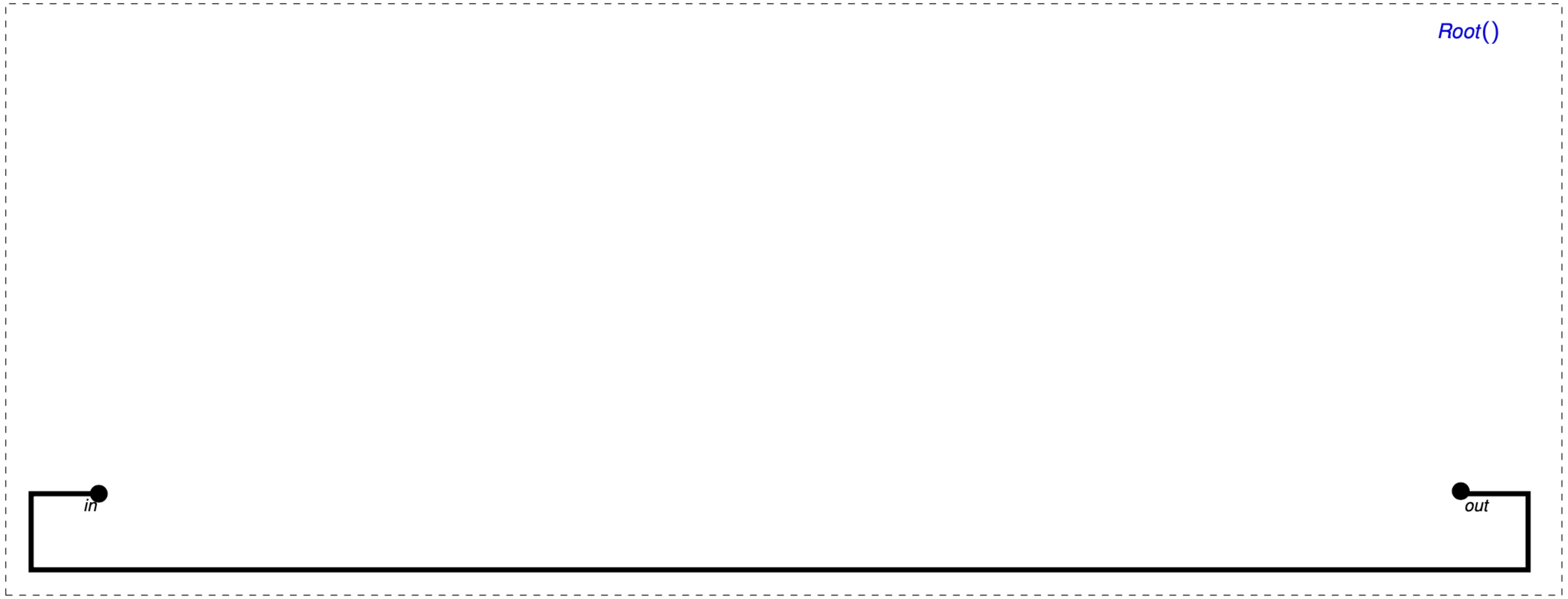
$\mathcal{T}(\mathcal{A}_{\Delta,A})$

Configurations are encoded as unfolding trees labeled with CL formulae

Check the entailment $A'(x_1 \ldots x_n) \models_{\Delta \cup \Delta'} A(x_1 \ldots x_n)$

A Tree with Leaves Linked in a Ring

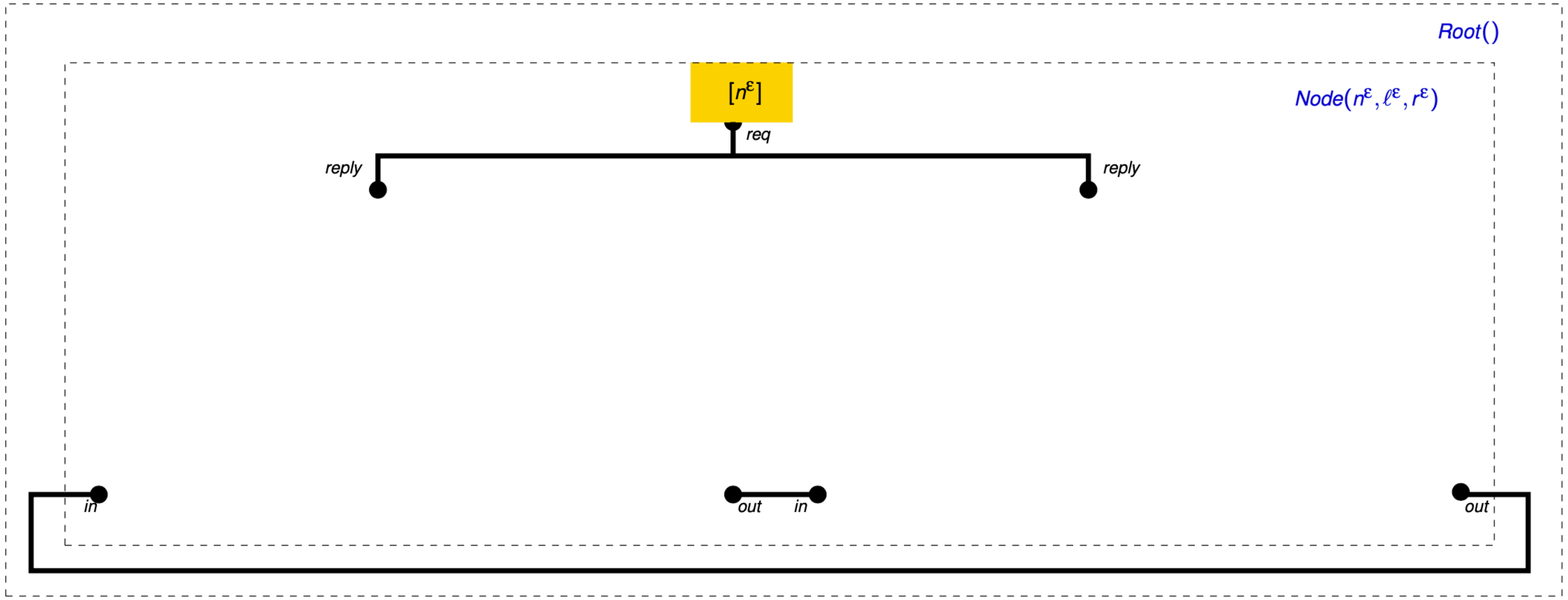
(α) $\mathit{Root}() \leftarrow \exists n \exists \ell \exists r\ .\ \langle r.\mathit{out}, \ell.\mathit{in} \rangle * \mathit{Node}(n, \ell, r)$



A Tree with Leaves Linked in a Ring

$$(\alpha) \quad \mathit{Root}() \leftarrow \exists n \exists \ell \exists r \,.\, \langle r.\mathit{out}, \ell.\mathit{in} \rangle * \mathit{Node}(n, \ell, r)$$

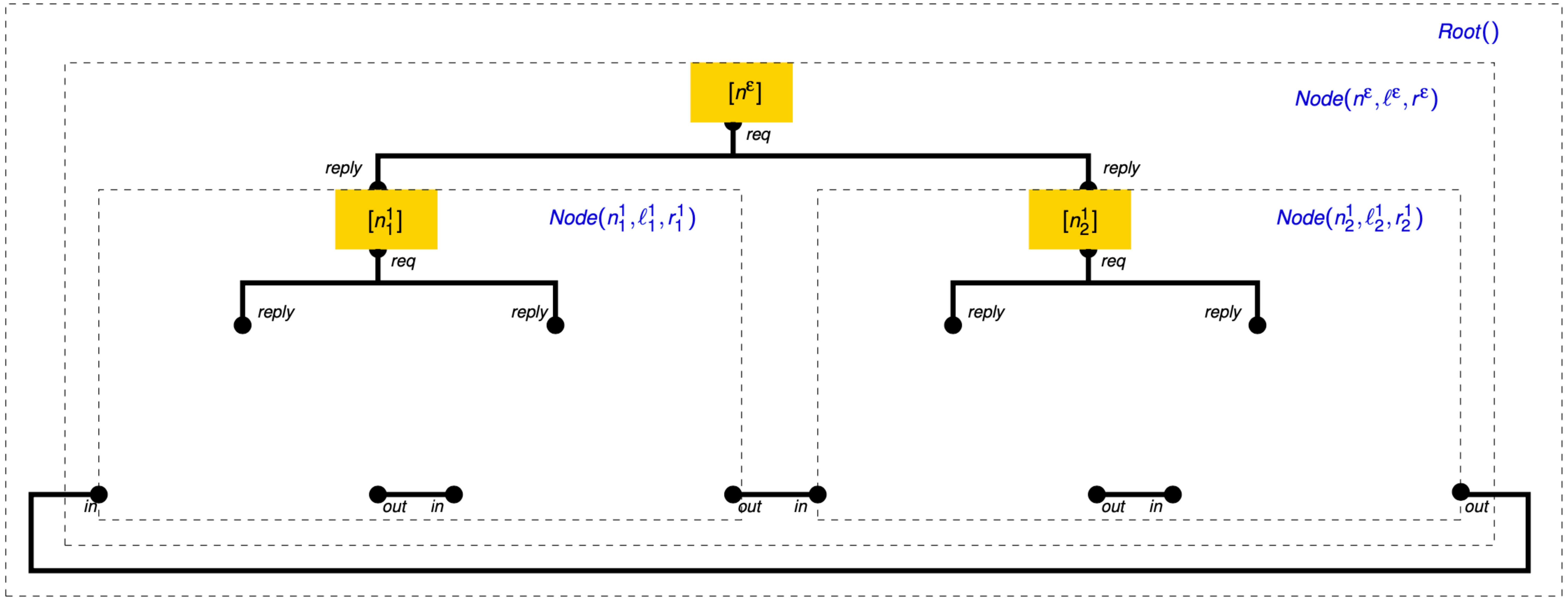
$$(\beta) \quad \mathit{Node}(n, \ell, r) \leftarrow \exists n_1 \exists r_1 \exists n_2 \exists \ell_2 \,.\, [n] * \langle n.\mathit{req}, n_1.\mathit{reply}, n_2.\mathit{reply} \rangle * \langle r_1.\mathit{out}, \ell_2.\mathit{in} \rangle * \mathit{Node}(n_1, \ell, r_1) * \mathit{Node}(n_2, \ell_2, r)$$



A Tree with Leaves Linked in a Ring

$$(\alpha)\quad \mathit{Root}() \leftarrow \exists n \exists \ell \exists r\ .\ \langle r.\mathit{out}, \ell.\mathit{in}\rangle * \mathit{Node}(n,\ell,r)$$

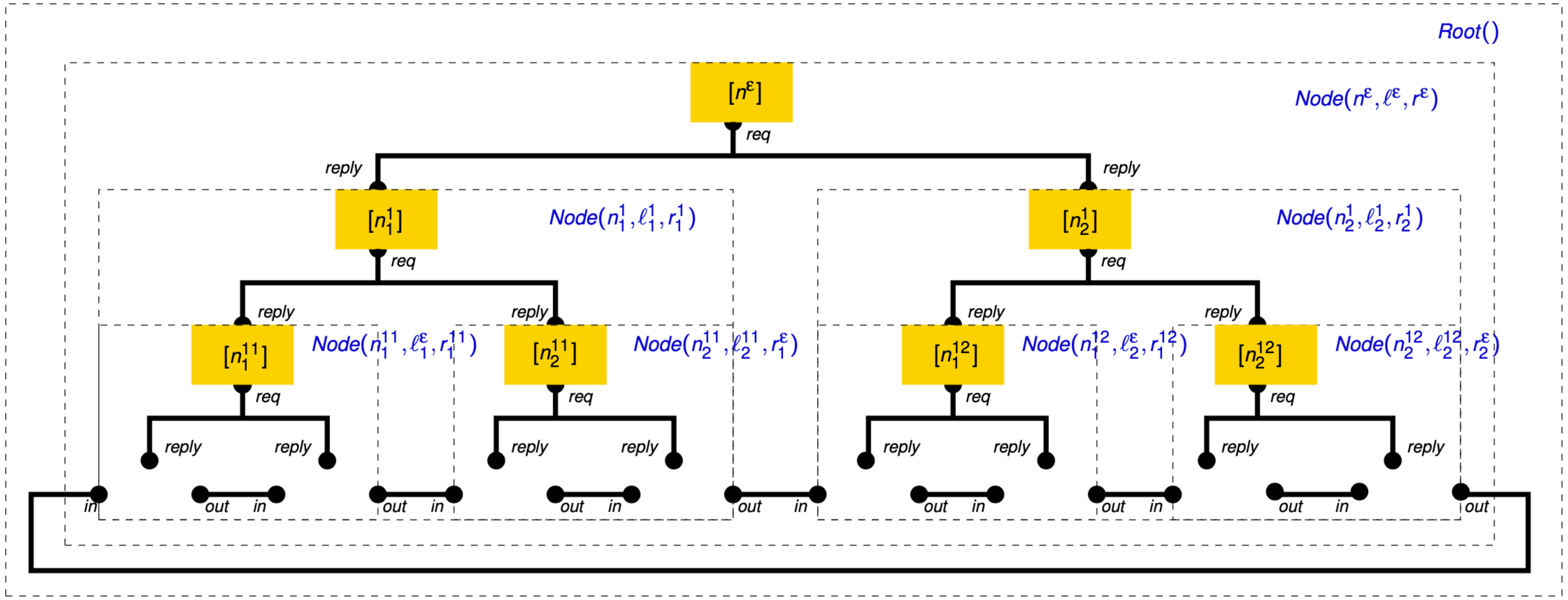
$$(\beta)\quad \mathit{Node}(n,\ell,r) \leftarrow \exists n_1 \exists r_1 \exists n_2 \exists \ell_2\ .\ [n] * \langle n.\mathit{req}, n_1.\mathit{reply}, n_2.\mathit{reply}\rangle * \langle r_1.\mathit{out}, \ell_2.\mathit{in}\rangle * \mathit{Node}(n_1,\ell,r_1) * \mathit{Node}(n_2,\ell_2,r)$$



$\mathit{Root}()$
$[n^{\varepsilon}]$
req
$\mathit{Node}(n^{\varepsilon},\ell^{\varepsilon},r^{\varepsilon})$
reply
reply
$[n_1^1]$
$\mathit{Node}(n_1^1,\ell_1^1,r_1^1)$
req
$[n_2^1]$
$\mathit{Node}(n_2^1,\ell_2^1,r_2^1)$
req
reply
reply
reply
reply
in
out
in
out
in
out
in
out

A Tree with Leaves Linked in a Ring

$$
\begin{array}{lrl}
(\alpha) & Root() & \leftarrow \exists n \exists \ell \exists r \,.\, \langle r.out, \ell.in \rangle * Node(n, \ell, r) \\
(\beta) & Node(n, \ell, r) & \leftarrow \exists n_1 \exists r_1 \exists n_2 \exists \ell_2 \,.\, [n] * \langle n.req, n_1.reply, n_2.reply \rangle * \langle r_1.out, \ell_2.in \rangle * Node(n_1, \ell, r_1) * Node(n_2, \ell_2, r)
\end{array}
$$



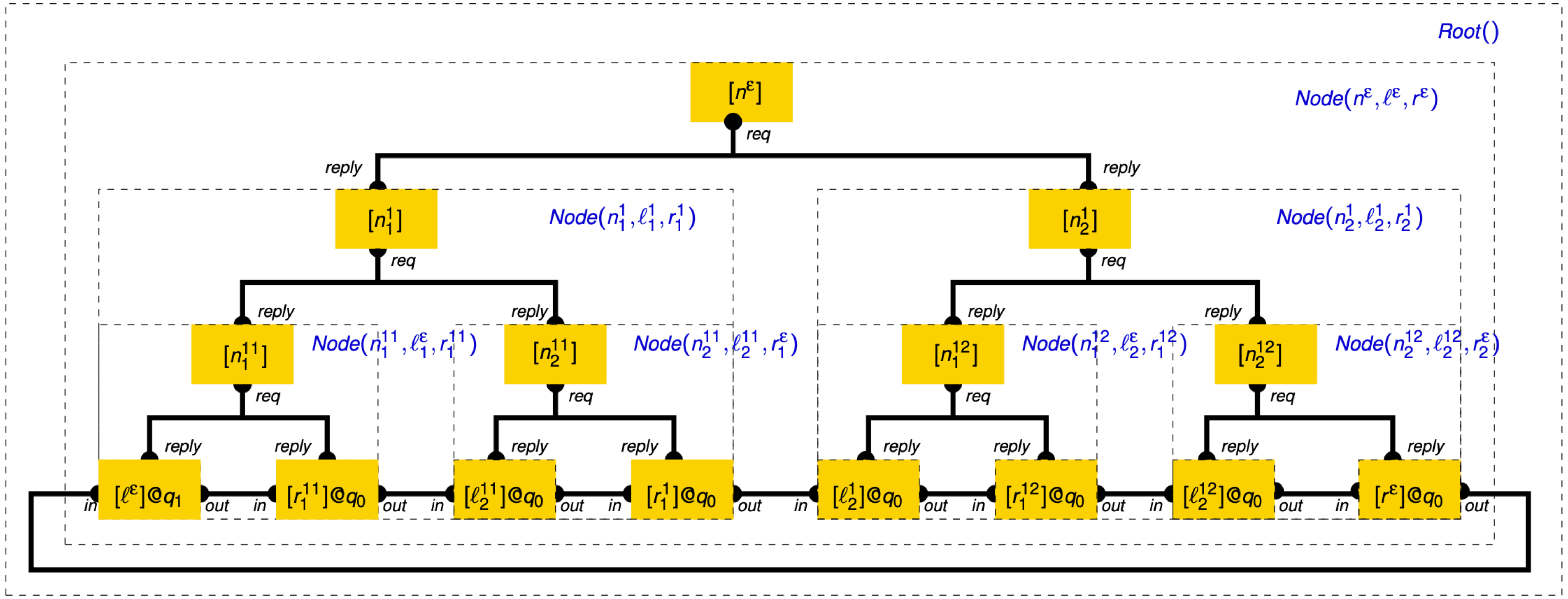
Root()
$[n^\varepsilon]$
req
$Node(n^\varepsilon, \ell^\varepsilon, r^\varepsilon)$
reply
reply
$[n_1^1]$
req
$Node(n_1^1, \ell_1^1, r_1^1)$
$[n_2^1]$
req
$Node(n_2^1, \ell_2^1, r_2^1)$
reply
reply
reply
reply
$[n_1^{11}]$
$Node(n_1^{11}, \ell_1^\varepsilon, r_1^{11})$
$[n_2^{11}]$
$Node(n_2^{11}, \ell_2^{11}, r_1^\varepsilon)$
$[n_1^{12}]$
$Node(n_1^{12}, \ell_2^\varepsilon, r_1^{12})$
$[n_2^{12}]$
$Node(n_2^{12}, \ell_2^{12}, r_2^\varepsilon)$
req
req
req
req
reply
reply
reply
reply
reply
reply
reply
reply
in
out
in
out
in
out
in
out
in
out
in
out
in
out

A Tree with Leaves Linked in a Ring

$$(\alpha)\quad Root() \leftarrow \exists n \exists \ell \exists r\, .\, \langle r.out, \ell.in \rangle * Node(n,\ell,r)$$

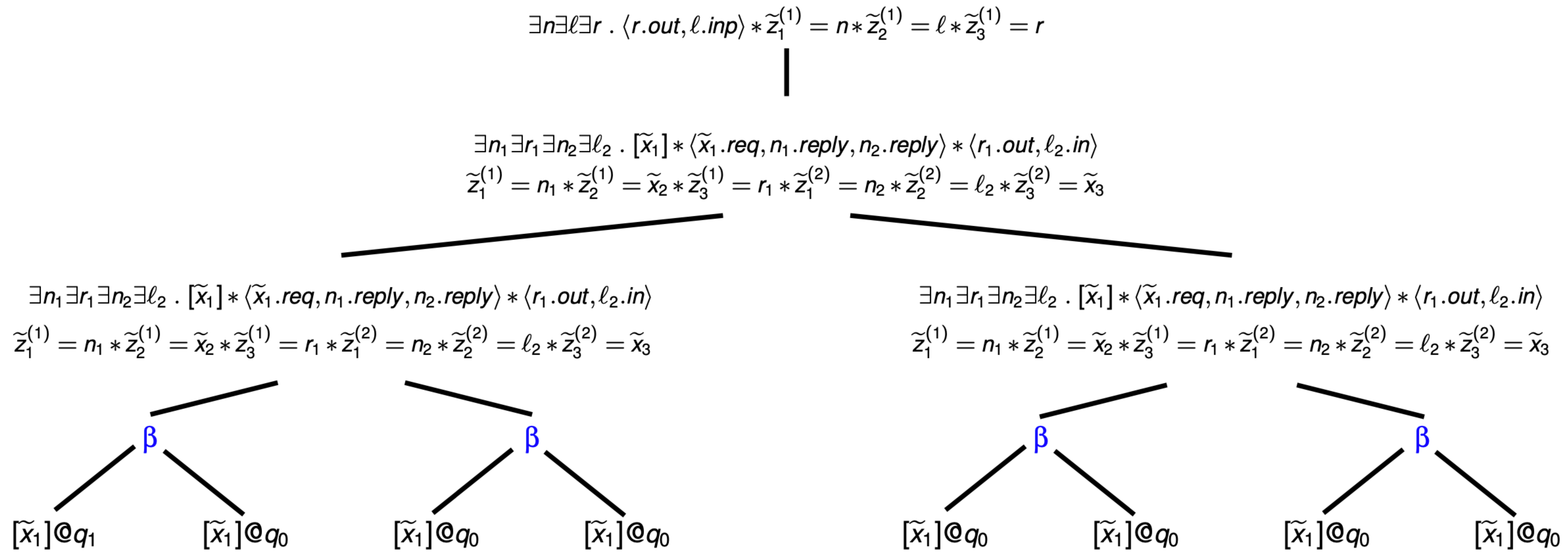
$$(\beta)\quad Node(n,\ell,r) \leftarrow \exists n_1 \exists r_1 \exists n_2 \exists \ell_2\, .\, [n] * \langle n.req, n_1.reply, n_2.reply \rangle * \langle r_1.out, \ell_2.in \rangle * Node(n_1,\ell,r_1) * Node(n_2,\ell_2,r)$$

$$(\gamma_0)\quad Node(n,\ell,r) \leftarrow [n]@q_0 * n = \ell * n = r \qquad (\gamma_1)\quad Node(n,\ell,r) \leftarrow [n]@q_1 * n = \ell * n = r$$



A Tree with Leaves Linked in a Ring

$$
\begin{array}{lll}
(\alpha) & Root() & \leftarrow \exists n \exists \ell \exists r \,.\, \langle r.out, \ell.in \rangle * Node(n,\ell,r) \\
(\beta) & Node(n,\ell,r) & \leftarrow \exists n_1 \exists r_1 \exists n_2 \exists \ell_2 \,.\, [n] * \langle n.req, n_1.reply, n_2.reply \rangle * \langle r_1.out, \ell_2.in \rangle * Node(n_1,\ell,r_1) * Node(n_2,\ell_2,r) \\
(\gamma_0) & Node(n,\ell,r) & \leftarrow [n]@q_0 * n = \ell * n = r \qquad (\gamma_1) \quad Node(n,\ell,r) \leftarrow [n]@q_1 * n = \ell * n = r
\end{array}
$$



$\exists n \exists \ell \exists r \,.\, \langle r.out, \ell.inp \rangle * \tilde{z}_1^{(1)} = n * \tilde{z}_2^{(1)} = \ell * \tilde{z}_3^{(1)} = r$

$\exists n_1 \exists r_1 \exists n_2 \exists \ell_2 \,.\, [\tilde{x}_1] * \langle \tilde{x}_1.req, n_1.reply, n_2.reply \rangle * \langle r_1.out, \ell_2.in \rangle$
$\tilde{z}_1^{(1)} = n_1 * \tilde{z}_2^{(1)} = \tilde{x}_2 * \tilde{z}_3^{(1)} = r_1 * \tilde{z}_1^{(2)} = n_2 * \tilde{z}_2^{(2)} = \ell_2 * \tilde{z}_3^{(2)} = \tilde{x}_3$

$\exists n_1 \exists r_1 \exists n_2 \exists \ell_2 \,.\, [\tilde{x}_1] * \langle \tilde{x}_1.req, n_1.reply, n_2.reply \rangle * \langle r_1.out, \ell_2.in \rangle$
$\tilde{z}_1^{(1)} = n_1 * \tilde{z}_2^{(1)} = \tilde{x}_2 * \tilde{z}_3^{(1)} = r_1 * \tilde{z}_1^{(2)} = n_2 * \tilde{z}_2^{(2)} = \ell_2 * \tilde{z}_3^{(2)} = \tilde{x}_3$

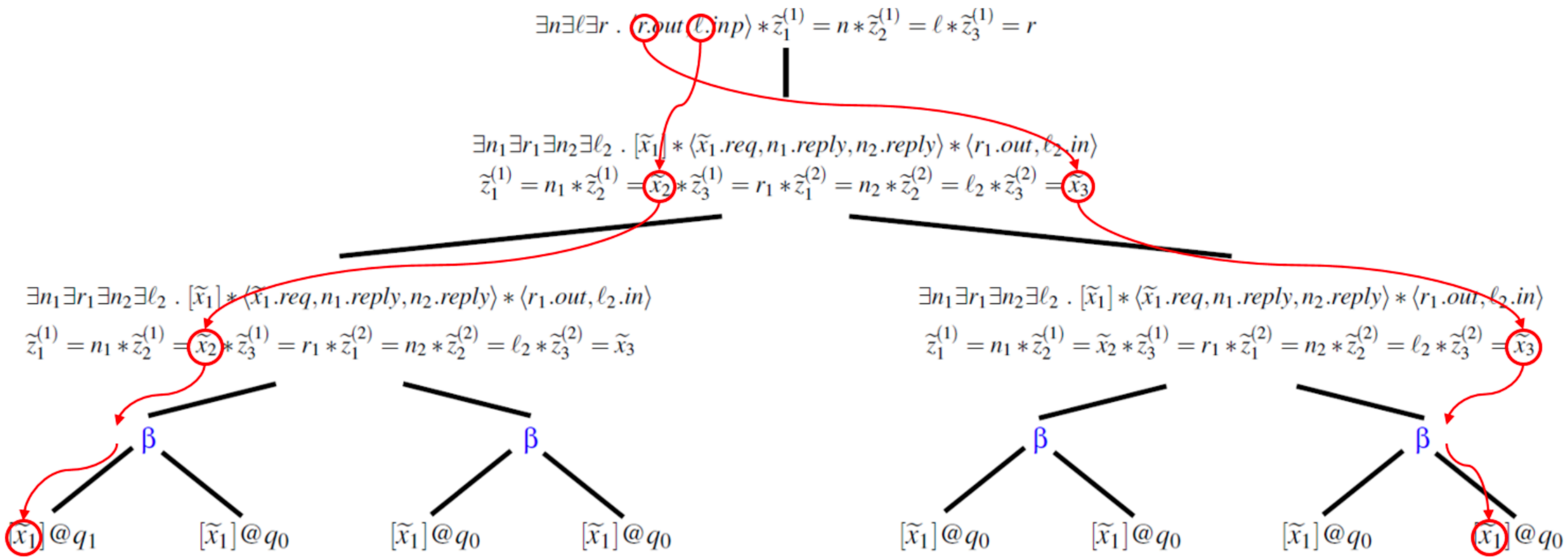
$\exists n_1 \exists r_1 \exists n_2 \exists \ell_2 \,.\, [\tilde{x}_1] * \langle \tilde{x}_1.req, n_1.reply, n_2.reply \rangle * \langle r_1.out, \ell_2.in \rangle$
$\tilde{z}_1^{(1)} = n_1 * \tilde{z}_2^{(1)} = \tilde{x}_2 * \tilde{z}_3^{(1)} = r_1 * \tilde{z}_1^{(2)} = n_2 * \tilde{z}_2^{(2)} = \ell_2 * \tilde{z}_3^{(2)} = \tilde{x}_3$

β β β β

$[\tilde{x}_1]@q_1$ $[\tilde{x}_1]@q_0$ $[\tilde{x}_1]@q_0$ $[\tilde{x}_1]@q_0$ $[\tilde{x}_1]@q_0$ $[\tilde{x}_1]@q_0$ $[\tilde{x}_1]@q_0$ $[\tilde{x}_1]@q_0$

Havoc Action as Tree Transductions

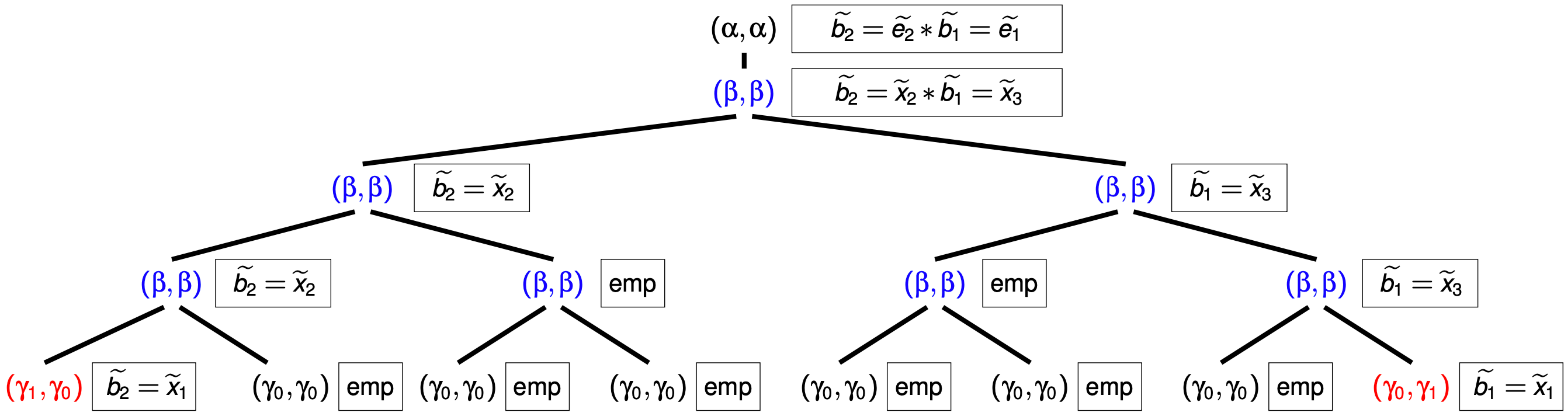
- Non-deterministically choses which interaction <$x_1.p_1$ ... $x_n.p_n$> is triggered
- Tracks each variable x_i to the atom [x]@q that instantiates it (creates the respective node)
- Change the states of these nodes according to the transitions of the behavior (state machine)



$\exists n \exists \ell \exists r \,.\, \langle r.out, \ell.inp \rangle * \widetilde{z}_1^{(1)} = n * \widetilde{z}_2^{(1)} = \ell * \widetilde{z}_3^{(1)} = r$
$\exists n_1 \exists r_1 \exists n_2 \exists \ell_2 \,.\, [\widetilde{x}_1] * \langle \widetilde{x}_1.req, n_1.reply, n_2.reply \rangle * \langle r_1.out, \ell_2.in \rangle$
$\widetilde{z}_1^{(1)} = n_1 * \widetilde{z}_2^{(1)} = \widetilde{x}_2 * \widetilde{z}_3^{(1)} = r_1 * \widetilde{z}_1^{(2)} = n_2 * \widetilde{z}_2^{(2)} = \ell_2 * \widetilde{z}_3^{(2)} = \widetilde{x}_3$
$\exists n_1 \exists r_1 \exists n_2 \exists \ell_2 \,.\, [\widetilde{x}_1] * \langle \widetilde{x}_1.req, n_1.reply, n_2.reply \rangle * \langle r_1.out, \ell_2.in \rangle$
$\widetilde{z}_1^{(1)} = n_1 * \widetilde{z}_2^{(1)} = \widetilde{x}_2 * \widetilde{z}_3^{(1)} = r_1 * \widetilde{z}_1^{(2)} = n_2 * \widetilde{z}_2^{(2)} = \ell_2 * \widetilde{z}_3^{(2)} = \widetilde{x}_3$
$\exists n_1 \exists r_1 \exists n_2 \exists \ell_2 \,.\, [\widetilde{x}_1] * \langle \widetilde{x}_1.req, n_1.reply, n_2.reply \rangle * \langle r_1.out, \ell_2.in \rangle$
$\widetilde{z}_1^{(1)} = n_1 * \widetilde{z}_2^{(1)} = \widetilde{x}_2 * \widetilde{z}_3^{(1)} = r_1 * \widetilde{z}_1^{(2)} = n_2 * \widetilde{z}_2^{(2)} = \ell_2 * \widetilde{z}_3^{(2)} = \widetilde{x}_3$
β
β
β
β
$[\widetilde{x}_1]@q_1$
$[\widetilde{x}_1]@q_0$
$[\widetilde{x}_1]@q_0$
$[\widetilde{x}_1]@q_0$
$[\widetilde{x}_1]@q_0$
$[\widetilde{x}_1]@q_0$
$[\widetilde{x}_1]@q_0$
$[\widetilde{x}_1]@q_0$

Havoc Action as Tree Transductions

- Non-deterministically choses which interaction <$x_1.p_1$ … $x_n.p_n$> is triggered
- Tracks each variable x_i to the atom [x]@q that instantiates it (creates the respective node)
- Change the states of these nodes according to the transitions of the behavior (state machine)


(α,α)
$\widetilde{b}_2 = \widetilde{e}_2 * \widetilde{b}_1 = \widetilde{e}_1$
(β,β)
$\widetilde{b}_2 = \widetilde{x}_2 * \widetilde{b}_1 = \widetilde{x}_3$
(β,β)
$\widetilde{b}_2 = \widetilde{x}_2$
(β,β)
$\widetilde{b}_1 = \widetilde{x}_3$
(β,β)
$\widetilde{b}_2 = \widetilde{x}_2$
(β,β)
emp
(β,β)
emp
(β,β)
$\widetilde{b}_1 = \widetilde{x}_3$
(γ_1,γ_0)
$\widetilde{b}_2 = \widetilde{x}_1$
(γ_0,γ_0)
emp
(γ_0,γ_0)
emp
(γ_0,γ_0)
emp
(γ_0,γ_0)
emp
(γ_0,γ_0)
emp
(γ_0,γ_0)
emp
(γ_0,γ_1)
$\widetilde{b}_1 = \widetilde{x}_1$

End of Part I

A simplified model of dynamic reconfigurable systems

- components with finite-state behavior and interactions of finite arity
- a sequential programming language for describing reconfiguration

A resource logic for describing possibly infinite sets of configurations

- inductively defined predicates

A proof system for reconfiguration programs

- uses local reasoning to a maximum extent
- generates external proof obligations (entailments)

Entailment Checking Between Inductive Sets of Configurations

Key to mechanising proof generation for reconfiguration programs

- checking havoc invariance requires entailment checking
- entailments is needed when applying the standard consequence rule of Hoare logic
- solving frame inference (conditional rule) uses similar techniques

Entailment Checking Between Inductive Sets of Configurations

Key to mechanising proof generation for reconfiguration programs

- checking havoc invariance requires entailment checking
- entailments is needed when applying the standard consequence rule of Hoare logic
- solving frame inference (conditional rule) uses similar techniques

Entailment of inductively defined predicates is a hard problem [Bozga, Bueri, I IJCAR'22]

- satisfiability is decidable (2EXP∩NP-hard)
- entailment is undecidable in general and decidable under certain restrictions (4EXP∩2EXP-hard)
- we currently try to understand what are the weakest such restrictions

Relational Structures

$\sum = \{\underbrace{R_1, \dots, R_N}_{\text{relation symbols}}, \underbrace{c_1, \dots, c_M}_{\text{constants}}\}$ relational signature

$S = (\underbrace{U}_{\text{universe}}, \underbrace{\sigma}_{\text{interpretation of symbols from } \sum})$ structure

Relational Structures

$\sum = \{\underbrace{R_1, \ldots, R_N}_{\text{relation symbols}}, \underbrace{c_1, \ldots., c_M}_{\text{constants}}\}$ relational signature

$S = (\underbrace{U}_{\text{universe}}, \underbrace{\sigma}_{\text{interpretation of symbols from } \sum})$ structure

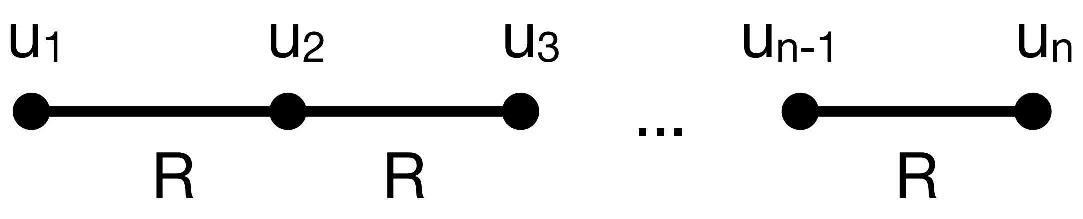
The tree-width is an integer that measures how close a structure (graph) is to a tree

Relational Structures

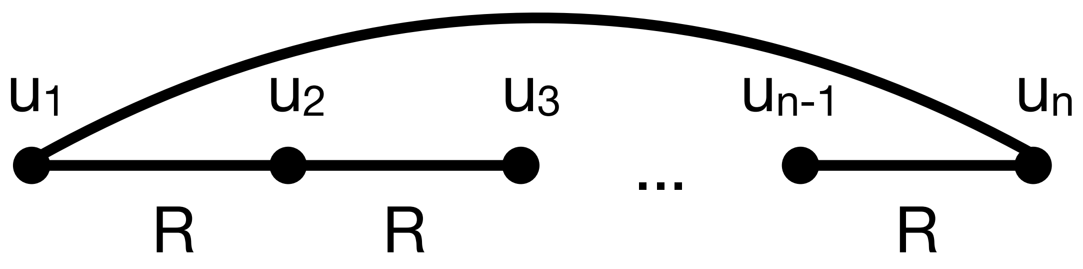
$\sum = \{\underbrace{R_1, \ldots, R_N}_{\text{relation symbols}}, \underbrace{c_1, \ldots., c_M}_{\text{constants}}\}$ relational signature

$S = (\underbrace{U}_{\text{universe}}, \underbrace{\sigma}_{\text{interpretation of symbols from } \sum})$ structure

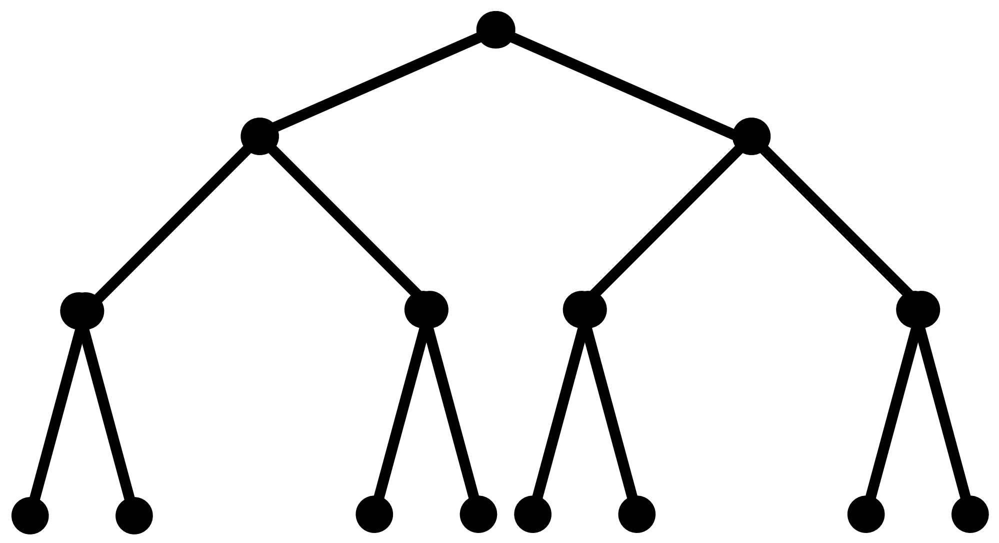
The tree-width is an integer that measures how close a structure (graph) is to a tree



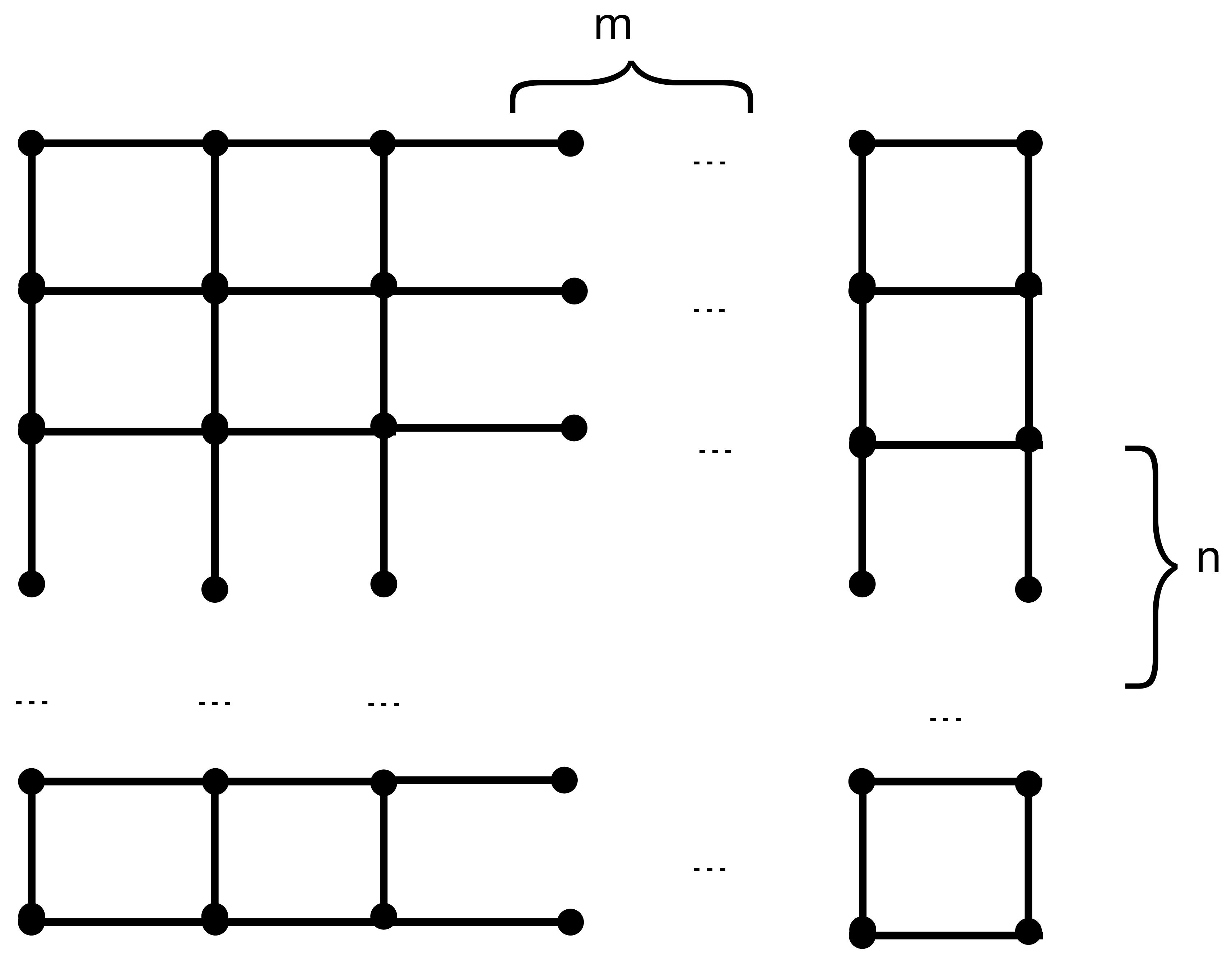
tree-width = 1



tree-width = 2



tree-width = 1



tree-width = min(n, m)

Separation Logic of Relations (SLR)

$\sum = \{\underbrace{R_1, ..., R_N}_{\text{relation symbols}}, \underbrace{c_1, ..., c_M}_{\text{constants}}\}$ relational signature

$S = (\underbrace{U}_{\text{universe}}, \underbrace{\sigma}_{\text{interpretation of symbols from } \sum})$ structure

Separation Logic of Relations (SLR)

$\sum = \{\underbrace{R_1, ..., R_N}_{\text{relation symbols}}, \underbrace{c_1, ..., c_M}_{\text{constants}}\}$ relational signature

$S = (\underbrace{U}_{\text{universe}}, \underbrace{\sigma}_{\text{interpretation of symbols from } \sum})$ structure

emp any structure with empty interpretation

Separation Logic of Relations (SLR)

$\sum = \{\underbrace{R_1, ..., R_N}_{\text{relation symbols}}, \underbrace{c_1, ..., c_M}_{\text{constants}}\}$ relational signature

$S = (\underbrace{U}_{\text{universe}}, \underbrace{\sigma}_{\text{interpretation of symbols from } \sum})$ structure

emp any structure with empty interpretation

$R(x_1, \ldots, x_n)$ all relations except R empty and R contains the tuple of values $x_1, \ldots, x_n$

Separation Logic of Relations (SLR)

$\sum = \{\underbrace{R_1, \ldots, R_N}_{\text{relation symbols}}, \underbrace{c_1, \ldots., c_M}_{\text{constants}}\}$ relational signature

$S = (\underbrace{U}_{\text{universe}}, \underbrace{\sigma}_{\text{interpretation of symbols from } \sum})$ structure

emp — any structure with empty interpretation

$R(x_1, \ldots, x_n)$ — all relations except R empty and R contains the tuple of values $x_1, \ldots, x_n$

$\phi_1 * \phi_2$ — any structure $S_1 \otimes S_2$, such that $S_i \models \phi_i$, for all $i=1,2$

- $(U_1,\sigma_1) \otimes (U_2,\sigma_2) = (U_1 \cup U_2, \sigma_1 \uplus \sigma_2)$
- $\sigma_1 \uplus \sigma_2$ is the point-wise disjoint union of interpretations

Separation Logic of Relations (SLR)

$\sum = \{\underbrace{R_1, ..., R_N}_{\text{relation symbols}}, \underbrace{c_1, ..., c_M}_{\text{constants}}\}$ relational signature

$S = (\underbrace{U}_{\text{universe}}, \underbrace{\sigma}_{\text{interpretation of symbols from } \sum})$ structure

emp — any structure with empty interpretation

$R(x_1, \ldots, x_n)$ — all relations except R empty and R contains the tuple of values $x_1, \ldots, x_n$

$\phi_1 * \phi_2$ — any structure $S_1 \otimes S_2$, such that $S_i \models \phi_i$, for all $i=1,2$

- $(U_1,\sigma_1) \otimes (U_2,\sigma_2) = (U_1 \cup U_2, \sigma_1 \uplus \sigma_2)$
- $\sigma_1 \uplus \sigma_2$ is the point-wise disjoint union of interpretations

$R_1(y_1, \ldots, y_n) * R_1(z_1, \ldots, z_n)$ implies $y_i \neq z_i$, for at least one $i=1, \ldots, n$

(Monadic) Second Order Logic

$\sum = \{\underbrace{R_1, \dots, R_N}_{\text{relation symbols}}, \underbrace{c_1, \dots., c_M}_{\text{constants}}\}$ relational signature

$S = (\underbrace{U}_{\text{universe}}, \underbrace{\sigma}_{\text{interpretation of symbols from } \sum})$ structure

$R(x_1, \dots, x_n)$	R contains the tuple of values $x_1, \dots, x_n$, ‣ the rest of the structure remains unspecified
$\exists x.\phi(x)$	quantification over individual elements of U
$\exists X.\phi(X)$	quantification over relations, i.e., subsets of $\underbrace{\mathrm{U\ x\ \dots\ x\ U}}_{\#(X)}$
$\neg\phi,\ \phi_1 \wedge \phi_2$	boolean connectives

(Monadic) Second Order Logic

$\sum = \{\underbrace{R_1, \ldots, R_N}_{\text{relation symbols}}, \underbrace{c_1, \ldots., c_M}_{\text{constants}}\}$ relational signature

$S = (\underbrace{U}_{\text{universe}}, \underbrace{\sigma}_{\text{interpretation of symbols from } \sum})$ structure

$R(x_1, \ldots, x_n)$ R contains the tuple of values $x_1, \ldots, x_n$,

- the rest of the structure remains unspecified

$\exists x.\phi(x)$ quantification over individual elements of U

$\exists X.\phi(X)$ quantification over relations, i.e., subsets of $U \underbrace{x \ldots x}_{\#(X)} U$

$\neg\phi$, $\phi_1 \wedge \phi_2$ boolean connectives

MSO is the fragment of SO where #(X)=1 for all relation variables

(Monadic) Second Order Logic

$\sum = \{\underbrace{R_1, \ldots, R_N}_{\text{relation symbols}}, \underbrace{c_1, \ldots., c_M}_{\text{constants}}\}$ relational signature

$S = (\underbrace{U}_{\text{universe}}, \underbrace{\sigma}_{\text{interpretation of symbols from } \sum})$ structure

$R(x_1, \ldots, x_n)$ R contains the tuple of values $x_1, \ldots, x_n$,

- the rest of the structure remains unspecified

$\exists x.\phi(x)$ quantification over individual elements of U

$\exists X.\phi(X)$ quantification over relations, i.e., subsets of $\underbrace{U \times \ldots \times U}_{\#(X)}$

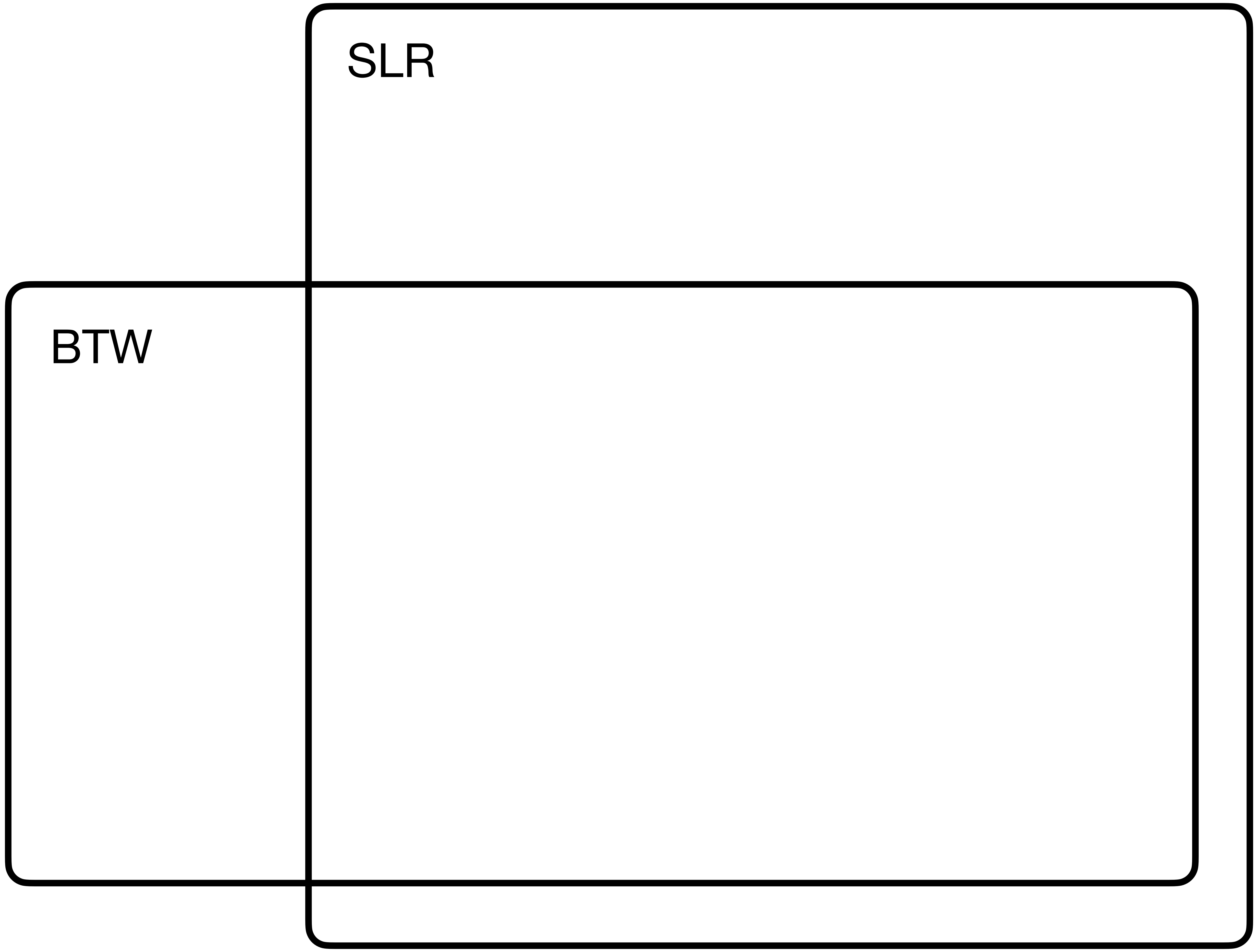
$\neg\phi$, $\phi_1 \wedge \phi_2$ boolean connectives

MSO is the fragment of SO where #(X)=1 for all relation variables

MSO is the yardstick of graph description logics:

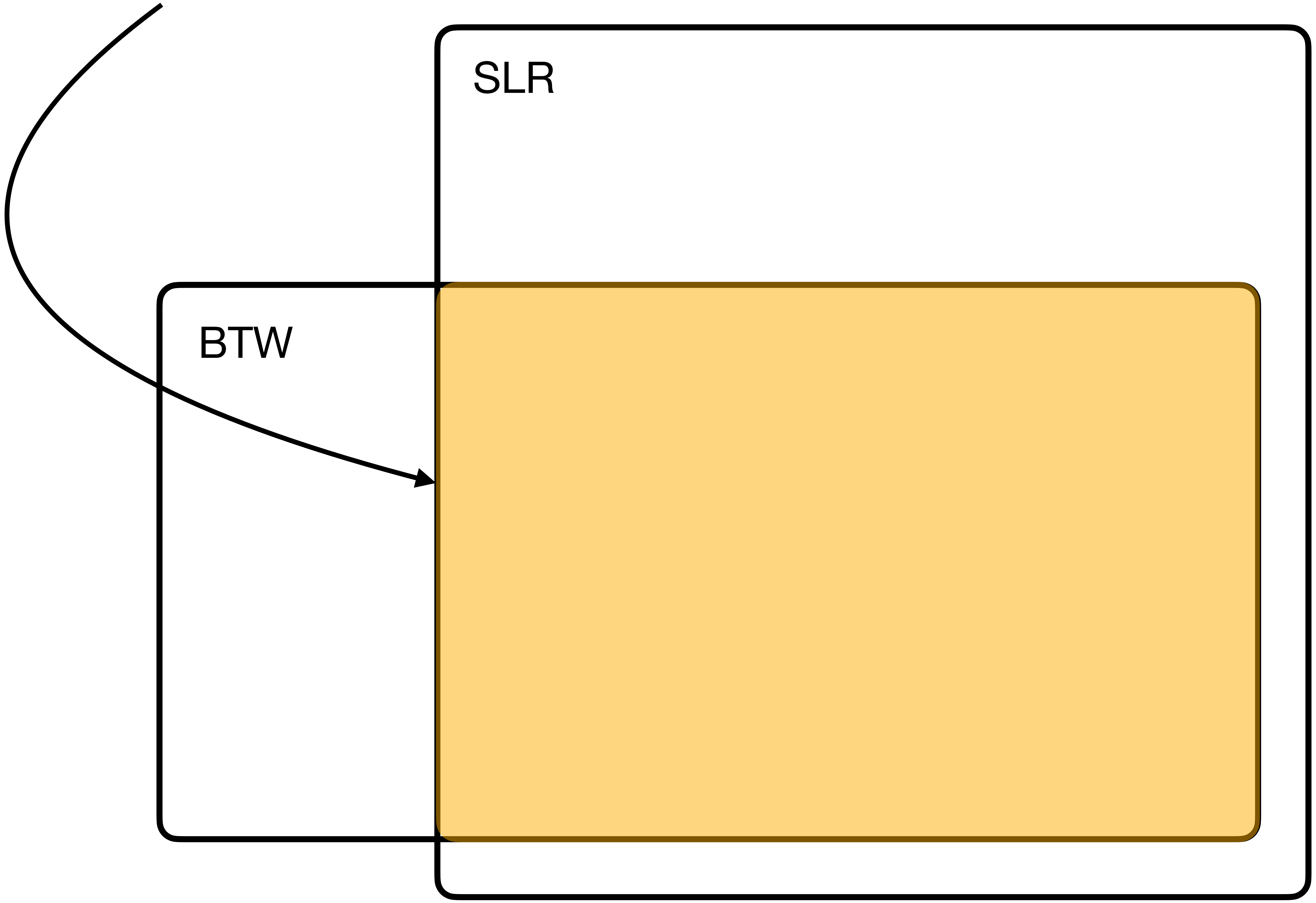
- Decidable for structures of bounded tree-width [Courcelle'90]
- Each class of structures with a decidable MSO theory has bounded tree-width [Seese'91]

The Big Picture



The Big Picture

A decidable characterization
[Bozga, Bueri, I, Zuleger ARXIV 2023a]



Canonical Models

ls(x,y) ← ∃z . R(x,z) * ls(z,y)

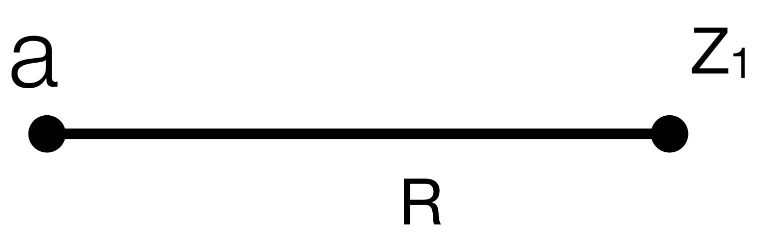
ls(x,y) ← emp * x=y

Canonical Models

$ls(x,y) \leftarrow \exists z \,.\, R(x,z) * ls(z,y)$

$ls(x,y) \leftarrow emp * x=y$

$ls(a,b) \Rightarrow \exists z_1 \,.\, R(a,z_1) * ls(z_1,b)$

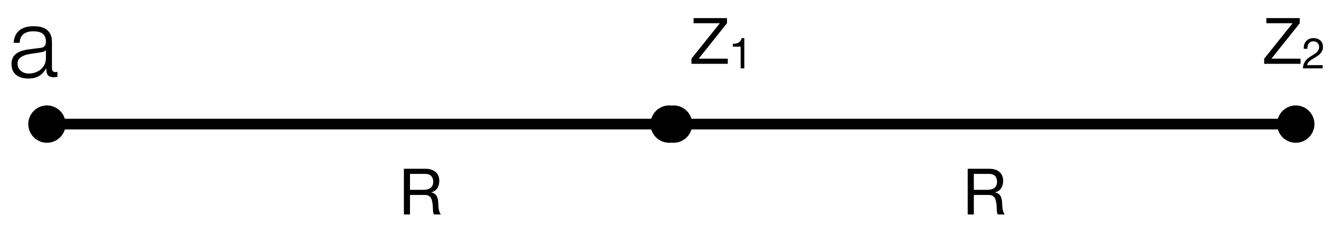


Canonical Models

$ls(x,y) \leftarrow \exists z \,.\, R(x,z) * ls(z,y)$

$ls(x,y) \leftarrow emp * x=y$

$ls(a,b) \Rightarrow \exists z_1 \,.\, R(a,z_1) * ls(z_1,b) \;\; \Rightarrow \exists z_1 \; \exists z_2 \,.\, R(a,z_1) * R(z_1,z_2) * ls(z_2,b)$

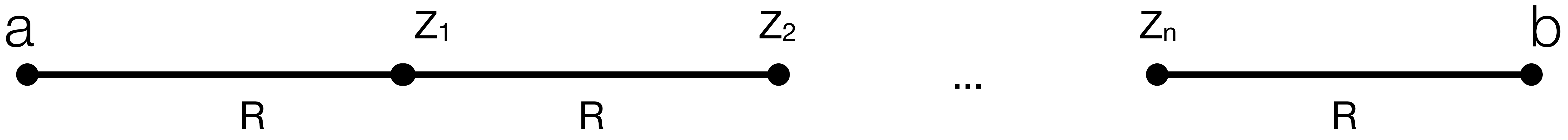


Canonical Models

$ls(x,y) \leftarrow \exists z \,.\, R(x,z) * ls(z,y)$

$ls(x,y) \leftarrow emp * x=y$

$ls(a,b) \Rightarrow \exists z_1 \,.\, R(a,z_1) * ls(z_1,b) \Rightarrow \exists z_1\, \exists z_2 \,.\, R(a,z_1) * R(z_1,z_2) * ls(z_2,b) \Rightarrow \ldots \Rightarrow \exists z_1\, \exists z_2 \ldots \exists z_n \,.\, R(a,z_1) * R(z_1,z_2) * \ldots * R(z_n,b)$

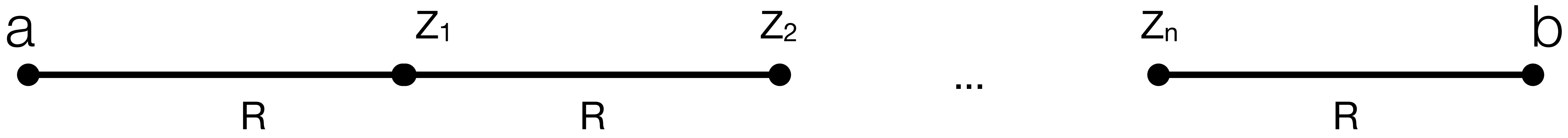


Canonical Models

$ls(x,y) \leftarrow \exists z \,.\, R(x,z) * ls(z,y)$

$ls(x,y) \leftarrow emp * x{=}y$

$ls(a,b) \Rightarrow \exists z_1 \,.\, R(a,z_1) * ls(z_1,b) \;\; \Rightarrow \exists z_1\, \exists z_2 \,.\, R(a,z_1) * R(z_1,z_2) * ls(z_2,b) \Rightarrow \ldots \Rightarrow \exists z_1\, \exists z_2 \ldots \exists z_n \,.\, R(a,z_1) * R(z_1,z_2) * \ldots * R(z_n,b)$



Existentially quantified variables introduced by the unfolding are instantiated by distinct elements

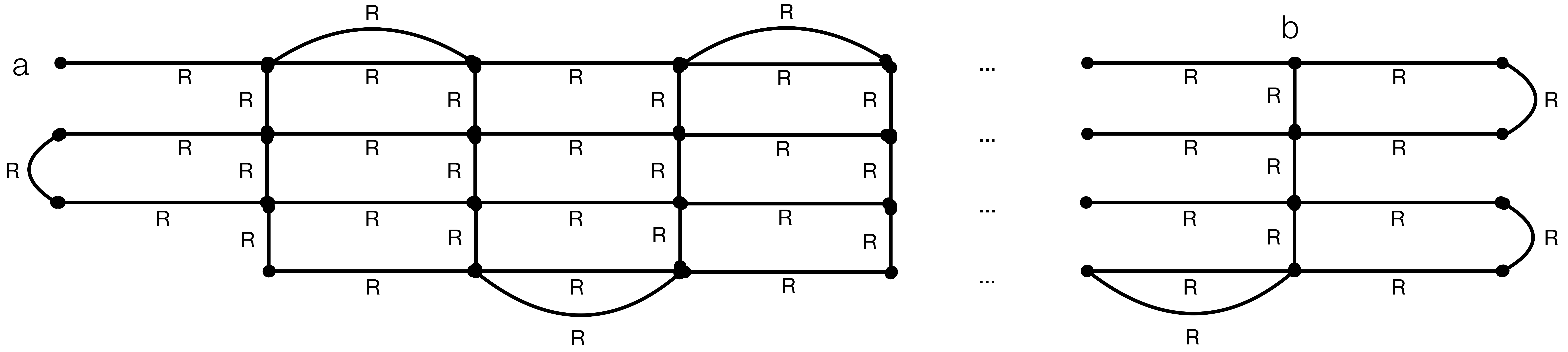
- there exists a uniform bound on the tree-width of canonical models
- the maximal number of variables that occur (free or bound) in an inductive definition

Canonical Models

ls(x,y) ← ∃z . R(x,z) * ls(z,y)

ls(x,y) ← emp * x=y

ls(a,b) ⇒ ∃z$_1$. R(a,z$_1$) * ls(z$_1$,b) ⇒ ∃z$_1$ ∃z$_2$. R(a,z$_1$) * R(z$_1$,z$_2$) * ls(z$_2$,b) ⇒ ... ⇒ ∃z$_1$ ∃z$_2$ ... ∃z$_n$. R(a,z$_1$) * R(z$_1$,z$_2$) * ... * R(z$_n$,b)

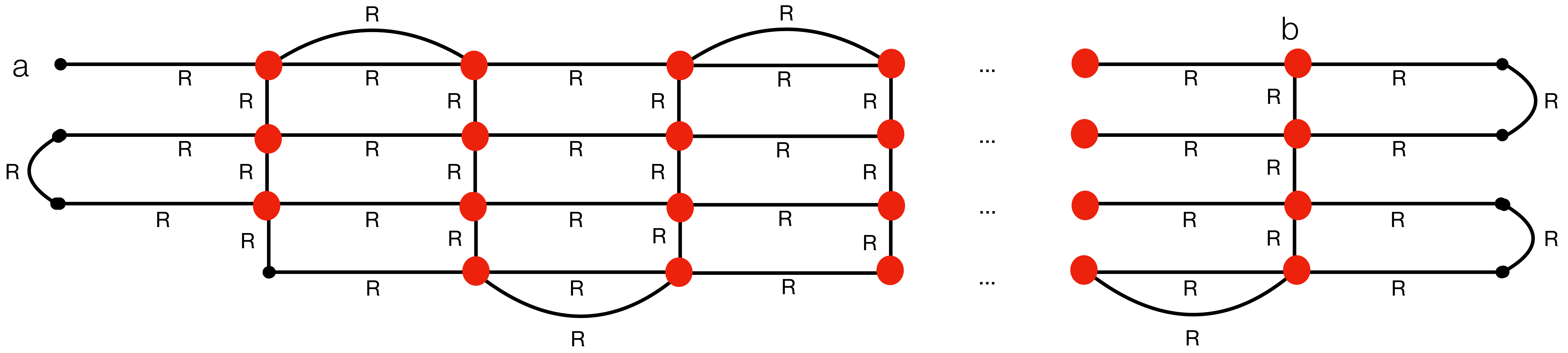


Canonical Models

ls(x,y) ← ∃z . R(x,z) * ls(z,y)

ls(x,y) ← emp * x=y

ls(a,b) ⇒ ∃z₁ . R(a,z₁) * ls(z₁,b) ⇒ ∃z₁ ∃z₂ . R(a,z₁) * R(z₁,z₂) * ls(z₂,b) ⇒ … ⇒ ∃z₁ ∃z₂ … ∃zₙ . R(a,z₁) * R(z₁,z₂) * … * R(zₙ,b)



Each model is obtained from a canonical model by internal fusion

- produces unbounded tree-width sets of models

Bounding the Tree-Width

$ls(x,y) \leftarrow \exists z \,.\, D(z) * R(x,z) * ls(z,y)$

$ls(x,y) \leftarrow emp * x=y$

Bounding the Tree-Width

$ls(x,y) \leftarrow \exists z \,.\, D(z) * R(x,z) * ls(z,y)$

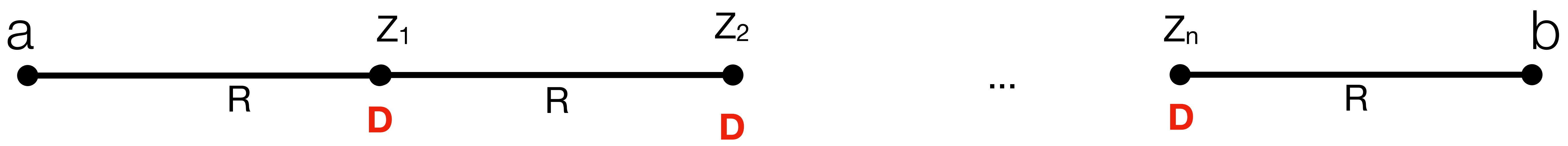
$ls(x,y) \leftarrow emp * x=y$

$ls(a,b) \Rightarrow \exists z_1 \,.\, D(z_1) * R(a,z_1) * ls(z_1,b)$

$\Rightarrow \exists z_1\, \exists z_2 \,.\, D(z_1) * R(a,z_1) * D(z_2) * R(z_1,z_2) * ls(z_2,b)$

...

$\Rightarrow \exists z_1\, \exists z_2 \ldots \exists z_n \,.\, D(z_1) * R(a,z_1) * D(z_2) * R(z_1,z_2) * \ldots * D(z_n) * R(z_n,b)$



Bounding the Tree-Width

$ls(x,y) \leftarrow \exists z \,.\, D(z) * R(x,z) * ls(z,y)$

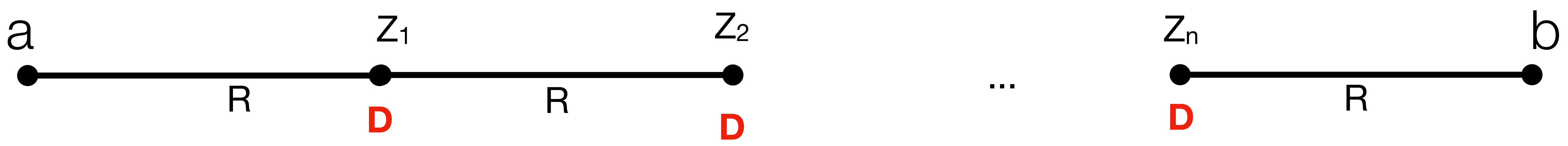
$ls(x,y) \leftarrow emp * x=y$

$ls(a,b) \Rightarrow \exists z_1 \,.\, D(z_1) * R(a,z_1) * ls(z_1,b)$

$\Rightarrow \exists z_1 \exists z_2 \,.\, D(z_1) * R(a,z_1) * D(z_2) * R(z_1,z_2) * ls(z_2,b)$

...

$\Rightarrow \exists z_1 \exists z_2 \ldots \exists z_n \,.\, D(z_1) * R(a,z_1) * D(z_2) * R(z_1,z_2) * \ldots * D(z_n) * R(z_n,b)$



The color of an element = the set of unary relation symbols labeling the element

- only elements with disjoint colors can be fused

Persistent Variables

$\mathrm{ls}(x,y) \leftarrow \exists z \,.\, R(z,y) * R(x,z) * \mathrm{ls}(z,y)$

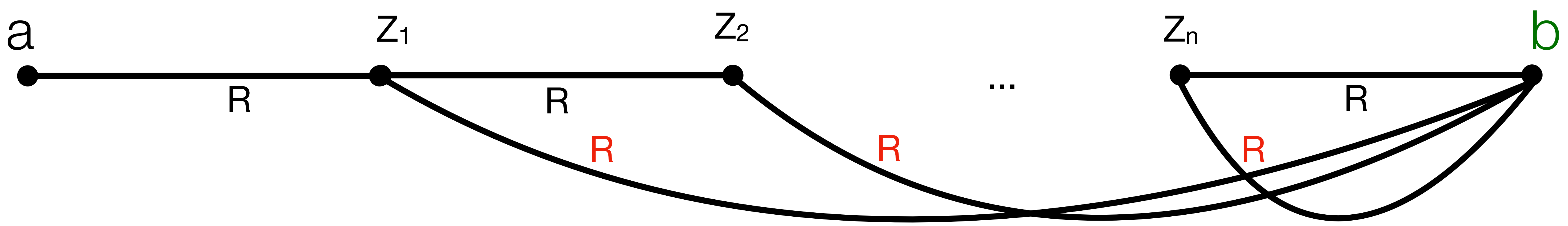
$\mathrm{ls}(x,y) \leftarrow \mathrm{emp} * x=y$

$\mathrm{ls}(a,b) \Rightarrow \exists z_1 \,.\, R(z_1,b) * R(a,z_1) * \mathrm{ls}(z_1,b)$

$\Rightarrow \exists z_1\, \exists z_2 \,.\, R(z_1,b) * R(a,z_1) * R(z_2,b) * R(z_1,z_2) * \mathrm{ls}(z_2,b)$

...

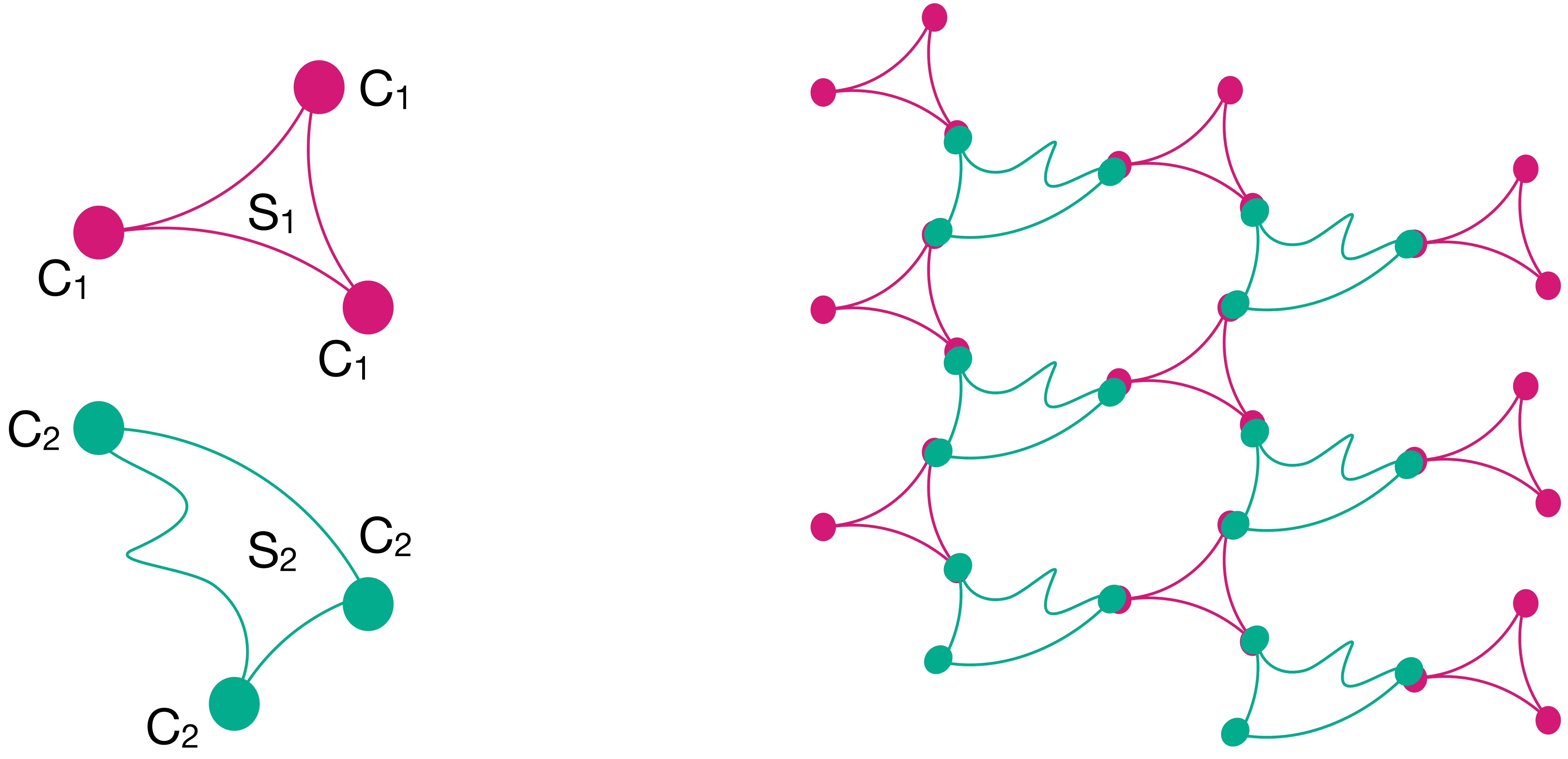
$\Rightarrow \exists z_1\, \exists z_2 \ldots \exists z_n \,.\, R(z_1,b) * R(a,z_1) * R(z_2,b) * R(z_1,z_2) * \ldots * R(z_n,b) * R(z_n,b)$



The color of an element = the set of relation atoms involving only constants besides the element

- persistent variables can be detected by a greatest fixpoint iteration over the set of inductive definitions

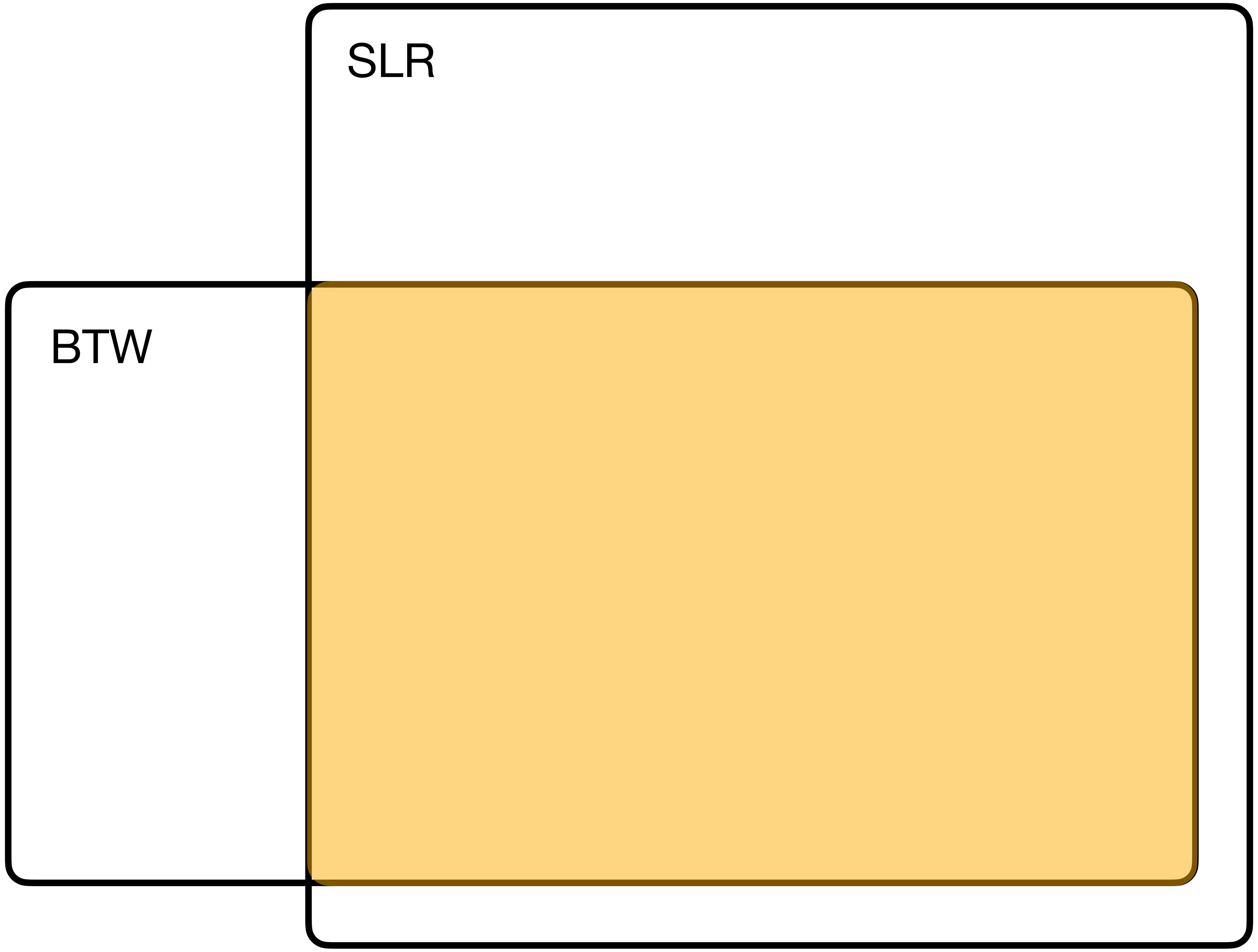
A Decidable Condition



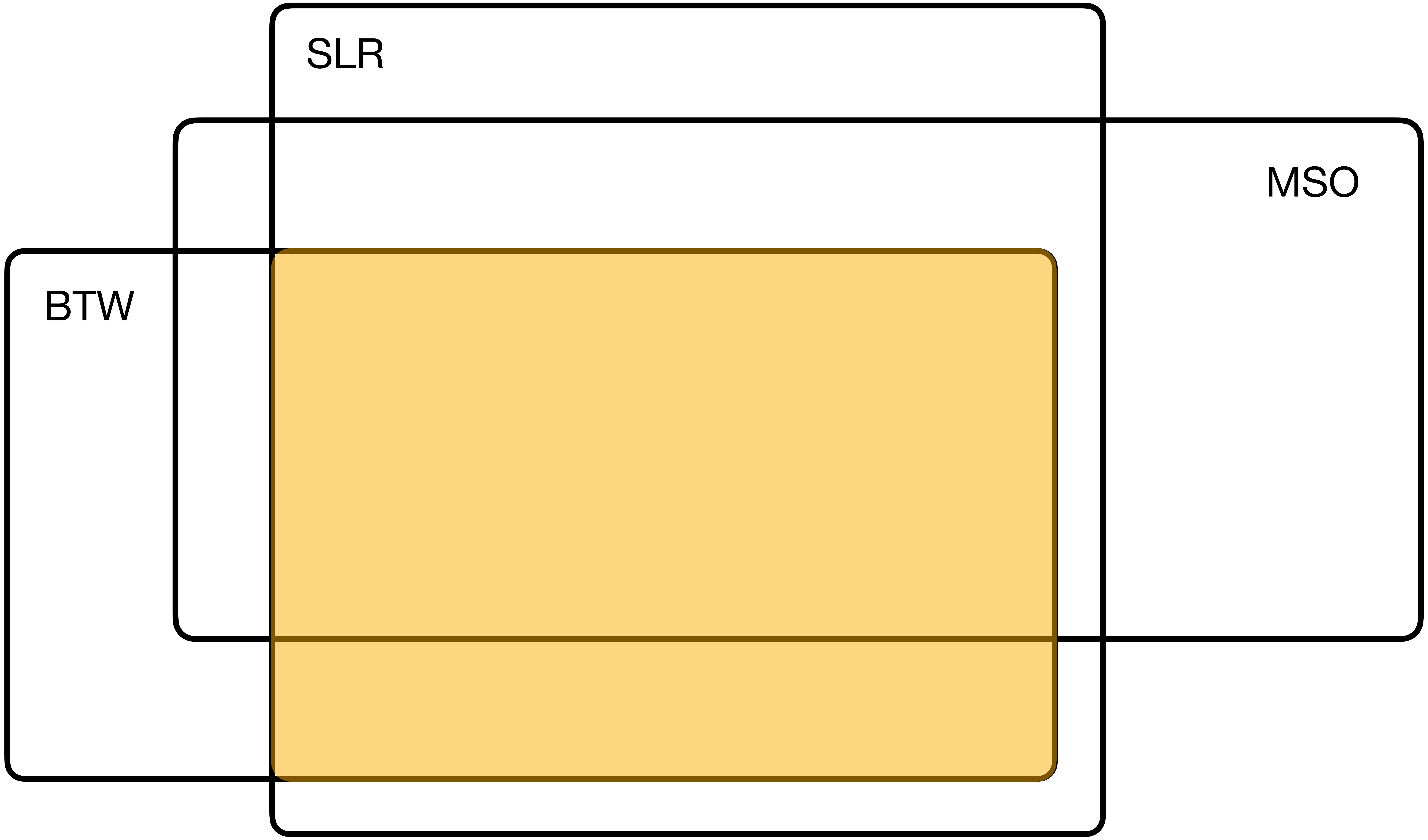
Given an SID Δ, the set of Δ-models of a given sentence ϕ is tree-width unbounded IFF there exist connected structures S_1 and S_2 satisfying the following conditions [Bozga, Bueri, I, Zuleger ARXIV 2023a]:

1. for each $k \geq 1$ there exists $n \geq k$, such that n copies of S_1 and S_2 can be embedded in some Δ-model of ϕ
2. each S_i has at least three occurrences of an element colored C_i, for i = 1,2
3. $C_1 \cap C_2 = \varnothing$

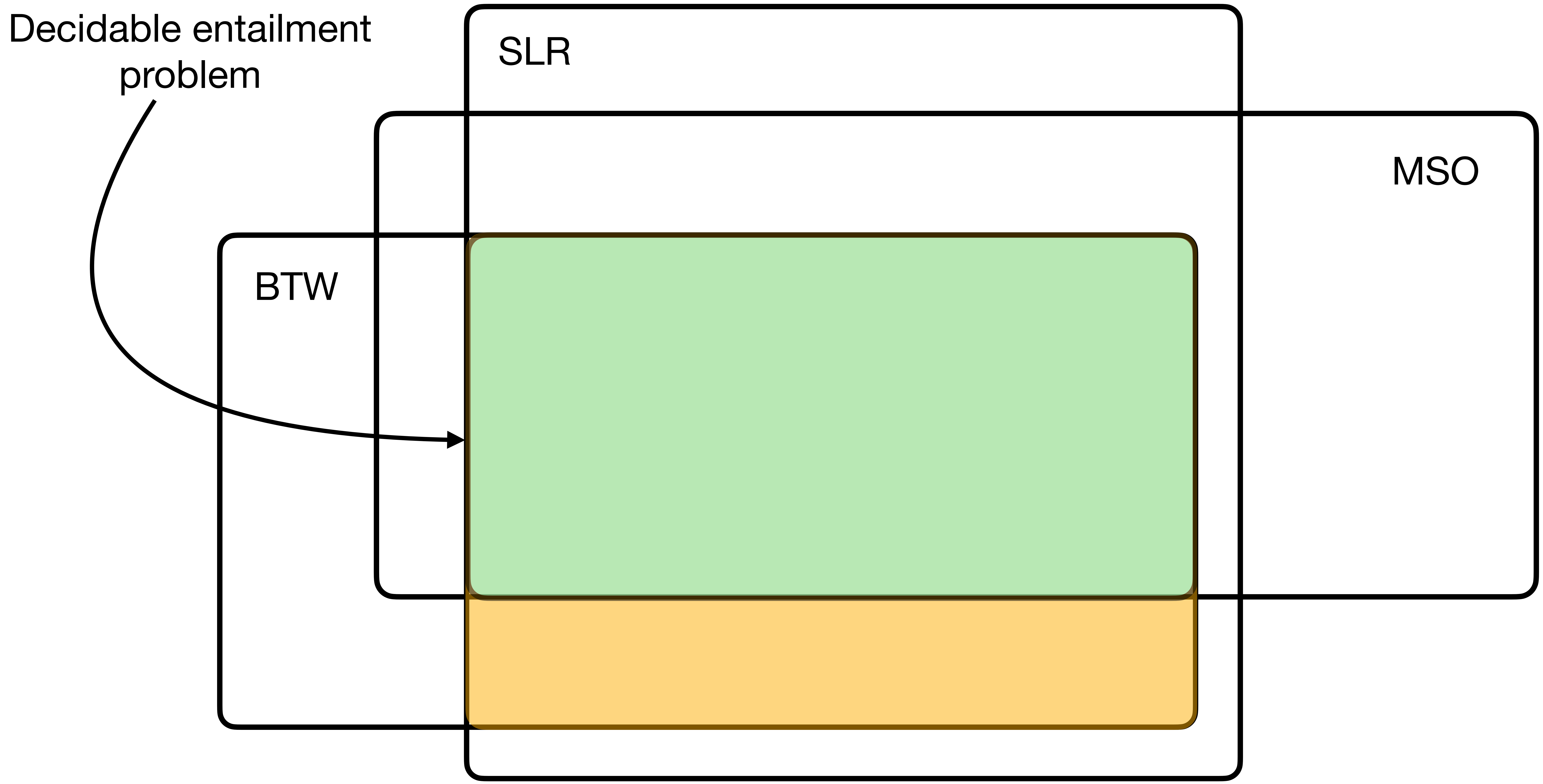
The Big Picture



The Big Picture



The Big Picture



Entailments in MSO ∩ BTW

$$L_1 \overset{?}{\subseteq} L_2$$

Entailments in MSO ∩ BTW

$$\begin{tikzcd} L_1 \arrow[d, dotted] & \overset{?}{\subseteq} & L_2 \arrow[d, dotted] \\ \phi_1 & \overset{?}{\Rightarrow} & \phi_2 \end{tikzcd}$$

Entailments in MSO $\cap$ BTW

$$\begin{tikzcd}
L_1 \arrow[r, phantom, "\overset{?}{\subseteq}"] \arrow[d, dotted] & L_2 \arrow[d, dotted] \\
\phi_1 \arrow[r, phantom, "\overset{?}{\Rightarrow}"] & \phi_2
\end{tikzcd}$$

Is the MSO formula $\phi_1 \wedge \neg\phi_2$ satisfiable ?

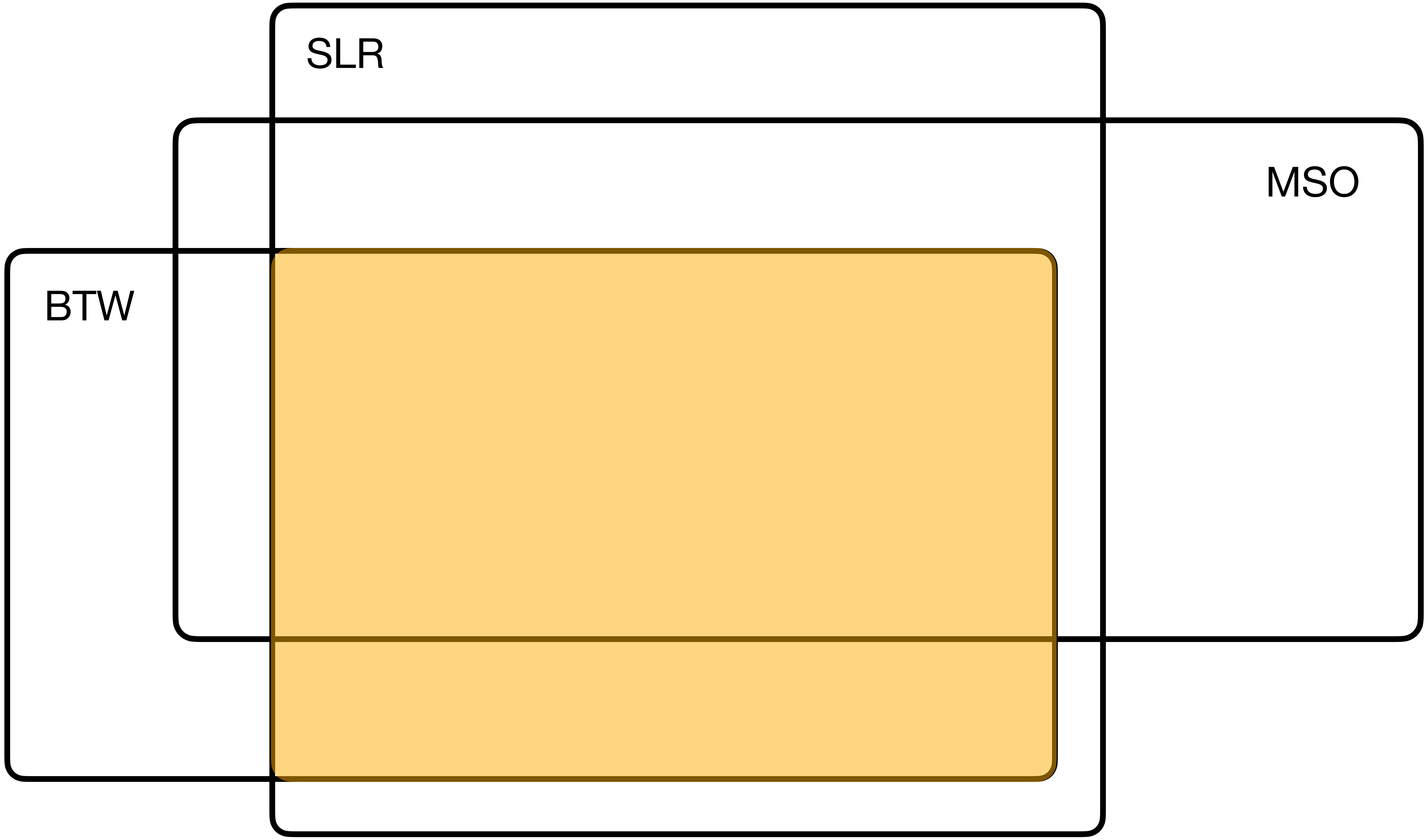
Entailments in MSO ∩ BTW

$$\begin{tikzcd} L_1 \arrow[r, phantom, "\overset{?}{\subseteq}"] \arrow[d, dotted] & L_2 \arrow[d, dotted] \\ \phi_1 \arrow[r, phantom, "\overset{?}{\Rightarrow}"] & \phi_2 \end{tikzcd}$$

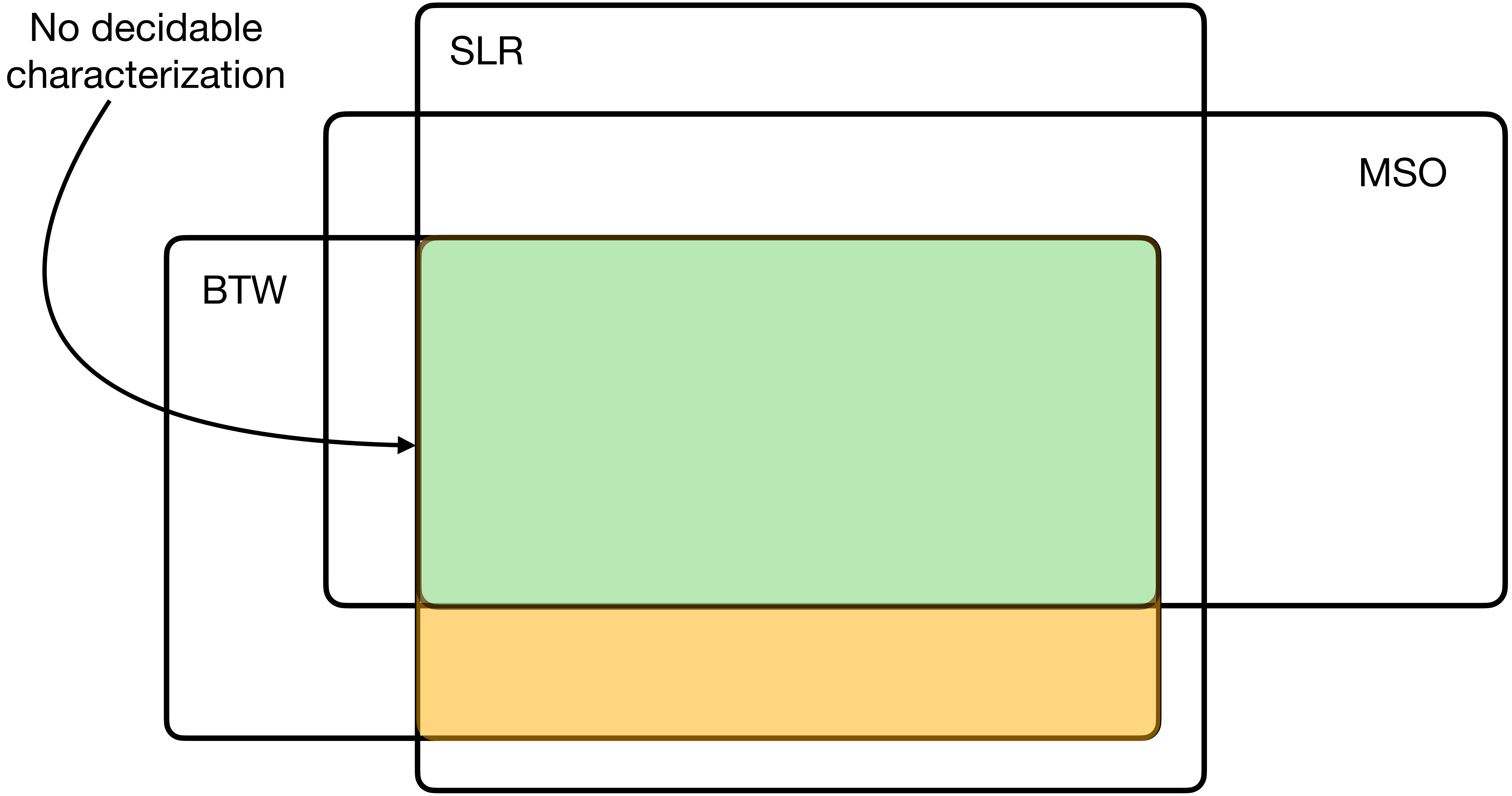
Is the MSO formula $\phi_1 \wedge \neg\phi_2$ satisfiable ?

Satisfiability of a MSO formula is decidable over $\{S \mid \mathit{tree\text{-}width}(S) \leqslant k\}$ [Courcelle'90]

The Big Picture

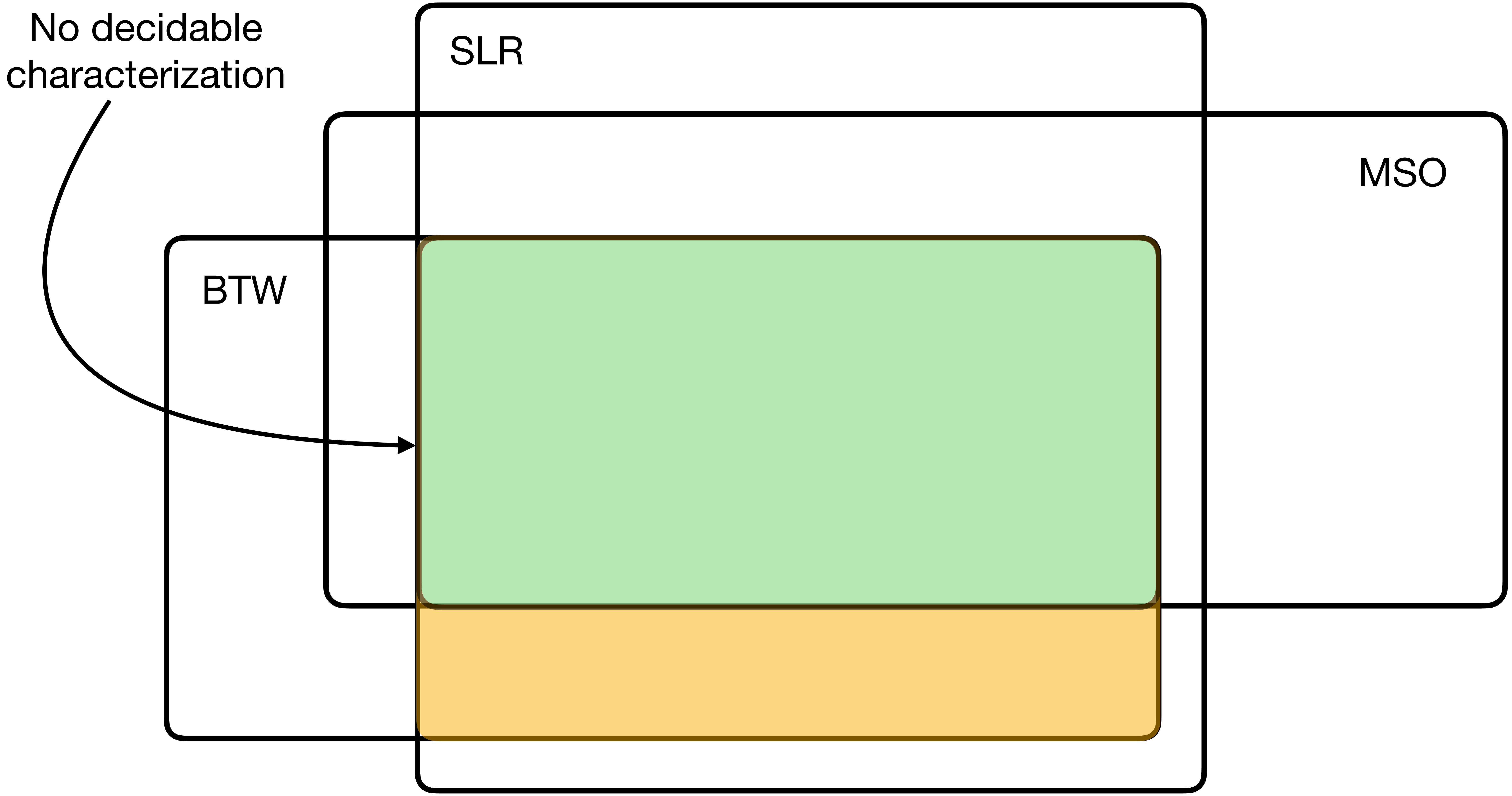


The Big Picture



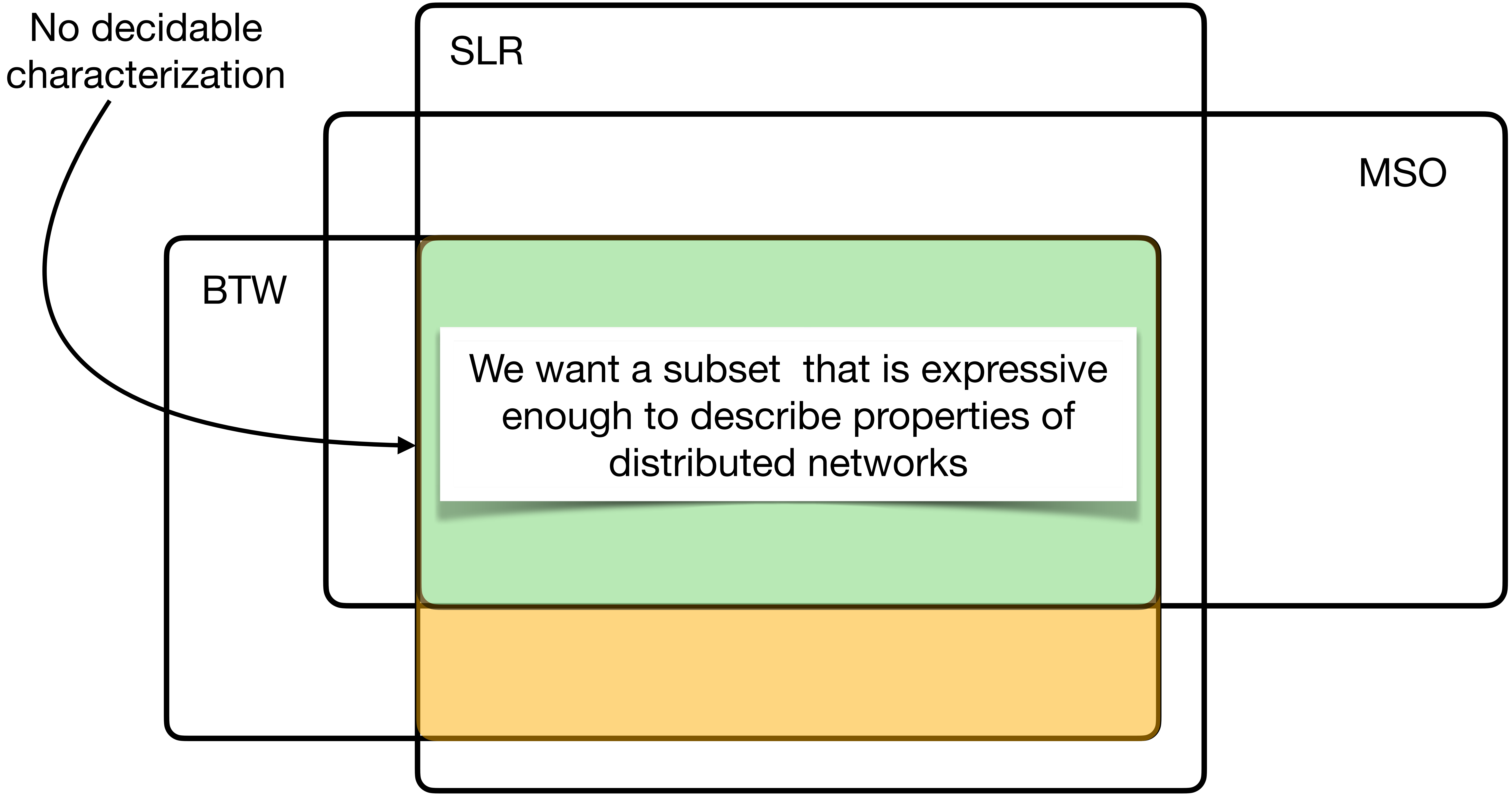
The Big Picture

Given a context-free word language L, the problem “*L is recognizable?”* is undecidable [Greibach’69]



The Big Picture

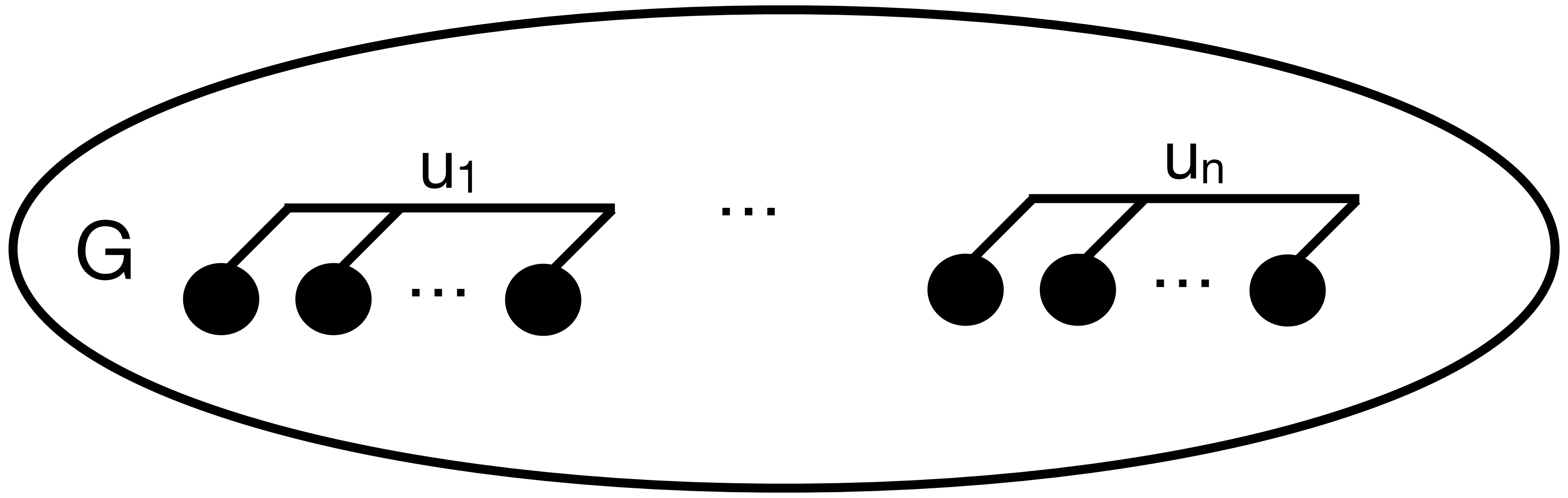
Given a context-free word language L, the problem “*L is recognizable?”* is undecidable [Greibach’69]



We want a subset that is expressive enough to describe properties of distributed networks

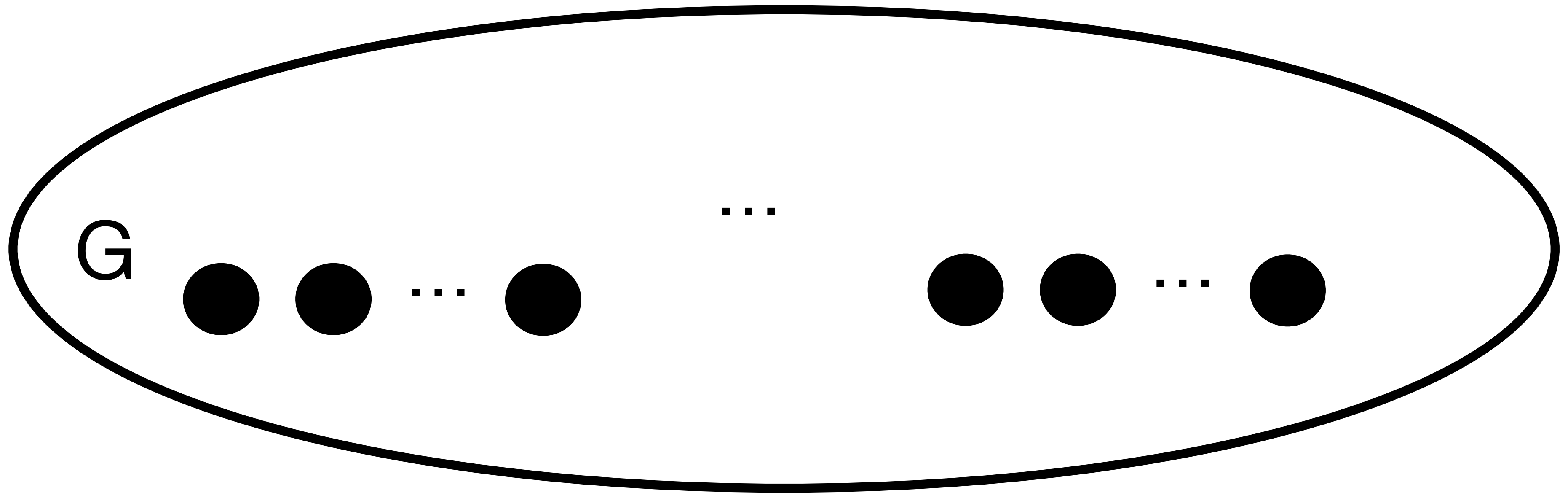
Graph Grammars

Hyperedge-replacement (HR) grammars with operations of the form $(G,u_1,\ldots,u_n)$ and $||_k$



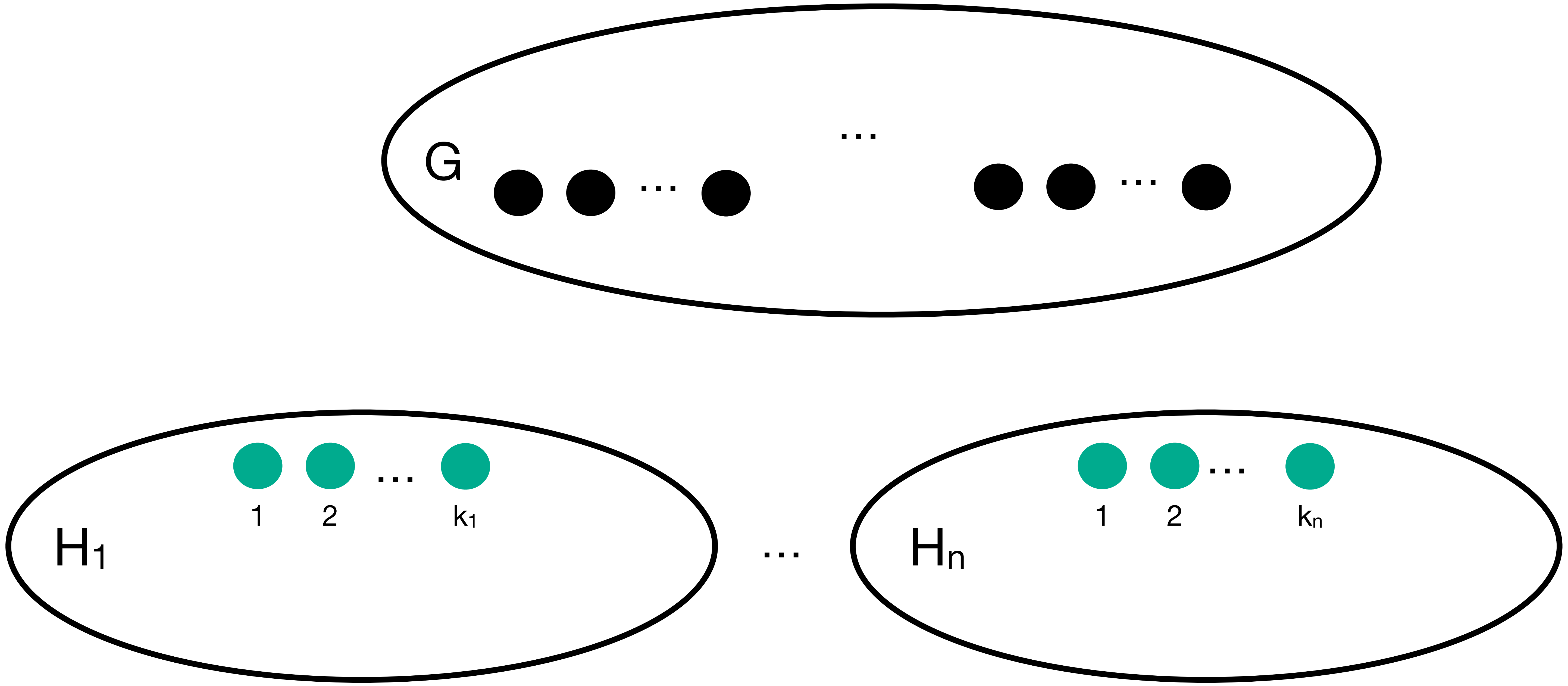
Graph Grammars

Hyperedge-replacement (HR) grammars with operations of the form $(G,u_1,\ldots,u_n)$ and $||_k$



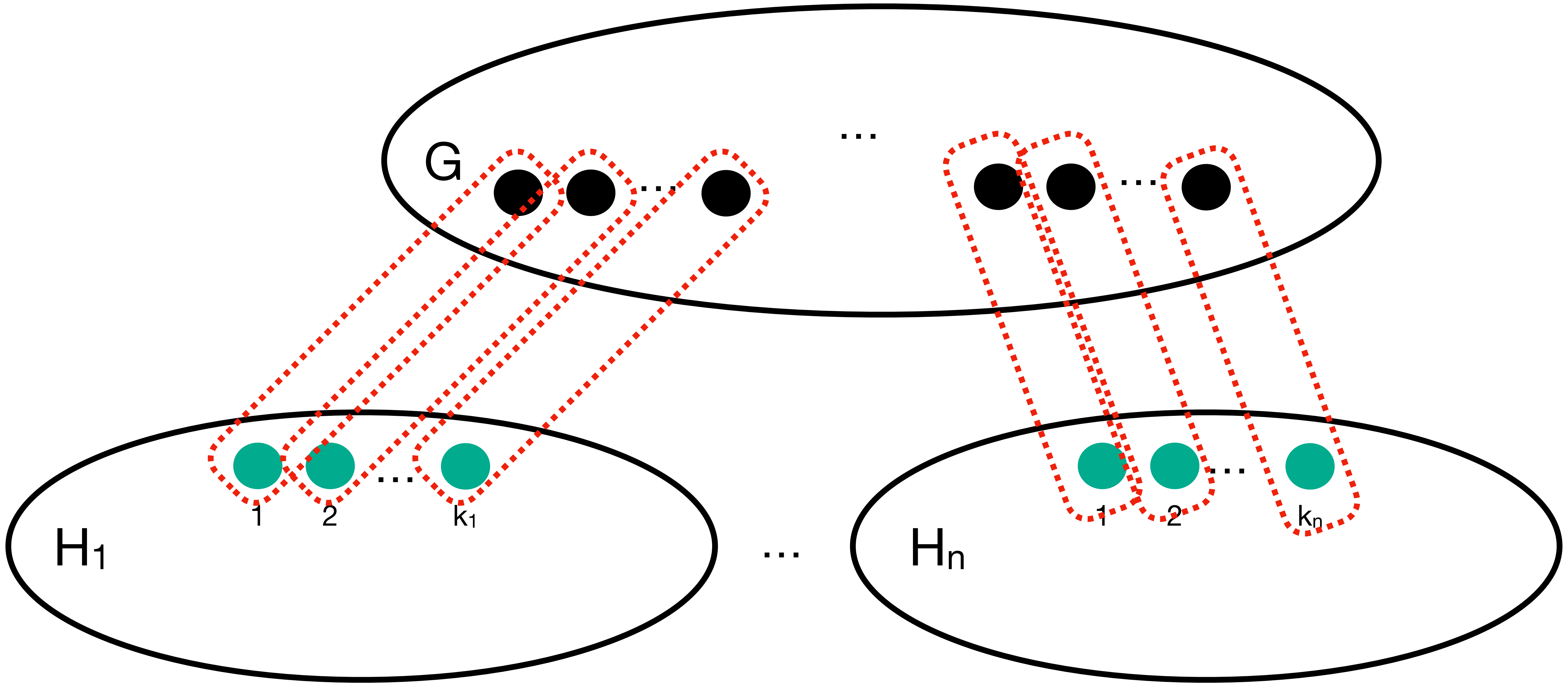
Graph Grammars

Hyperedge-replacement (HR) grammars with operations of the form $(G,u_1,\ldots,u_n)$ and $||_k$



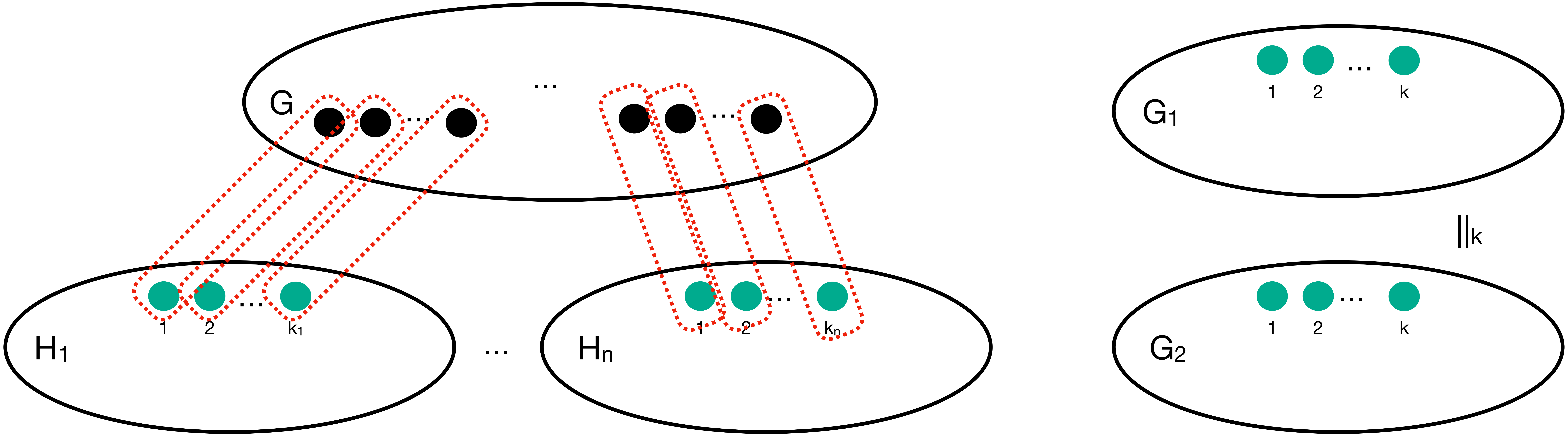
Graph Grammars

Hyperedge-replacement (HR) grammars with operations of the form $(G,u_1,\ldots,u_n)$ and $||_k$



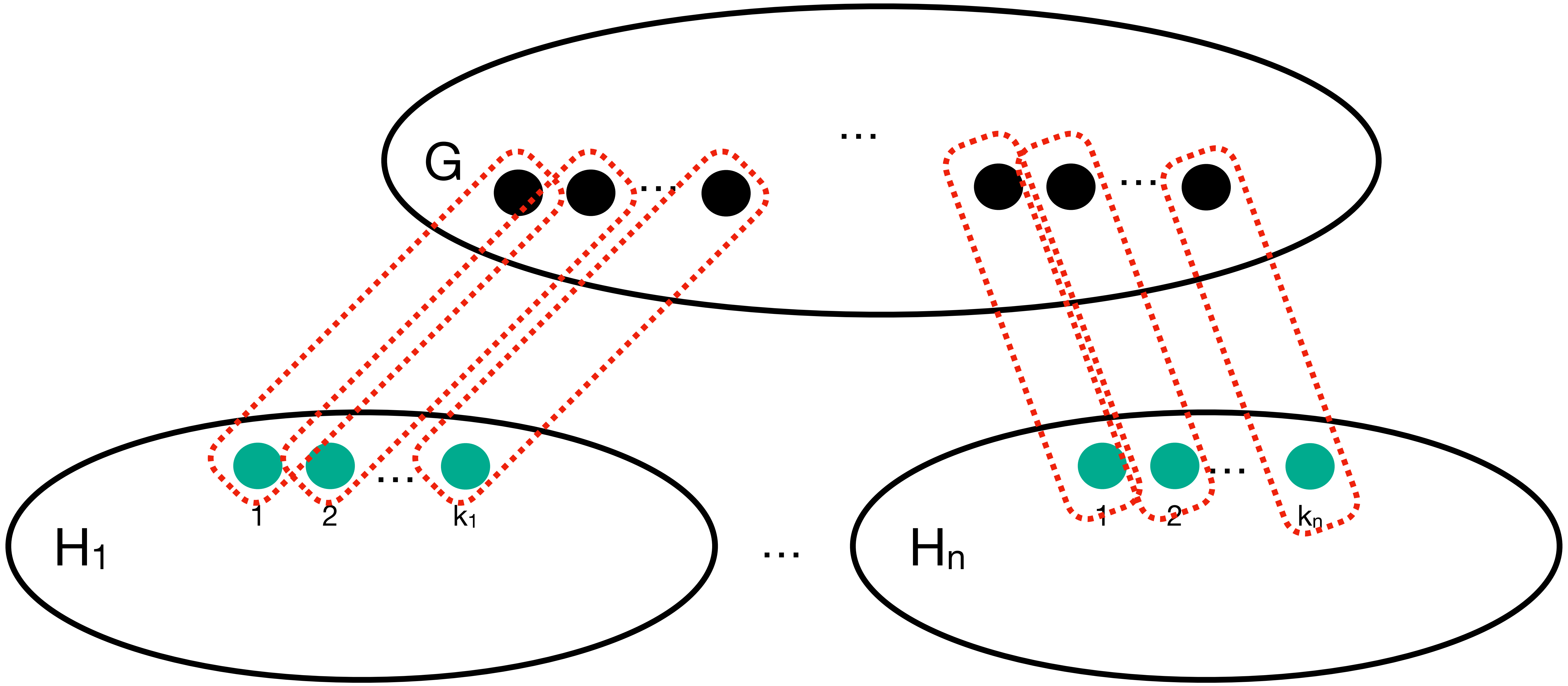
Graph Grammars

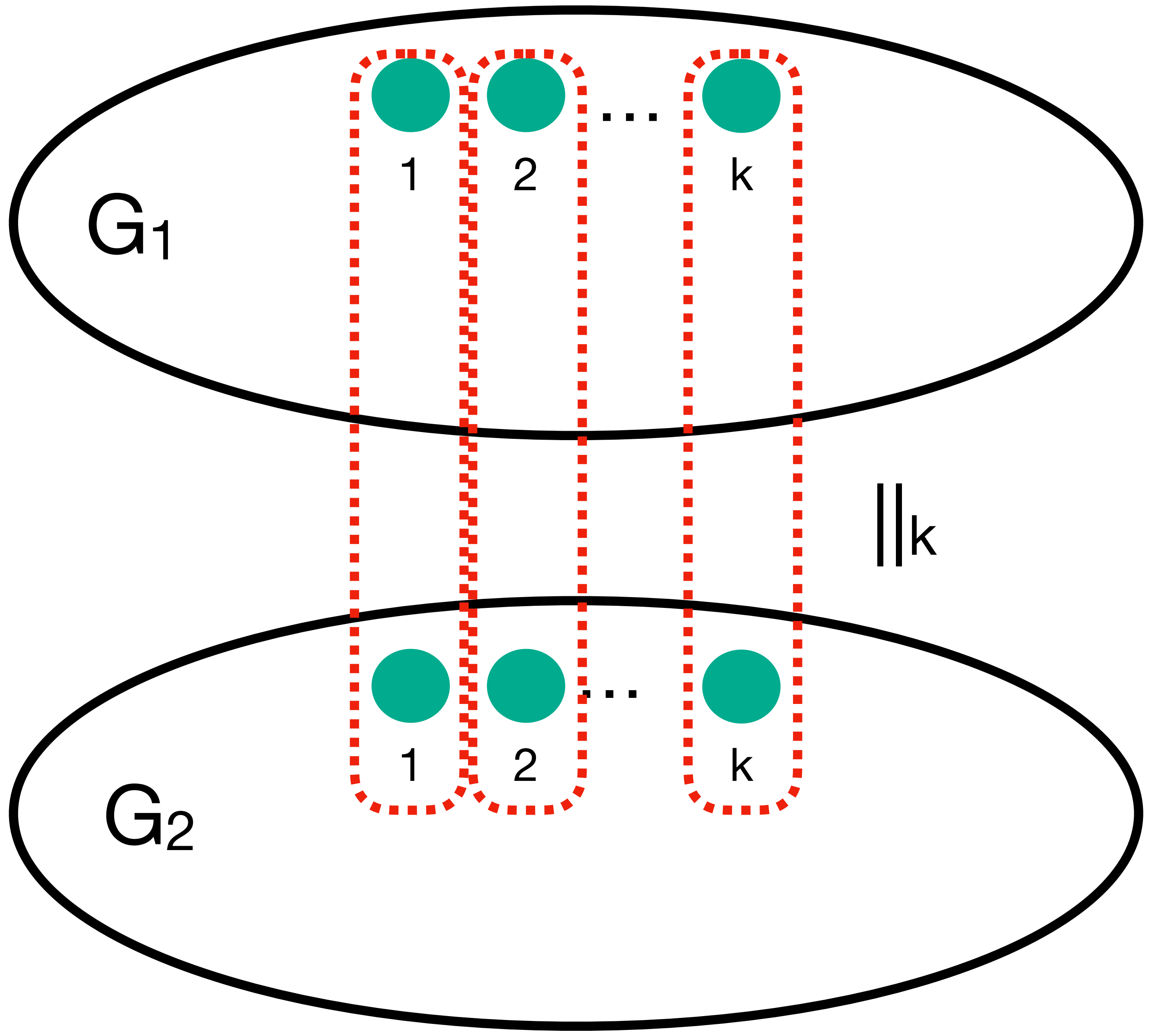
Hyperedge-replacement (HR) grammars with operations of the form $(G,u_1,\ldots,u_n)$ and $\|_k$



Graph Grammars

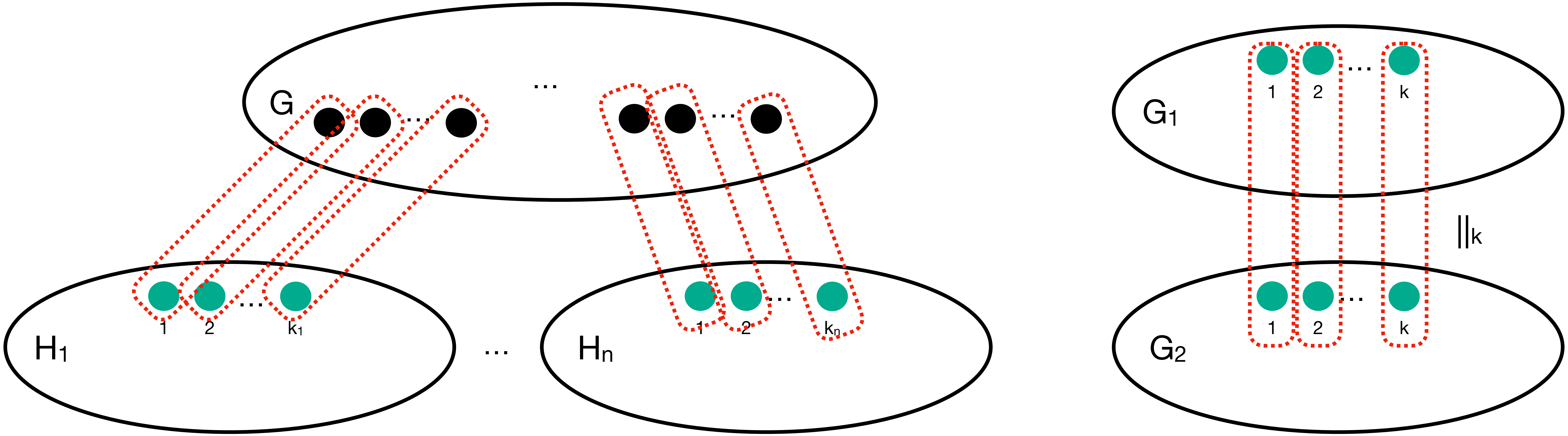
Hyperedge-replacement (HR) grammars with operations of the form $(G, u_1, \ldots, u_n)$ and $\|_k$





Graph Grammars

Hyperedge-replacement (HR) grammars with operations of the form $(G,u_1,\ldots,u_n)$ and $\|_k$



Grammar rules of the form $u \to v \|_k w$ or $u \to (G, v_1, \ldots v_n)$

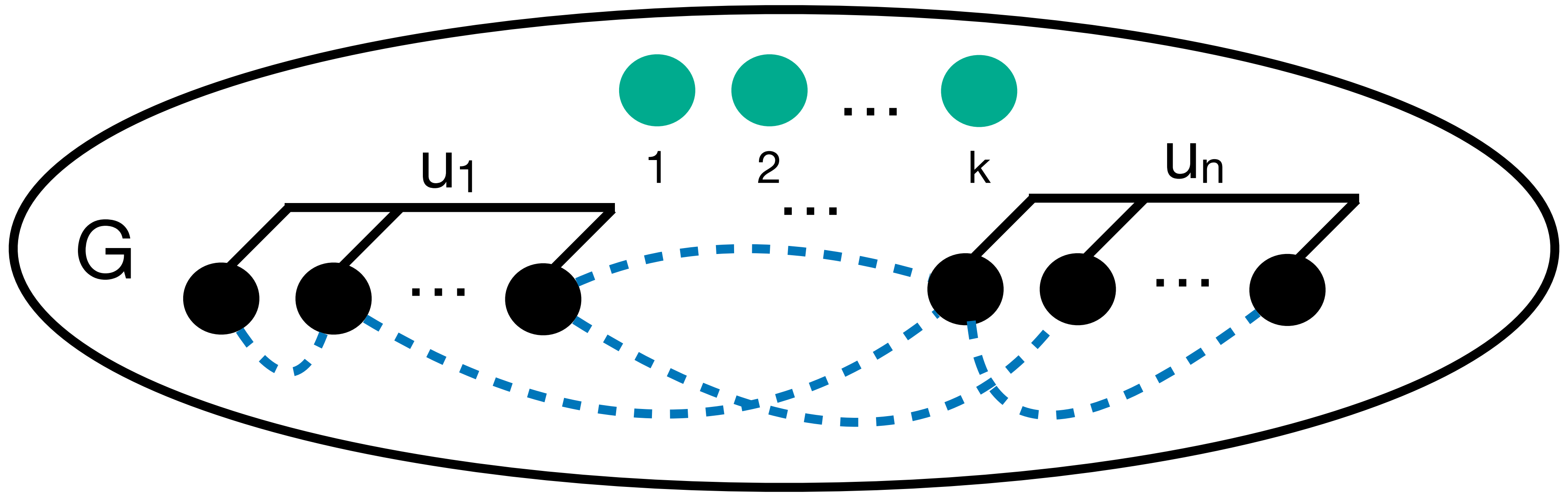
A context-free graph language is a component of the least solution (with rules viewed as set constraints)

Regular Graph Grammars

Hyperedge-replacement (HR) grammars with operations of the form $(G,u_1,\ldots,u_n)$ and $||_k$

Additional conditions on each $(G,u_1,\ldots,u_n)$ [Courcelle'91]

1. G has at least one edge
    - either a single terminal edge with only sources attached,
    - or at least one internal vertex on each edge
2. Any two vertices are linked by a terminal and internal path

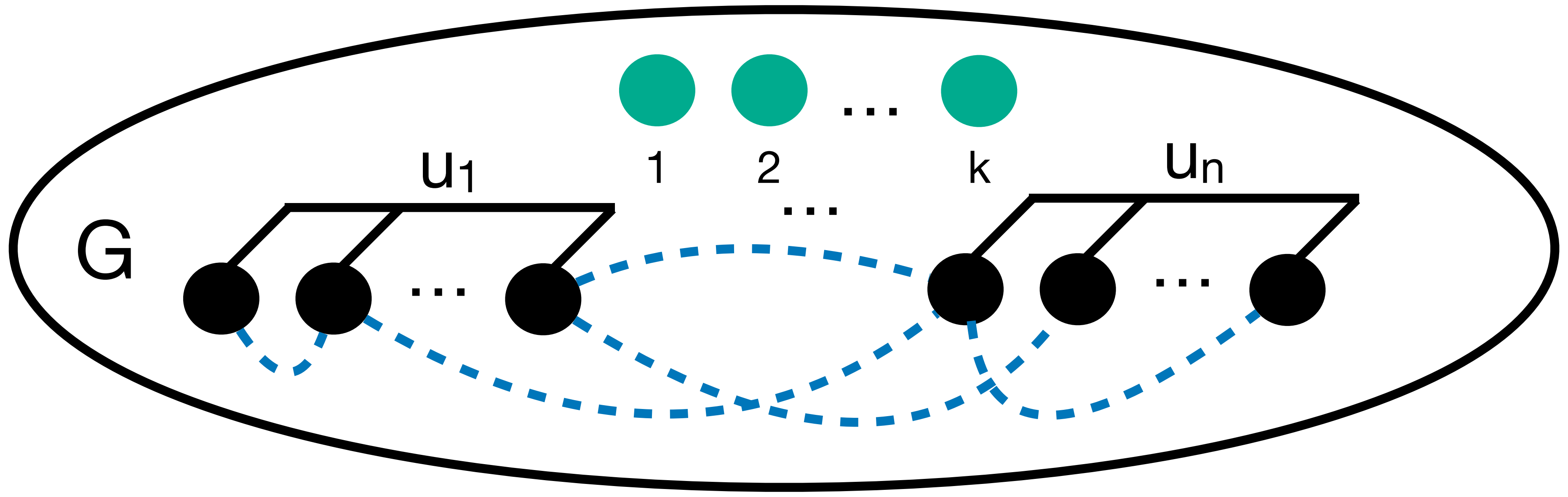


Regular Graph Grammars

Hyperedge-replacement (HR) grammars with operations of the form $(G,u_1,\ldots,u_n)$ and $||_k$

Additional conditions on each $(G,u_1,\ldots,u_n)$ [Courcelle'91]

1. G has at least one edge
   - either a single terminal edge with only sources attached,
   - or at least one internal vertex on each edge
2. Any two vertices are linked by a terminal and internal path



Three types of rules, where U and W are disjoint sets of nonterminals:

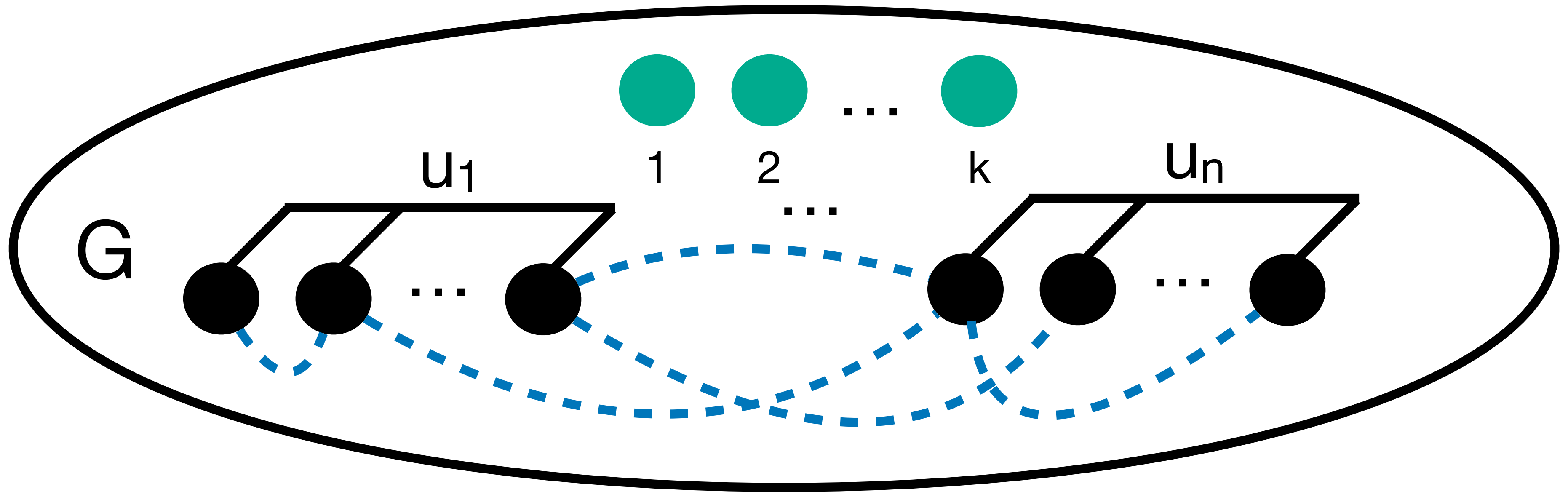
- $u \rightarrow u \,||_k\, w$, $u \in U$, $w \in W$
- $u \rightarrow w_1 \,||_k \ldots ||_k\, w_n$, $u \in U$, $w_1, \ldots w_n \in W$
- $w \rightarrow G(u_1,\ldots,u_n)$, $w \in W$, $u_1,\ldots,u_n \in U \uplus W$

Regular Graph Grammars

Hyperedge-replacement (HR) grammars with operations of the form $(G,u_1,\ldots,u_n)$ and $\|_k$

Additional conditions on each $(G,u_1,\ldots,u_n)$ [Courcelle'91]

1. G has at least one edge
   - either a single terminal edge with only sources attached,
   - or at least one internal vertex on each edge
2. Any two vertices are linked by a terminal and internal path



Three types of rules, where U and W are disjoint sets of nonterminals:

- $u \rightarrow u \|_k w$, $u \in U$, $w \in W$
- $u \rightarrow w_1 \|_k \ldots \|_k w_n$, $u \in U$, $w_1, \ldots w_n \in W$
- $w \rightarrow G(u_1,\ldots,u_n)$, $w \in W$, $u_1,\ldots,u_n \in U \uplus W$

The context-free sets produced by regular graph grammars are MSO-definable [Courcelle'92]

(Regular) Grammars vs (Regular) SIDs

$$u \rightarrow (G, v_1, \ldots v_n)$$

$$P(\underbrace{x_1, \ldots x_{\#P}}_{\text{sources}}) \leftarrow \underbrace{\exists y_1 \ldots \exists y_m}_{\text{internal vertices}} .\ \psi * \pmb{\ast}_{i=1..n}\, Q_i(\underbrace{z_{i,1}, \ldots, z_{i,\#Q_i}}_{\text{nonterminal edges}})$$

(Regular) Grammars vs (Regular) SIDs

$$u \rightarrow (G, v_1, \dots v_n)$$

$$P(\underbrace{x_1, \dots x_{\#P}}_{\text{sources}}) \leftarrow \underbrace{\exists y_1 \dots \exists y_m}_{\text{internal vertices}} .\ \psi * \mathop{\ast}_{i=1..n} Q_i(\underbrace{z_{i,1}, \dots, z_{i,\#Q_i}}_{\text{nonterminal edges}})$$

regular HR operations

⬍

regular inductive definitions

(Regular) Grammars vs (Regular) SIDs

$u \rightarrow (G, v_1, \ldots v_n)$

$$\underbrace{P(x_1, \ldots x_{\#P})}_{\text{sources}} \leftarrow \underbrace{\exists y_1 \ldots \exists y_m}_{\text{internal vertices}} .\ \psi * \mathop{\Large\ast}_{i=1..n} \underbrace{Q_i(z_{i,1}, \ldots, z_{i,\#Qi})}_{\text{nonterminal edges}}$$

regular HR operations

⬍

regular inductive definitions

If Δ is a regular SID, there exists a regular graph grammar that produces the canonical Δ-models of a given SLR sentence

(Regular) Grammars vs (Regular) SIDs

$$u \rightarrow (G, v_1, \ldots v_n)$$

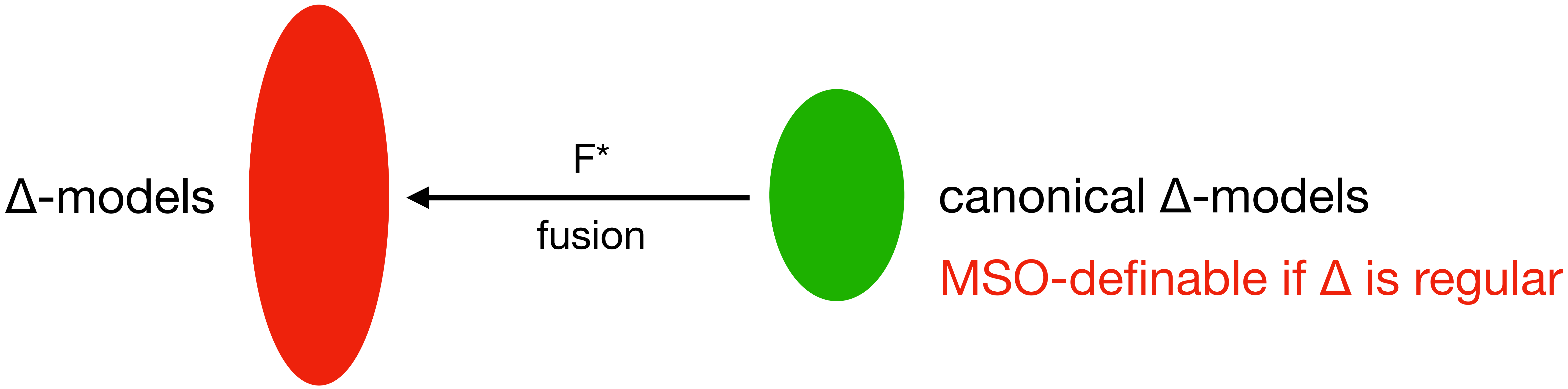
$$P(\underbrace{x_1, \ldots x_{\#P}}_{\text{sources}}) \leftarrow \underbrace{\exists y_1 \ldots \exists y_m}_{\text{internal vertices}} . \psi * \mathop{\ast}_{i=1..n} Q_i(\underbrace{z_{i,1}, \ldots, z_{i,\#Q_i}}_{\text{nonterminal edges}})$$

regular HR operations

⬍

regular inductive definitions

> If Δ is a regular SID, there exists a regular graph grammar that produces the canonical Δ-models of a given SLR sentence



Δ-models

F*

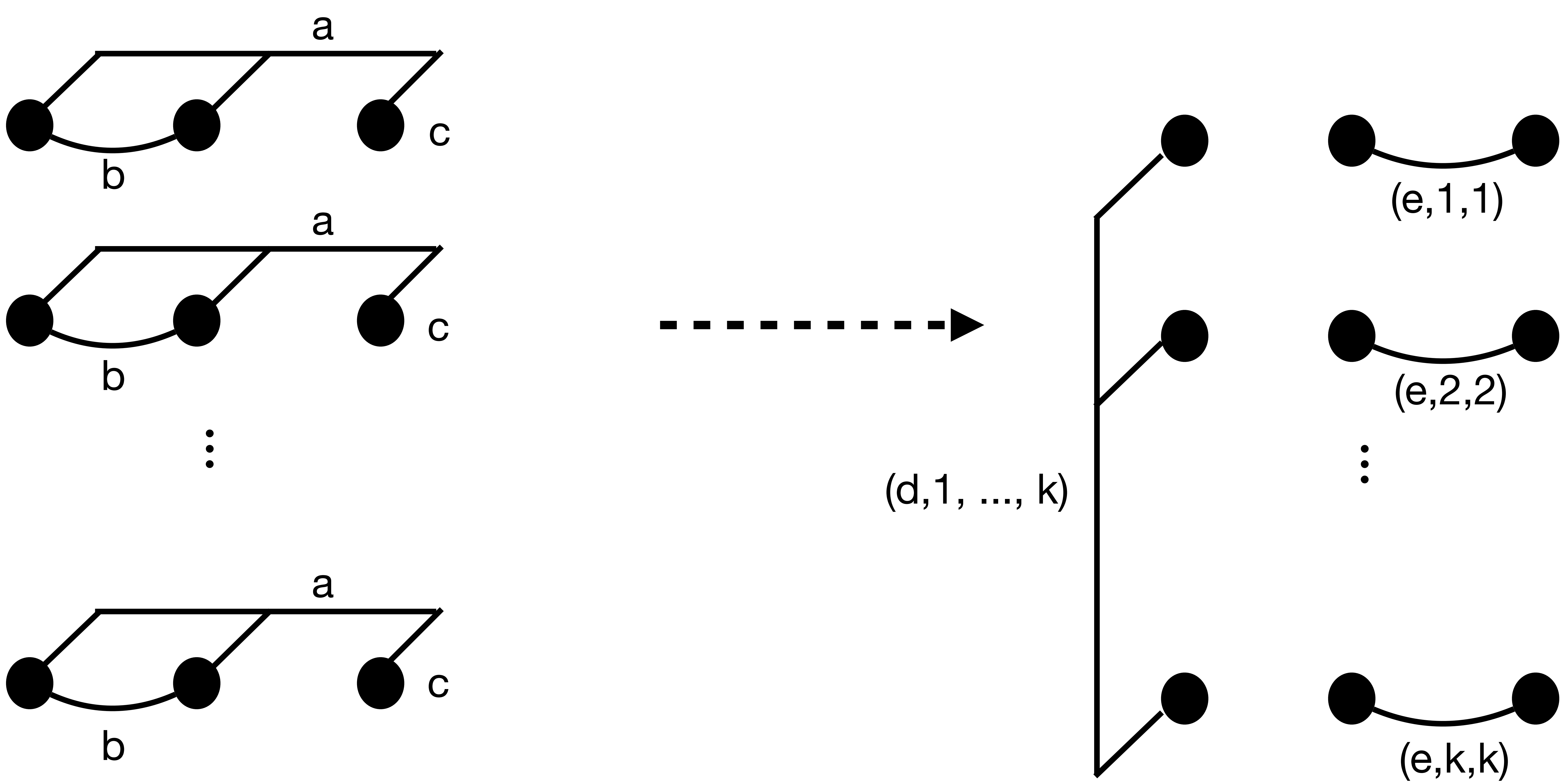
fusion

canonical Δ-models

MSO-definable if Δ is regular

Definable Transductions

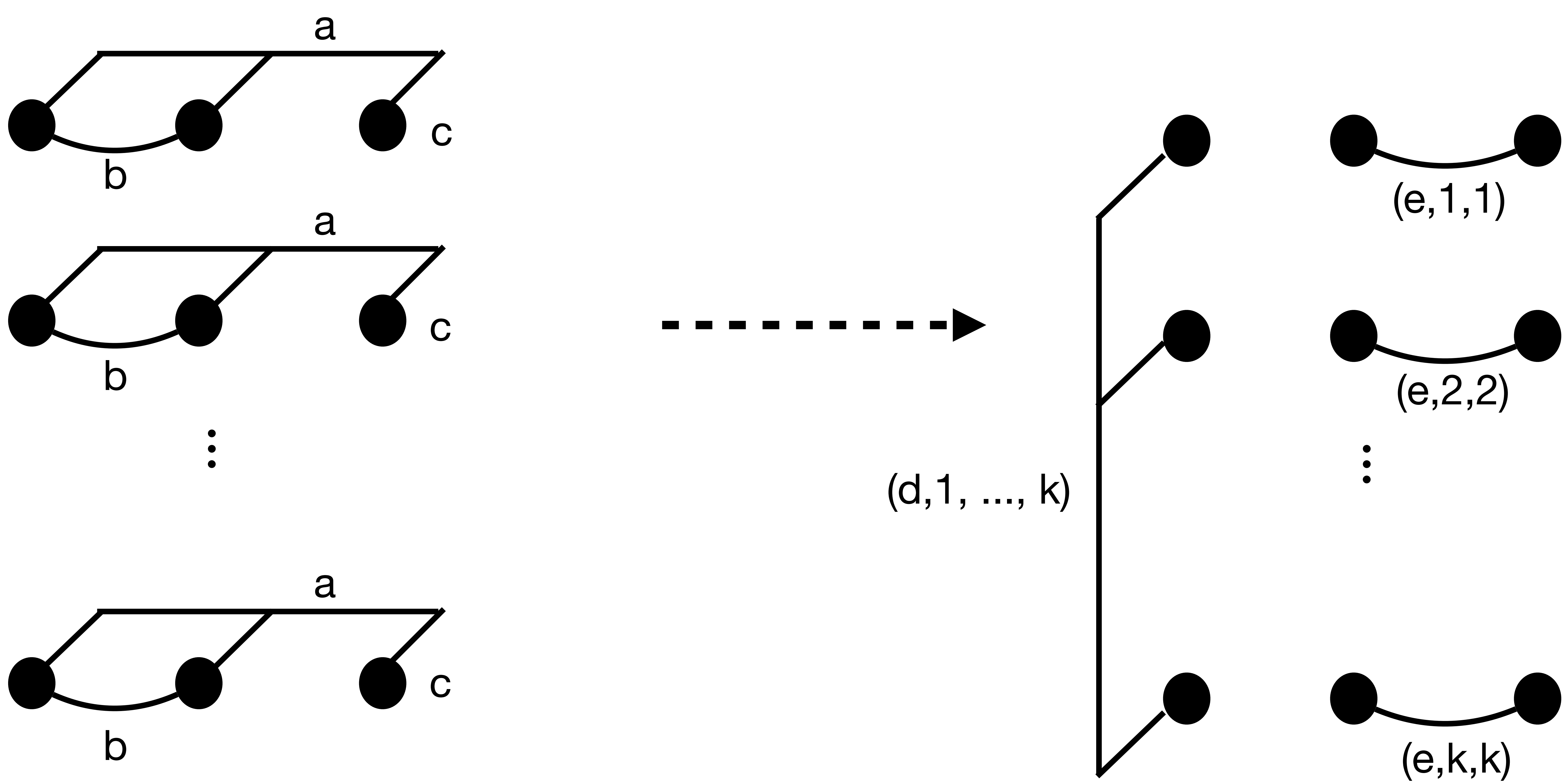
k layers =
copies of the input structure



$\mathbb{R}$={a,b,c}

$\mathbb{R}'$={d,e}

Definable Transductions



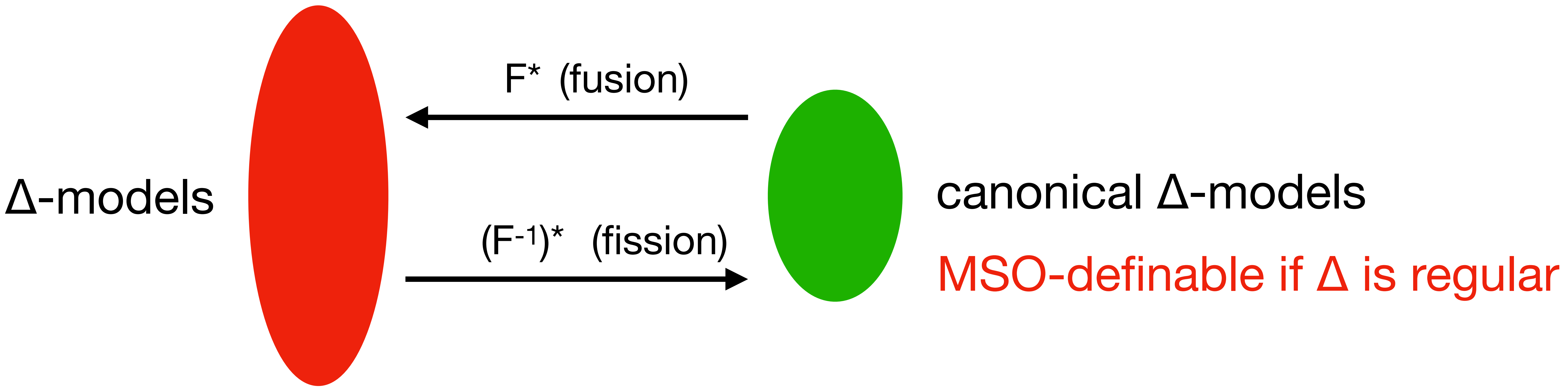
k layers =
copies of the input structure

$\mathbb{R}$={a,b,c}

$\mathbb{R}'$={d,e}

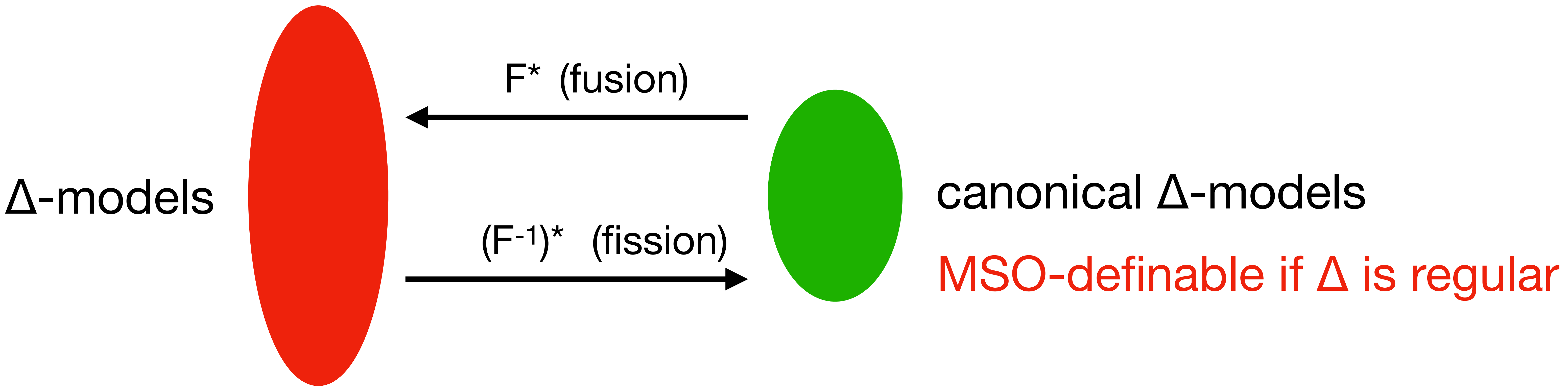
If L’ ⊆ Struc($\mathbb{R}'$) is MSO-definable and R is a definable $\mathbb{R}$-$\mathbb{R}'$ transduction then R^{-1}(L') ⊆ Struc($\mathbb{R}$) is MSO-definable

MSO-Definable Sets of Models



MSO-definable if Δ is regular

MSO-Definable Sets of Models

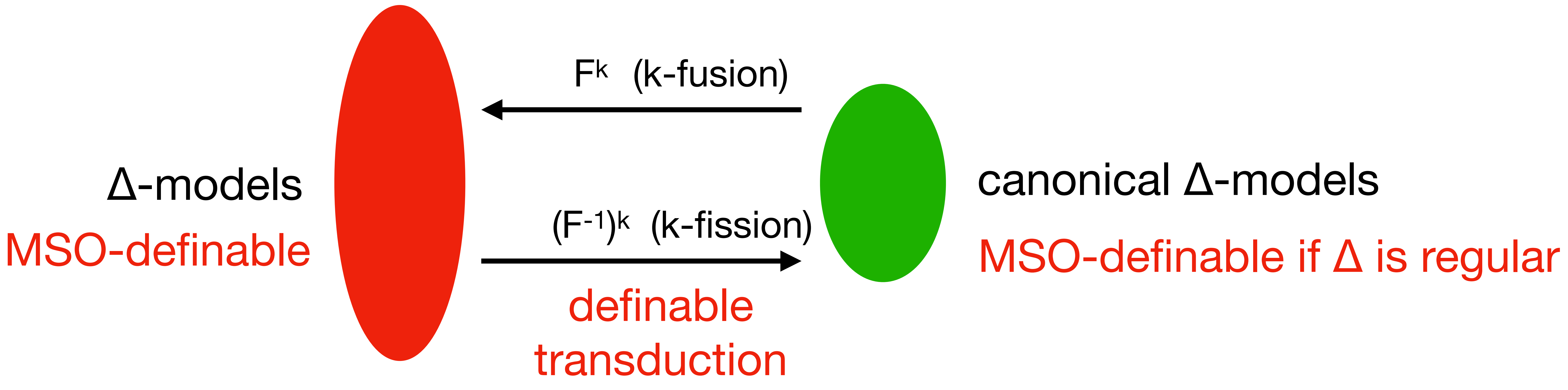


F^{-1} is a definable transduction, but $(F^{-1})^*$ is (provably) not, in general

- transduction scheme that uses quantification over sets of edges

For a regular SID Δ, assuming that the set of Δ-models of a given sentence has bounded tree-width, this set is obtained from the set of canonical Δ-models by applying F^k, for a bounded k≥1

MSO-Definable Sets of Models

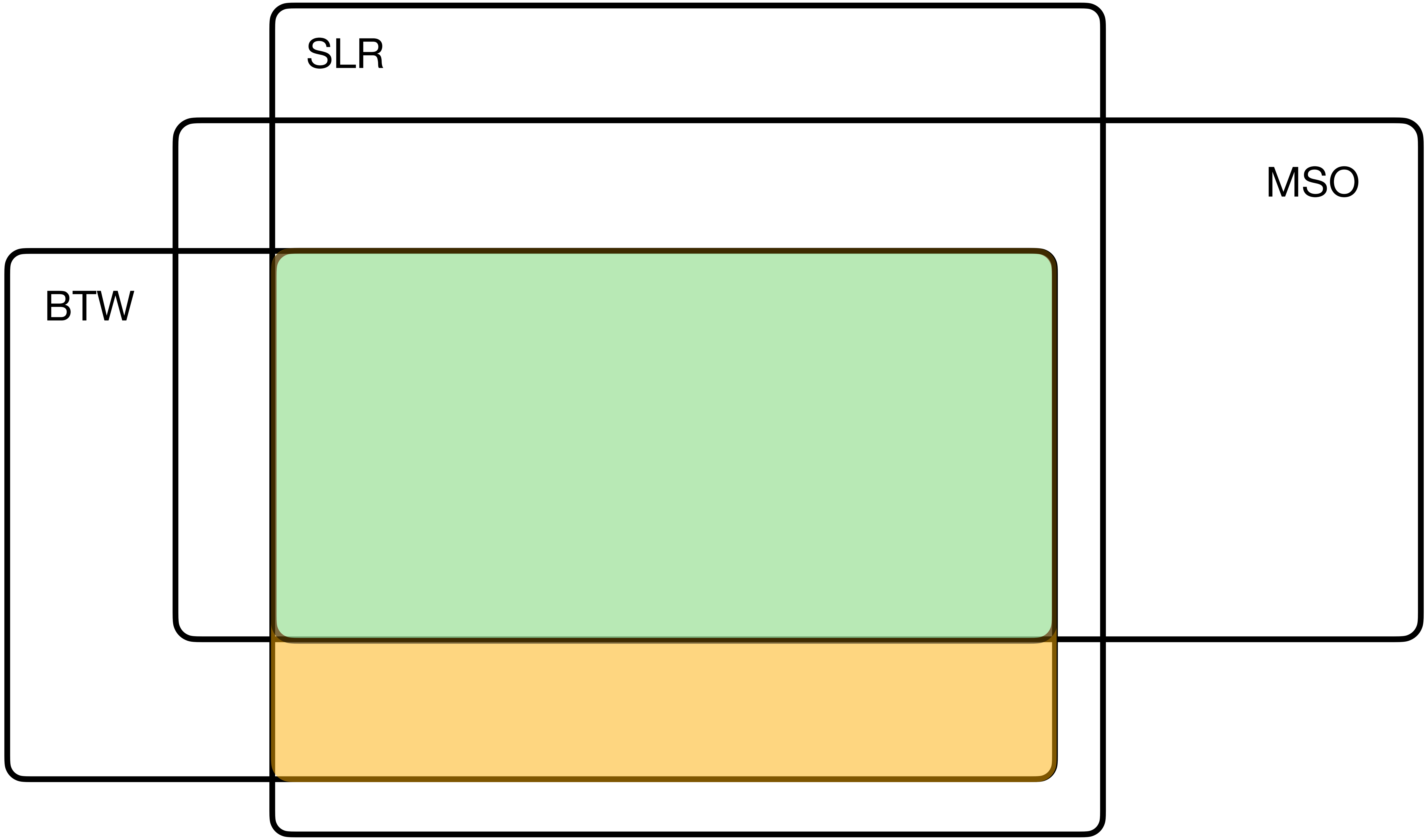


F^{-1} is a definable transduction, but $(F^{-1})^*$ is (provably) not, in general

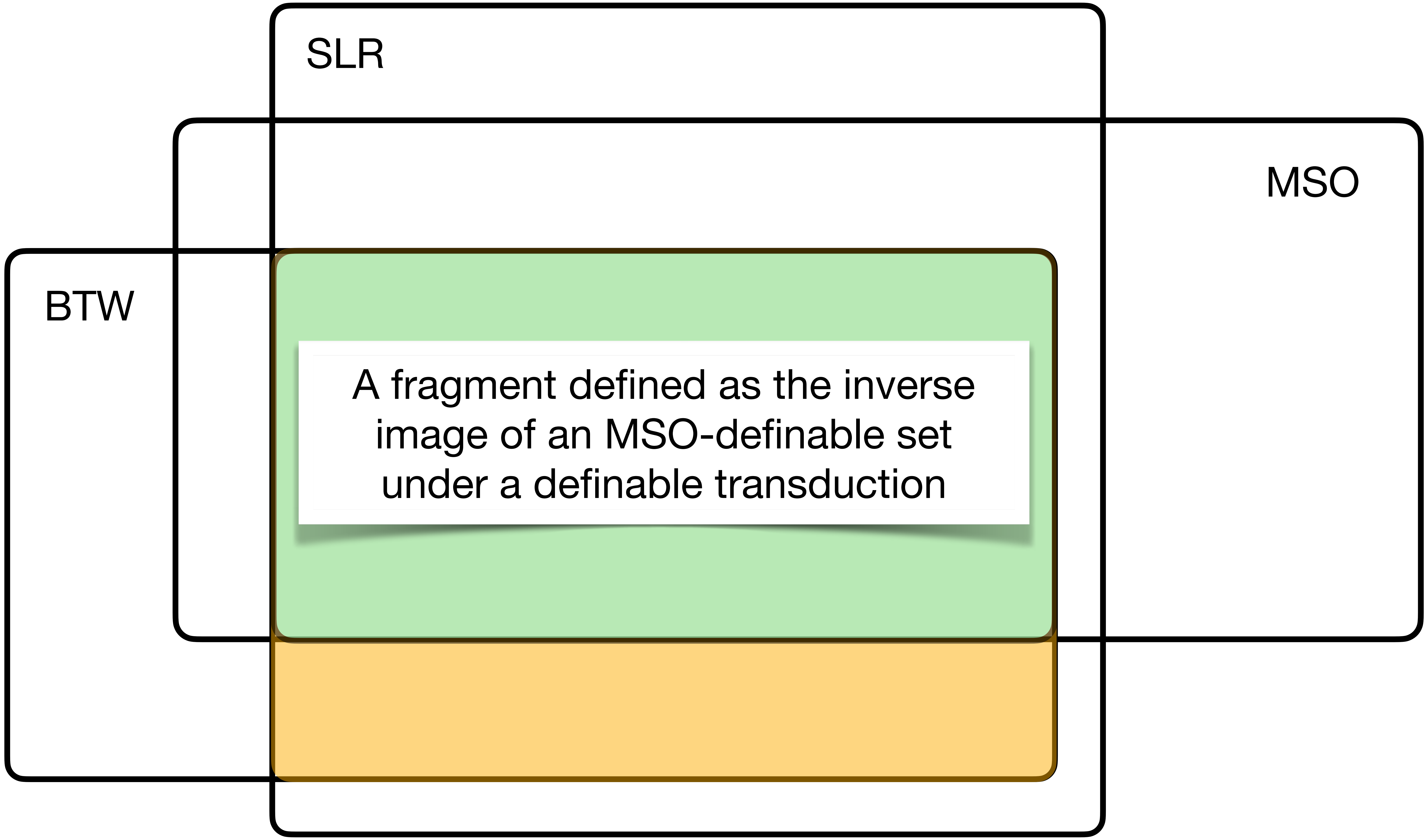
- transduction scheme that uses quantification over sets of edges

For a regular SID Δ, assuming that the set of Δ-models of a given sentence has bounded tree-width, this set is obtained from the set of canonical Δ-models by applying F^k, for a bounded $k \geq 1$

The Big Picture



The Big Picture



A fragment defined as the inverse image of an MSO-definable set under a definable transduction

Conclusions and Future Work

A definition of a large fragment of SLR that describes MSO-definable and tree-width bounded sets of structures

- the idea can be used starting with other MSO-definable HR grammars (e.g., series-parallel graphs)

Conclusions and Future Work

A definition of a large fragment of SLR that describes MSO-definable and tree-width bounded sets of structures

- the idea can be used starting with other MSO-definable HR grammars (e.g., series-parallel graphs)

Future Work

- A grammar-based characterization of HR and (C)MSO-definable sets
- Complexity for entailments between SLR ∩ BTW ∩ CMSO sets