micromodels of software

declarative modelling
and analysis with Alloy

lecture 1: introduction

Daniel Jackson
MIT Lab for Computer Science
Marktoberdorf, August 2002

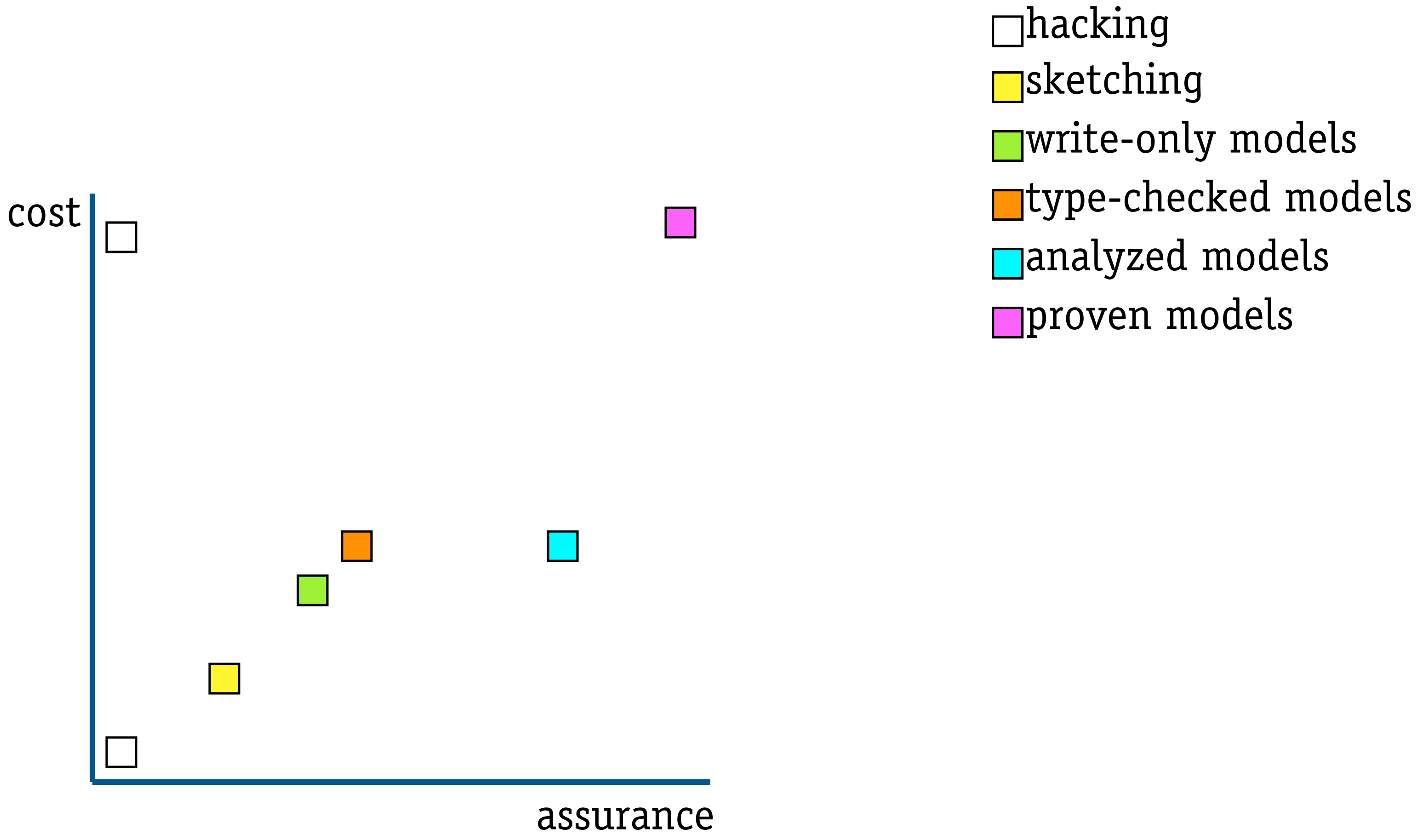
lightweight models

a foundation for robust, useable programs

elements
- small & simple notations
- partial models & analyses
- full automation

focus on risky aspects
- hard to get right, or to check
- structure-determining
- high cost of failure

what assurance costs



my work in marktoberdorf context

computation, not interaction

- complementary to Harel & Pnueli
- relational, not algebraic (cf. Tarlecki and Meseguer)
- underlying idioms due to Hoare, Woodcock et al

designed for experts, but not super-experts

- like Harel, not Rushby & Moore
- simulation, not just checking

role of mathematics

- only way to make things simple
- semantics in terms of sets, and SAT

started this in 1994, and have had some successes
but much less mature than ACL2, PVS, Statemate, etc

features of Alloy

structural

- express complex structure, static and dynamic
- with just a few powerful operators

declarative

- a full logic, with conjunction and negation
- describe system as collection of constraints

analyzable

- simulation & checking
- fully automatic

structural

structure is everywhere

- highway systems, postal routes, company organizations, library catalogues, address books, phone networks, …

structure is becoming more pervasive

- self-assembling software (eg, Observer pattern)
- memory gets cheaper: address books in every phone

tool researchers have neglected structure

- one traffic light is a state machine, but a city’s lights are a net

There is no problem in computer science that cannot be solved by introducing another level of indirection, but that usually reveals new problems *--David Wheeler*

declarative

declarative description

- model is collection of properties
- the more you say, the less happens

advantages

- incrementality: to say more, add a property
- partiality: doesn’t require special constructs
- simplicity: no separate language of properties

```
Sys meets Prop:  Sys => Prop
```

why less is more

- less constrained system means implementation freedom
- less constrained environment means greater safety

analyzable

‘write-only’ models

- useful if precise enough
- but missed opportunity (and wishful thinking)

tool-assisted modelling

- simulate and check incrementally
- catch errors early, develop confidence
- optimize for failing case: most of my examples will be wrong

Alloy’s analysis

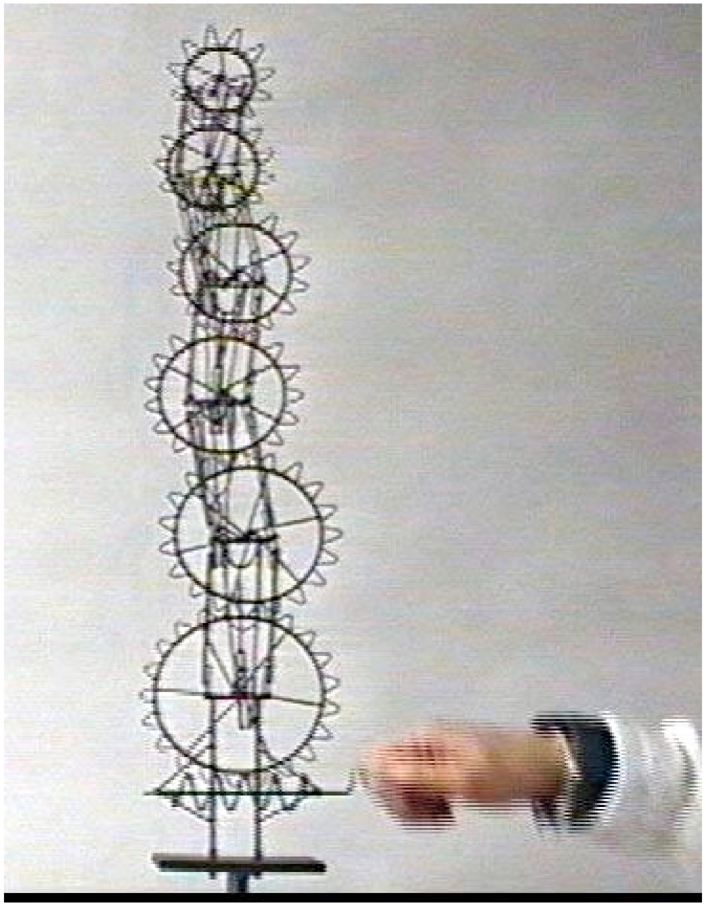
- fully automatic, with no user intervention
- concrete: generates samples & counterexamples
- like testing, sound but not complete
- unlike testing, billions cases/second

declarative & executable?

traditionally
- declarative XOR executable
- good arguments for both

but can have cake and eat it
- with right analysis technology

Alloy’s analysis can ‘execute’ a model
- forwards or backwards
- without test cases
- no ad hoc restrictions on logic



Small Tower of 6 Gears, Arthur Ganson

a numbering problem

given

- document whose paragraphs are tagged with styles
- style sheet that gives numbering rules for styles

produce

- document with numbered paragraphs
  (like my Marktoberdorf notes)

```
\part Introduction
\section Motivation
\subsection Why?
\section Overview
\part Conclusions
\section Unrelated Work
```

```
Part A: Introduction
A.1 Motivation
A.1.1 Why?
A.2 Overview
Part B: Conclusions
B.1 Unrelated Work
```

a candidate solution

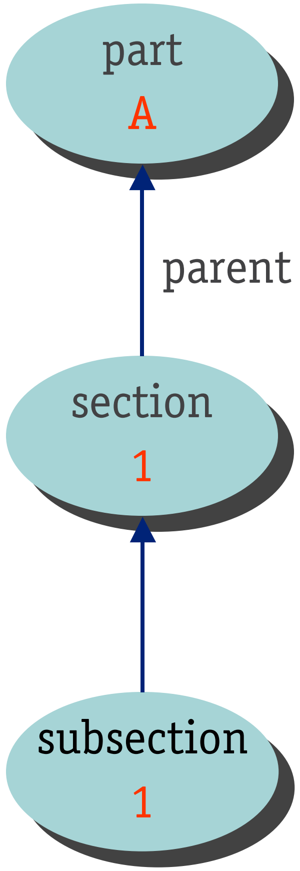
style sheet assigns to each style

- an initial value for numbering
- optionally, a parent

```
<style:part><init:A>
<style:section><parent:part><init:1>
<style:subsection><parent: section><init:1>
```

```
\part Introduction
\section Motivation
\subsection Why?
\section Overview
\part Conclusions
\section Unrelated Work
```

```
Part A: Introduction
A.1 Motivation
A.1.1 Why?
A.2 Overview
Part B: Conclusions
B.1 Unrelated Work
```



styles

declare styles & parent relation

```
sig Style {parent: option Style}
```

ask for a sample

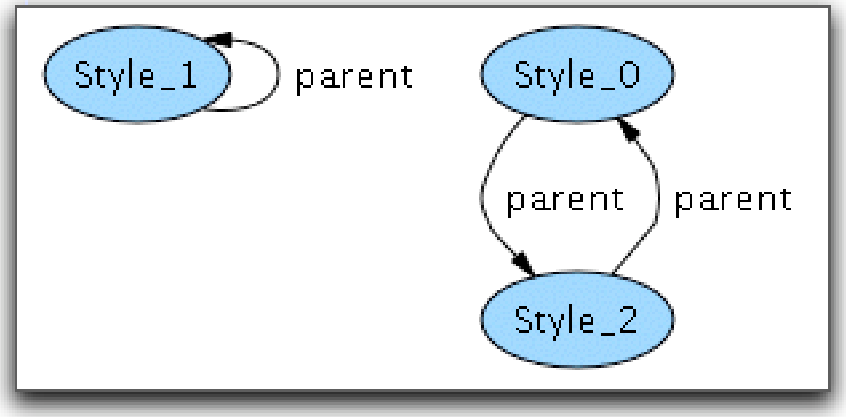
```
fun Show () {some parent}
run Show
```

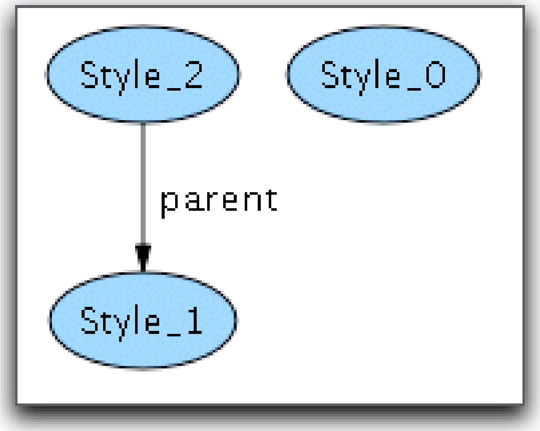
constrain parent relation to be acyclic

```
fact {Acyclic (parent)}
```

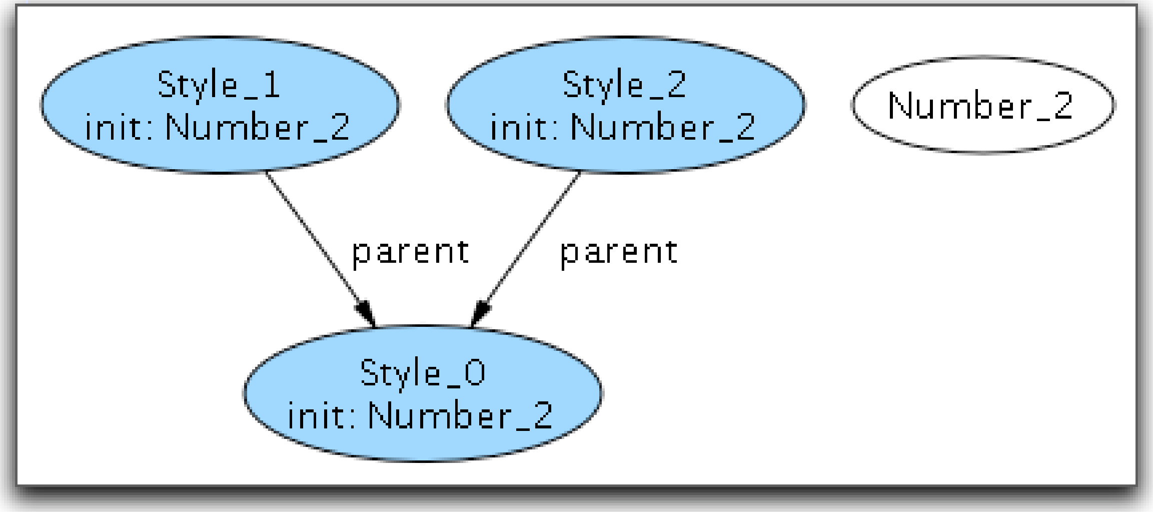
how to define acyclic

```
fun Acyclic [t] (r: t -> t) {no iden[t] & ^r}
```





numbers



introduce numbers

```
sig Number {
  next: option Number
  }{this != next}
```

add numbers to styles

```
sig NumberedStyle extends Style {init: Number}
fact {Style = NumberedStyle}
```

ask for a sample

```
fun Show () {
  some parent}
run Show
```

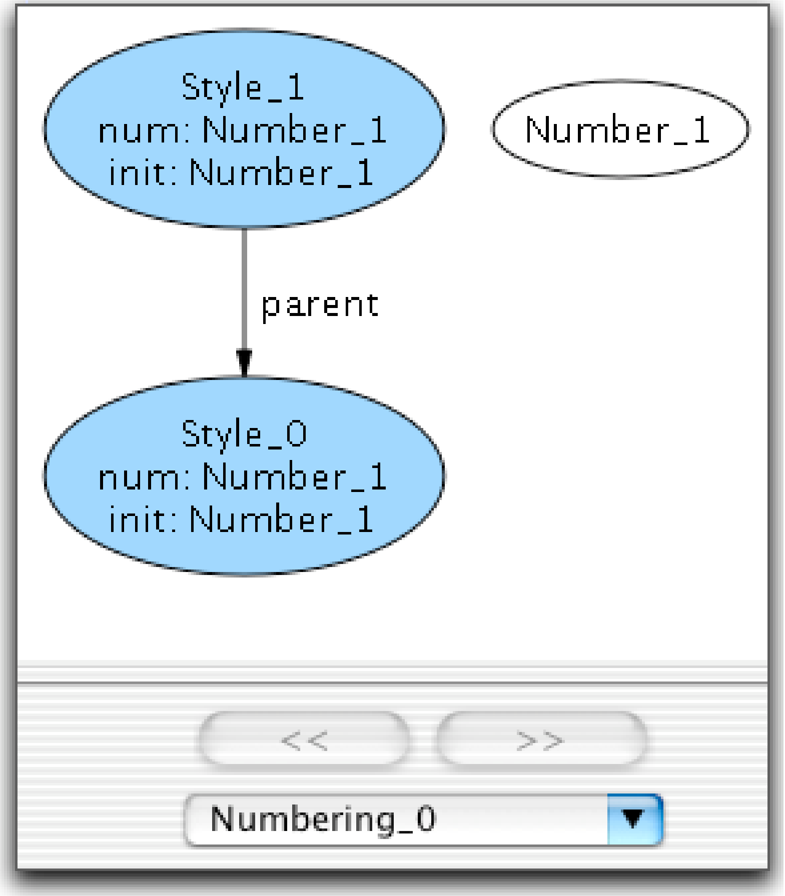
numbering

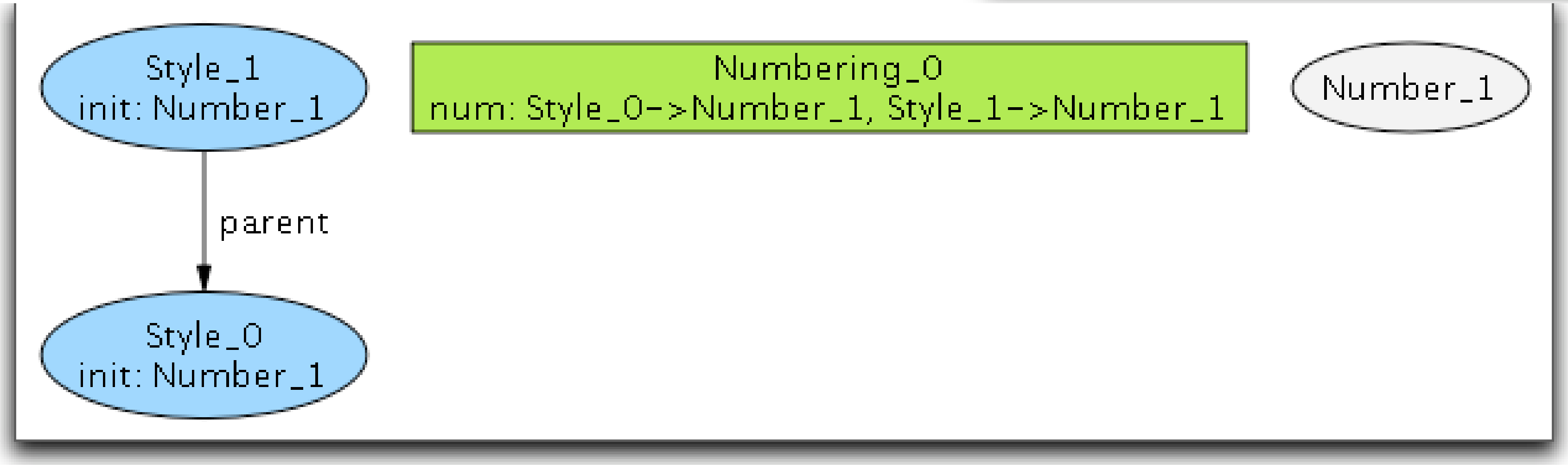
declare numbering

```
sig Numbering {
  num: Style ->? Number}
```

ask for a sample

```
fun ShowNumbering () {some num}
run ShowNumbering
  for 2 but 1 Numbering
```





numbering algorithm

what numbering *n’* follows *n* for paragraph of style *s*?

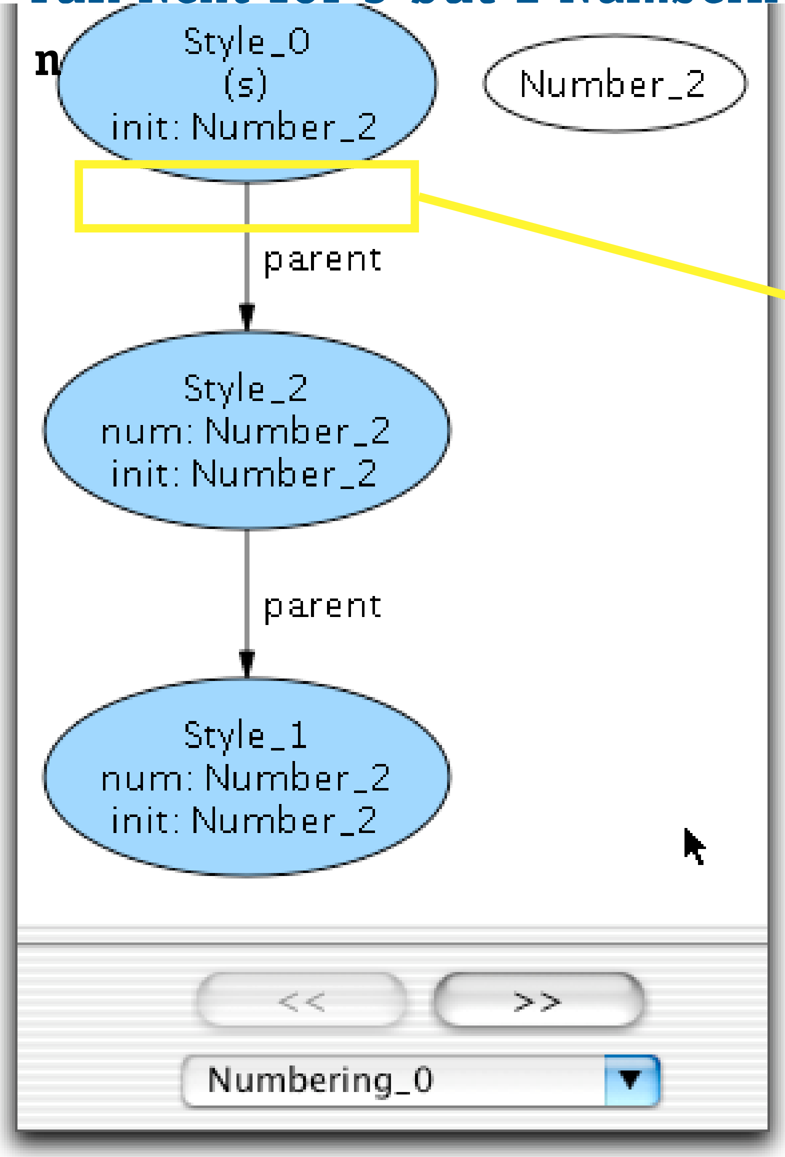
- ie, just gave numbering *n*
- encounter paragraph with style *s*
- must now generate numbering *n’*

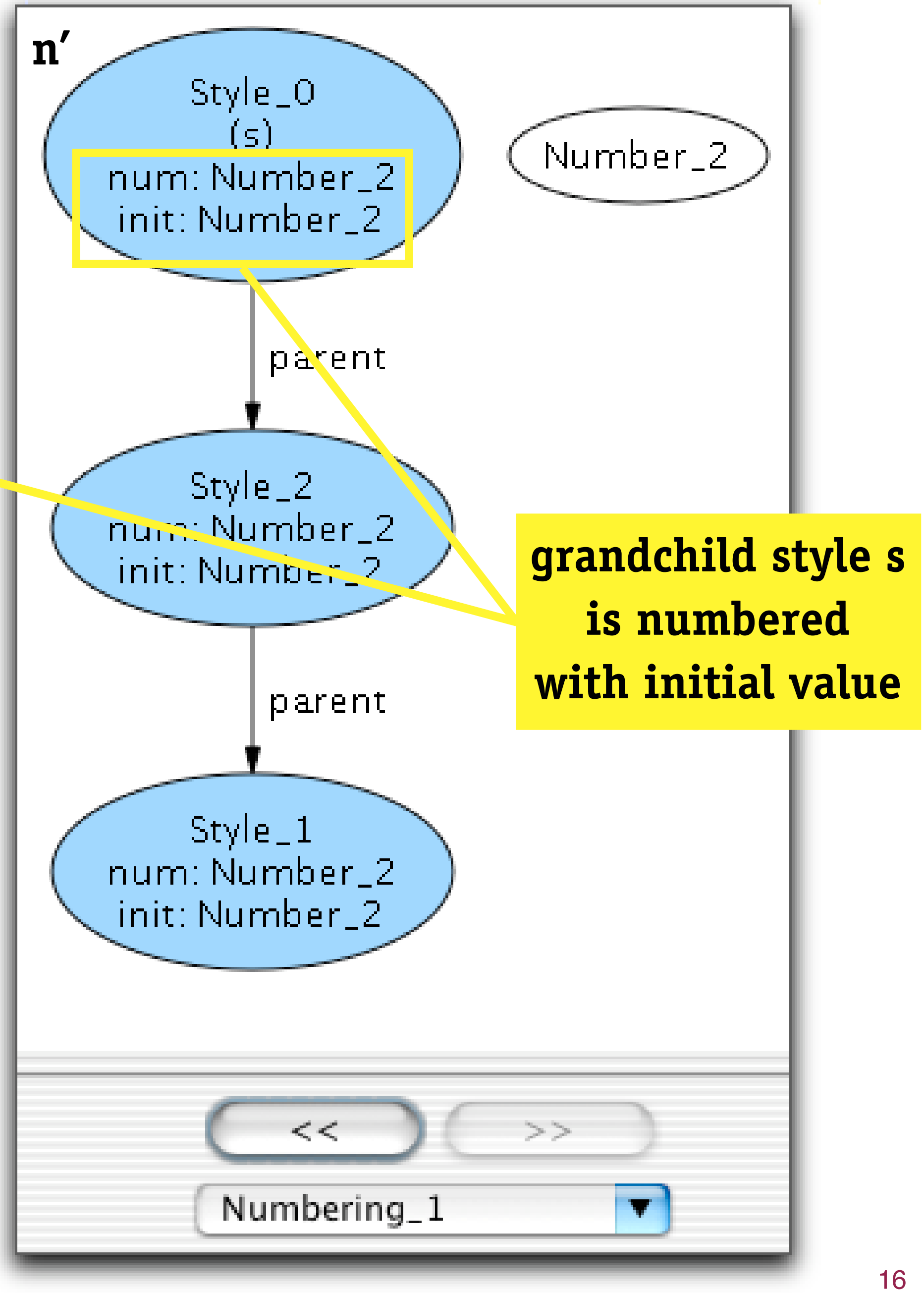
an attempt:

```
fun Next (n,n': Numbering, s: Style) {
  n'.num =
    {d: s.^parent, x: Number | x = n.num[d]} +
    s -> if no n.num[s] then s.init else n.num[s].next
  }
```

showing next

run Next **for** 3 **but** 2 Numbering

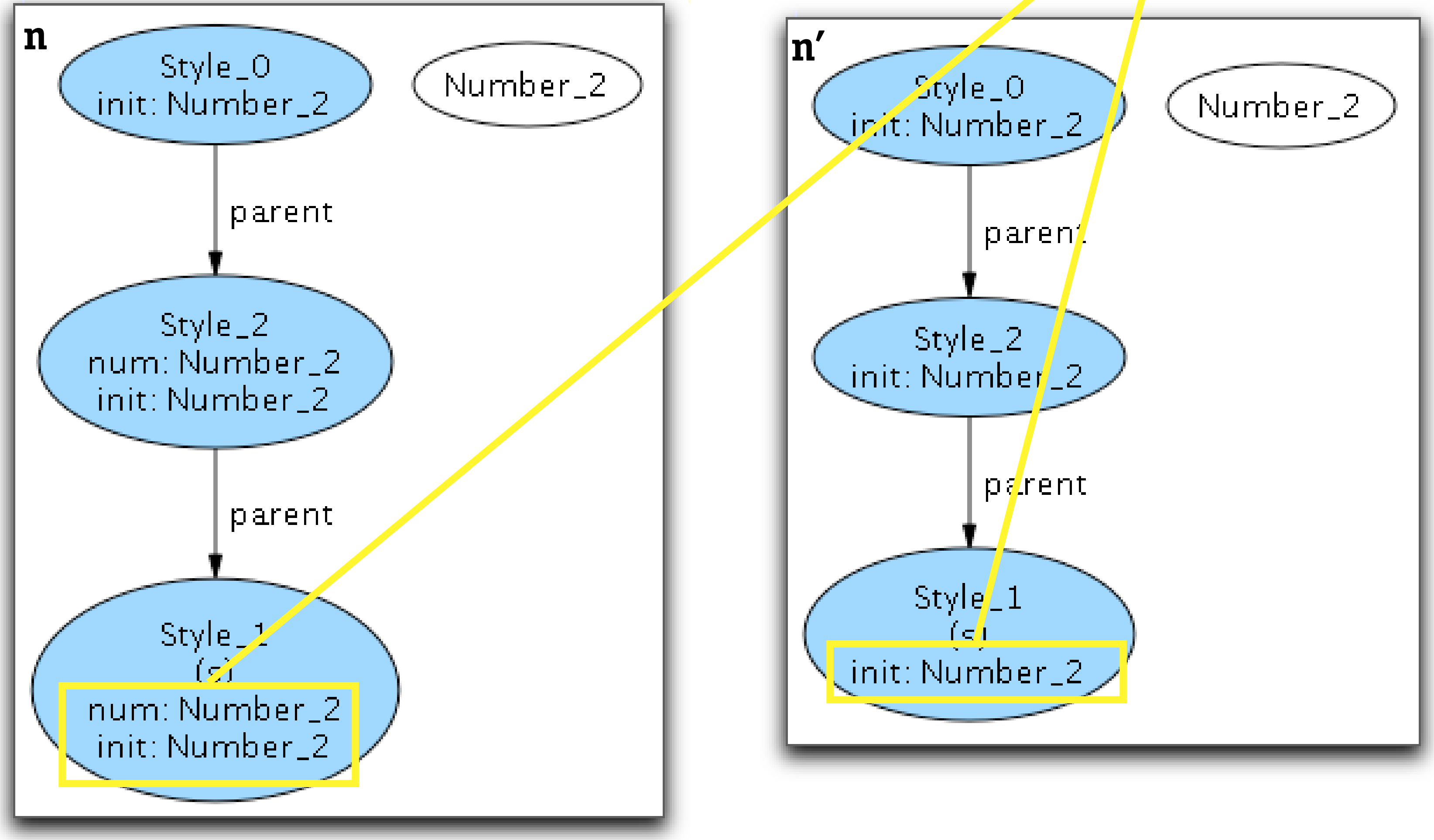




grandchild style s is numbered with initial value

guiding the simulation

```
fun ShowNext (n,n': Numbering, s: Style) {
  Next (n,n',s)  && some n.num[s.~parent]}
run ShowNext for 3 but 2 Numbering
```

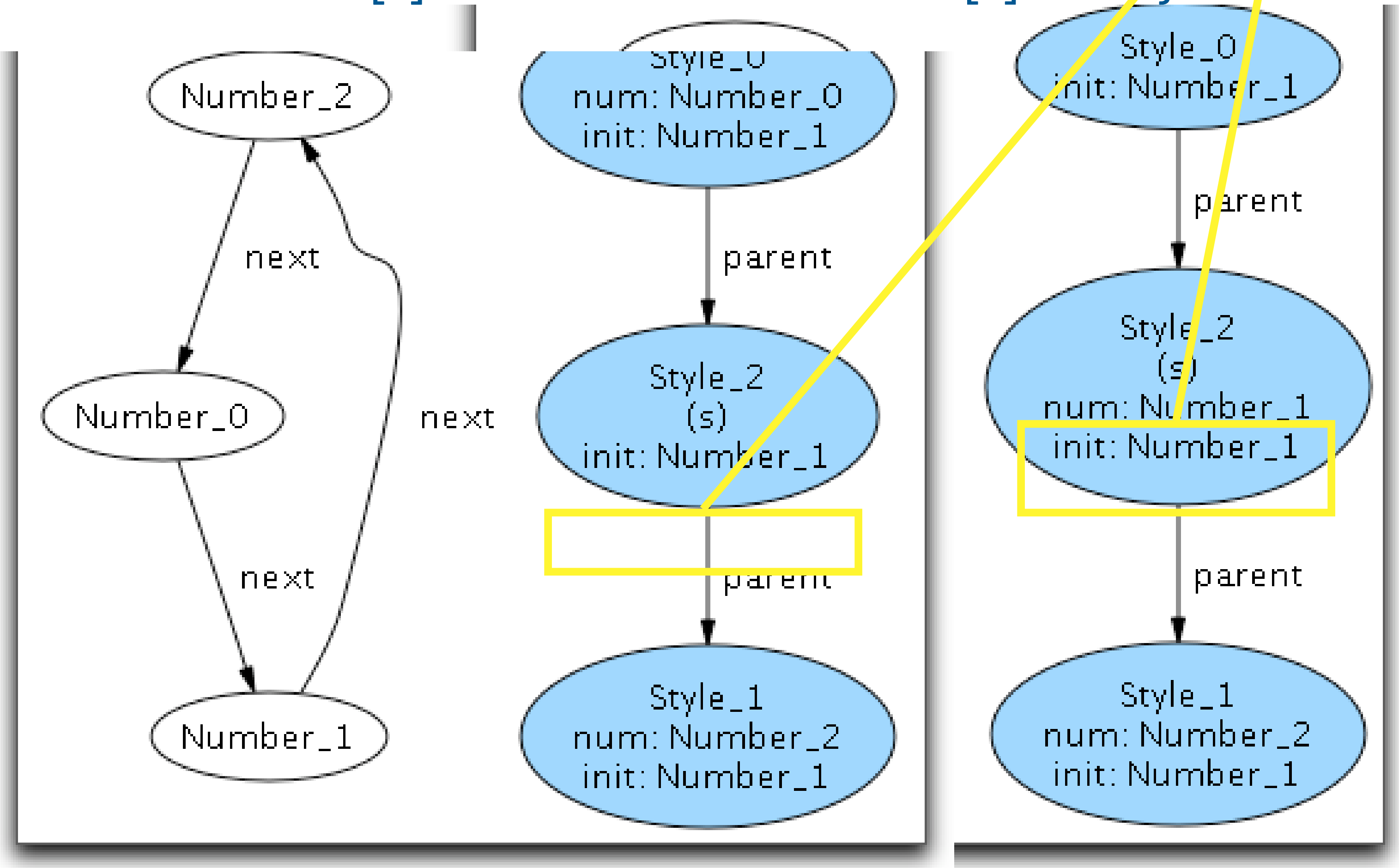


root style s
loses its number
because no next!

fixing the operation

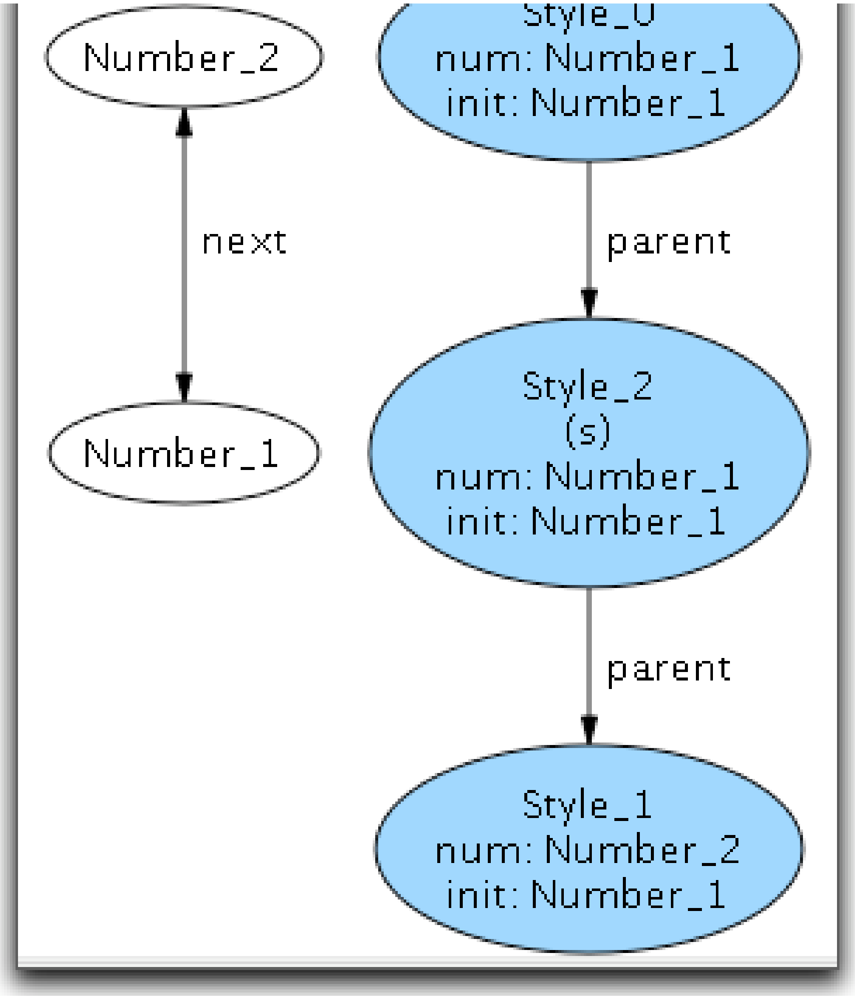
```
fun Next (n,n': Numbering, s: Style) {
  let i = n.num[s] | some i => some i.next
  n'.num = {d: s.^parent, x: Number | x = n.num[d]} +
    s -> if no n.num[s] then s.init else n.num[s].next }
```

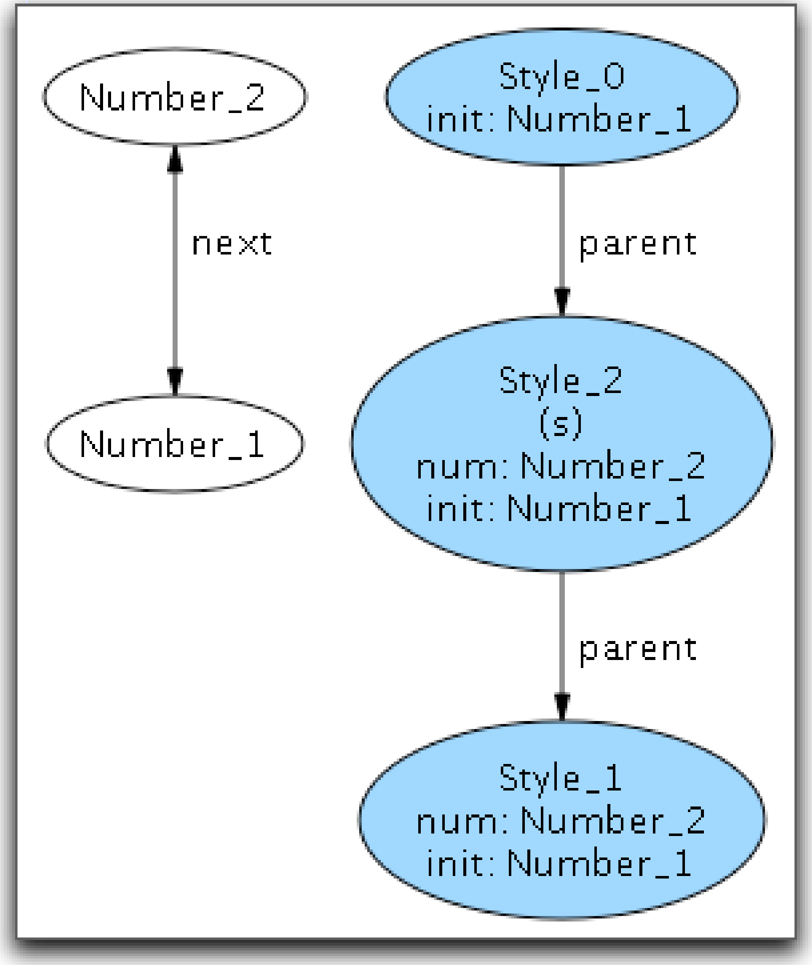
style s gets numbered with initial value



guiding the simulation

```
fun ShowNext (n,n': Numbering, s: Style) {
  Next (n,n',s)  && some n.num[s.~parent] && some n.num[s]}
run ShowNext for 3 but 2 Numbering
```



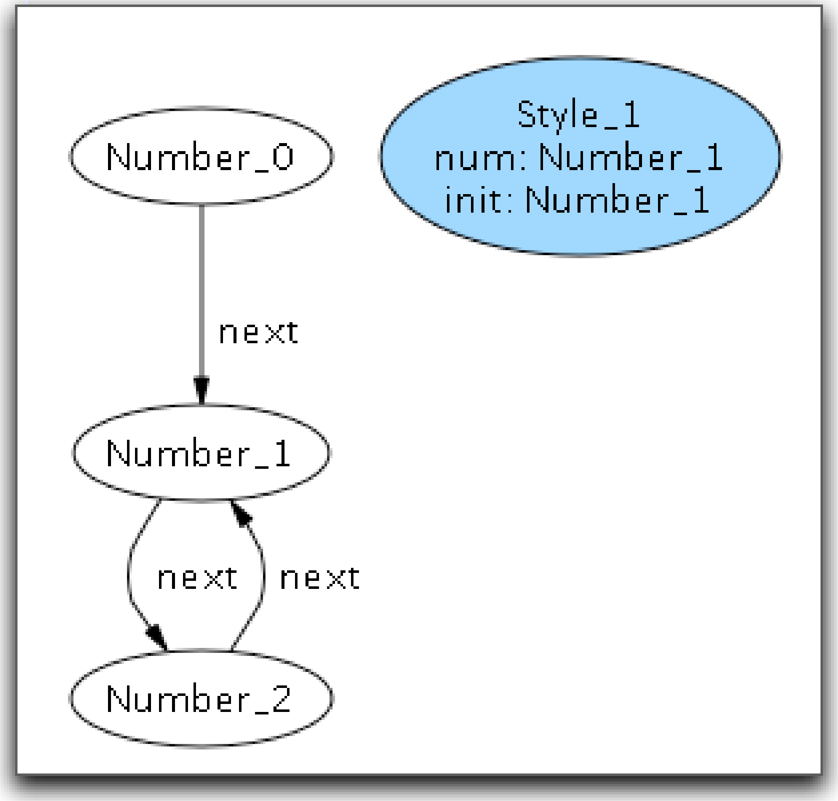


checking a property

if style is not a parent, step is reversible

```
assert Reversible {
  all n0, n1, n: Numbering, s: Style - Style.parent |
    Next(n0,n,s) && Next(n1,n,s) => n0.num = n1.num}

check Reversible
```



trying again…

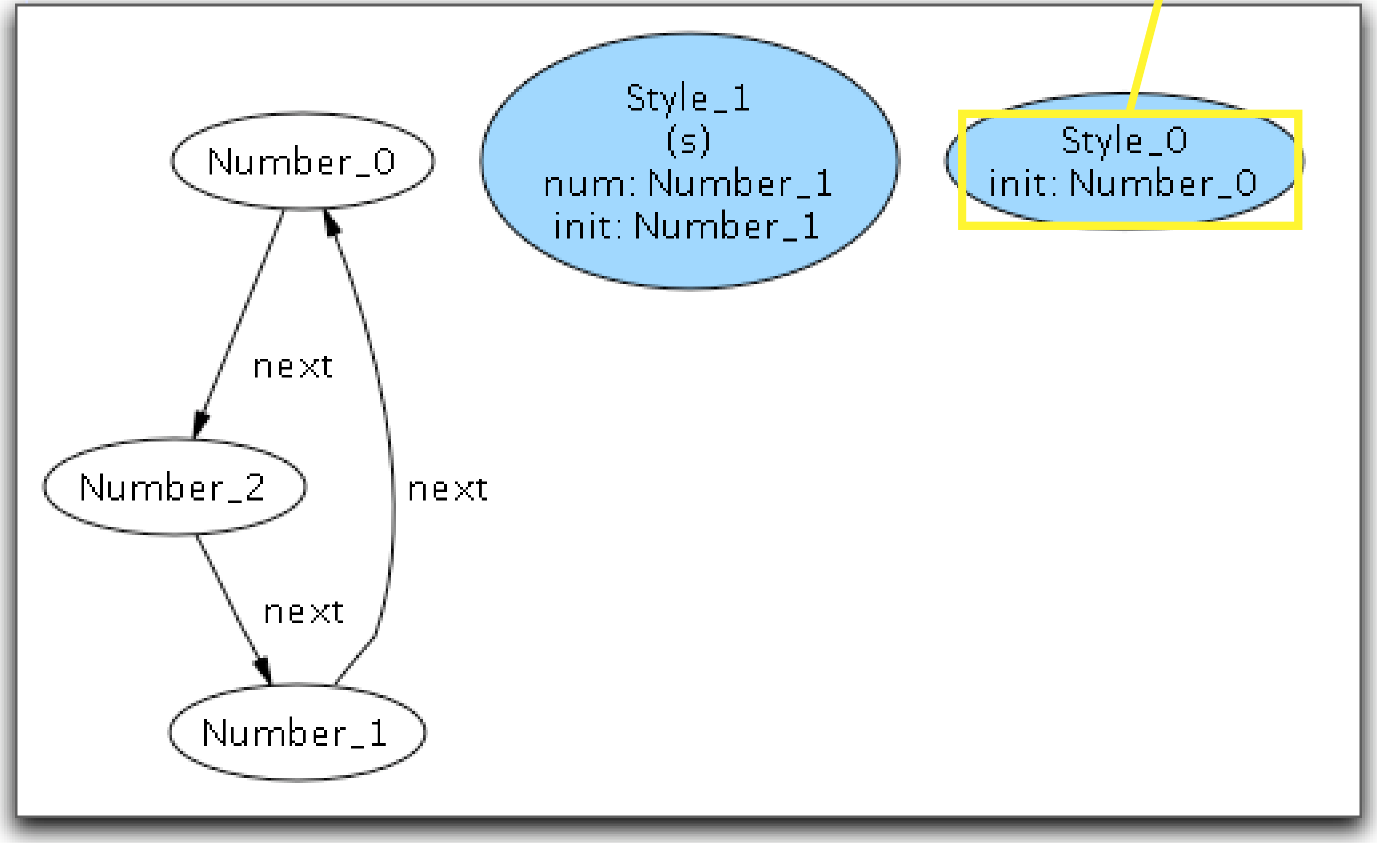
make numbering injective

```
    fact {Injective (next)}
```

does this fix the problem?

counterexample

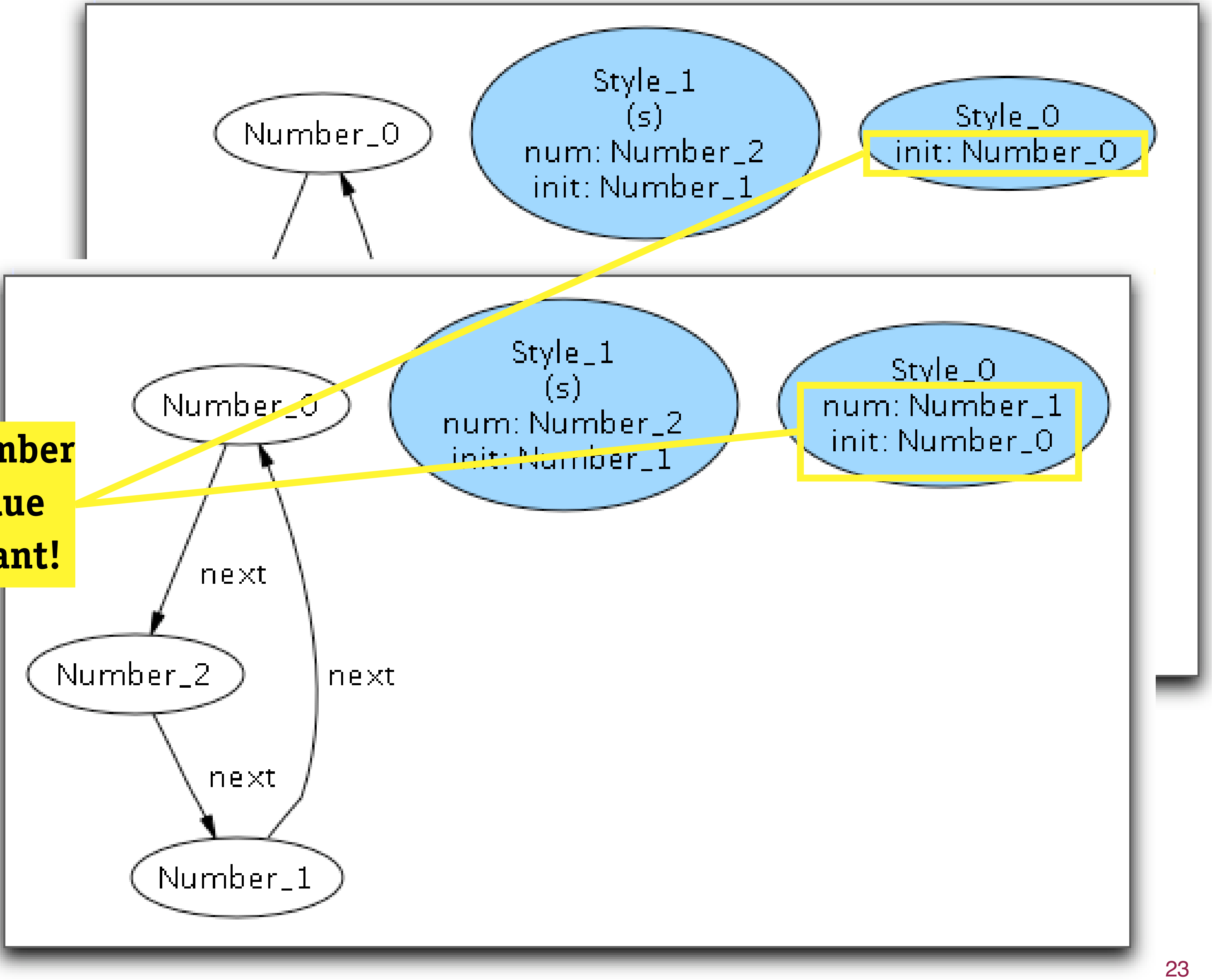
after numbering n



adjacent style has no number afterwards

counterexample, ctd

before numberings *n0* and *n1*



adjacent style's number eliminated, so value before was irrelevant!

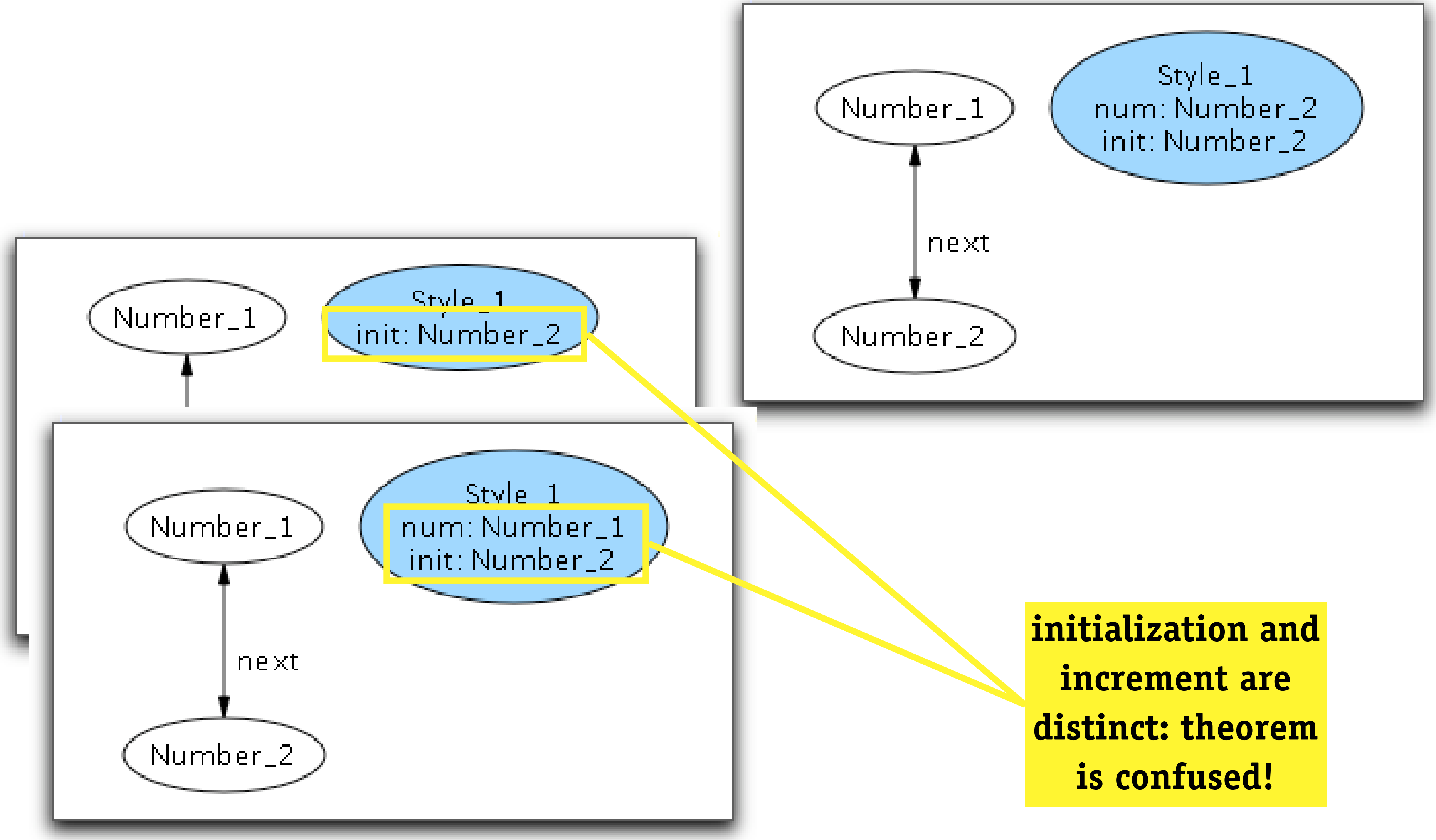
masking

check again, assuming styles form a line

```
assert ReversibleWhenLine {
  Injective(parent)
  && (some root: Style | Style in root.*~parent) =>
  all n0, n1, n: Numbering, s: Style - Style.parent |
    Next(n0,n,s) && Next(n1,n,s) => n0.num = n1.num}

check ReversibleWhenLine
```

counterexample, again



initialization and increment are distinct: theorem is confused!

checking a refactoring

are these equivalent?

```
fun Next1 (n,n′: Numbering, s: Style) {
  n′.num =
    {d: s.^parent, x: Number | x = n.num[d]} +
    s -> if no n.num[s] then s.init else n.num[s].next
  }
fun Next2 (n,n′: Numbering, s: Style) {
  all d: s.^parent | n′.num[d] = n.num[d]
  n′.num[s] = if no n.num[s] then s.init else n.num[s].next
  }
```

ask the tool:

```
assert Same {
  all n,n′: Numbering, s: Style | Next1(n,n′,s) iff Next2(n,n′,s)}
```

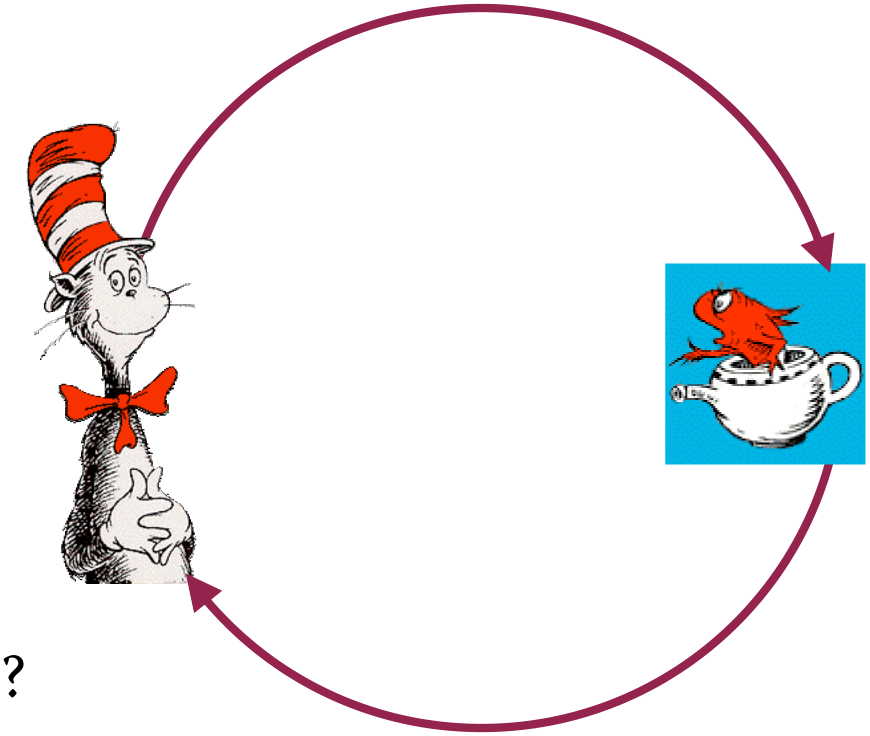
what happened

incrementality

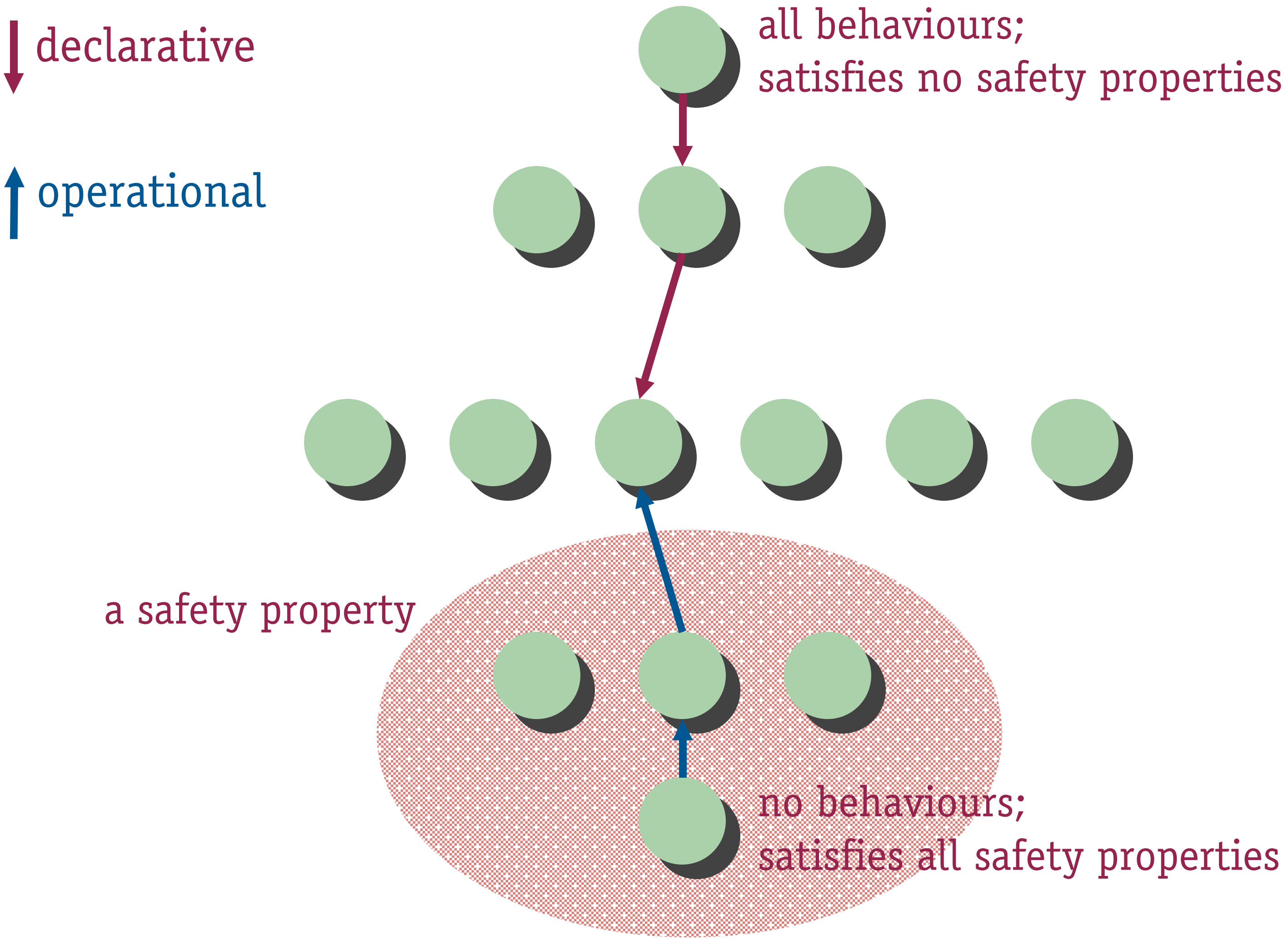
- write a bit, analyze a bit
- constrain just enough to get key properties
- avoids wasted time, encourages small models

analysis prompted questions

- number must have next?
- two numbers have same next?
- style hierarchy a tree? line?



declarative vs. operational development



declarative

operational

all behaviours;
satisfies no safety properties

a safety property

no behaviours;
satisfies all safety properties

what’s been done?

analyzing implemented systems

- Intentional naming (Khurshid)
- Chord peer-to-peer lookup (Wee)
- Transaction cache (Tucker)

analyzing existing models

- Microsoft COM (Sullivan, from Z)
- Firewire leader election (me, from Vaandrager’s IOA)
- Unison file synchronizer (Nolte, from Pierce’s maths)
- UML meta model (Vaziri, from OCL)
- Classic distributed algorithms (Shlyakhter, from SMV)

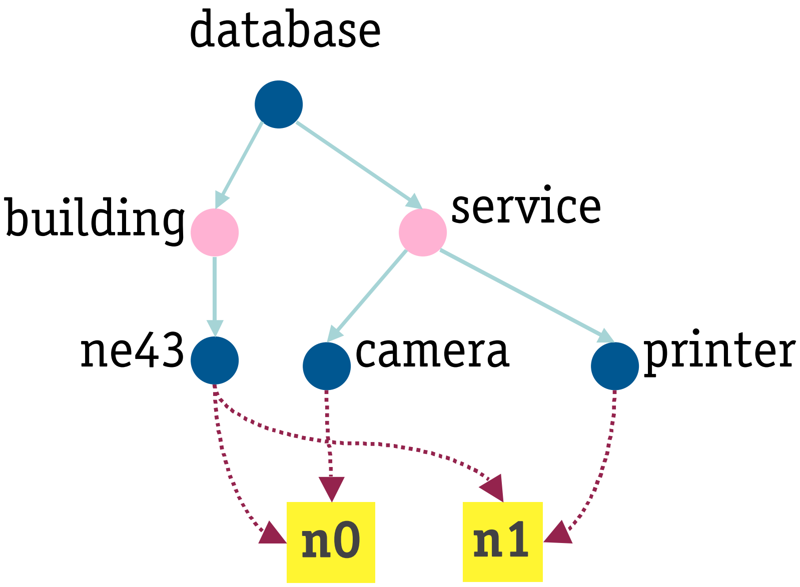
typically

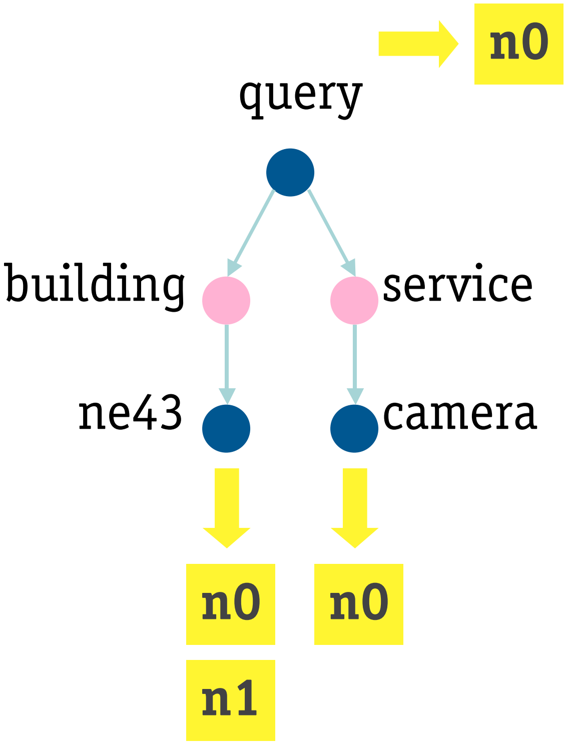
- 200 lines of Alloy, 30-200 hours work

example: intentional naming

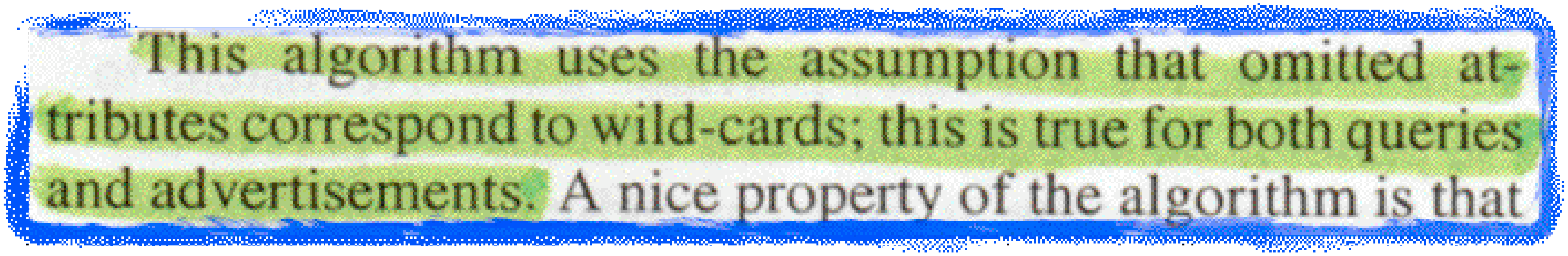
query scheme

- intentional names are trees
- result of query is set of simple names





results



This algorithm uses the assumption that omitted attributes correspond to wild-cards; this is true for both queries and advertisements. A nice property of the algorithm is that

what we did

- analyzed claims made in paper: mostly untrue
- analyzed algebraic properties: also untrue
  eg, add is monotonic
- adapted model for fixes in code: also broken
- developed new semantics & checked it

reflections

- initial analysis took 2 weeks and 100 lines of Alloy
- found all bugs in trees of 4 nodes or less -- approx 10 secs
- 2000 lines of tests hadn’t found bugs in a year

challenge: get numbering right

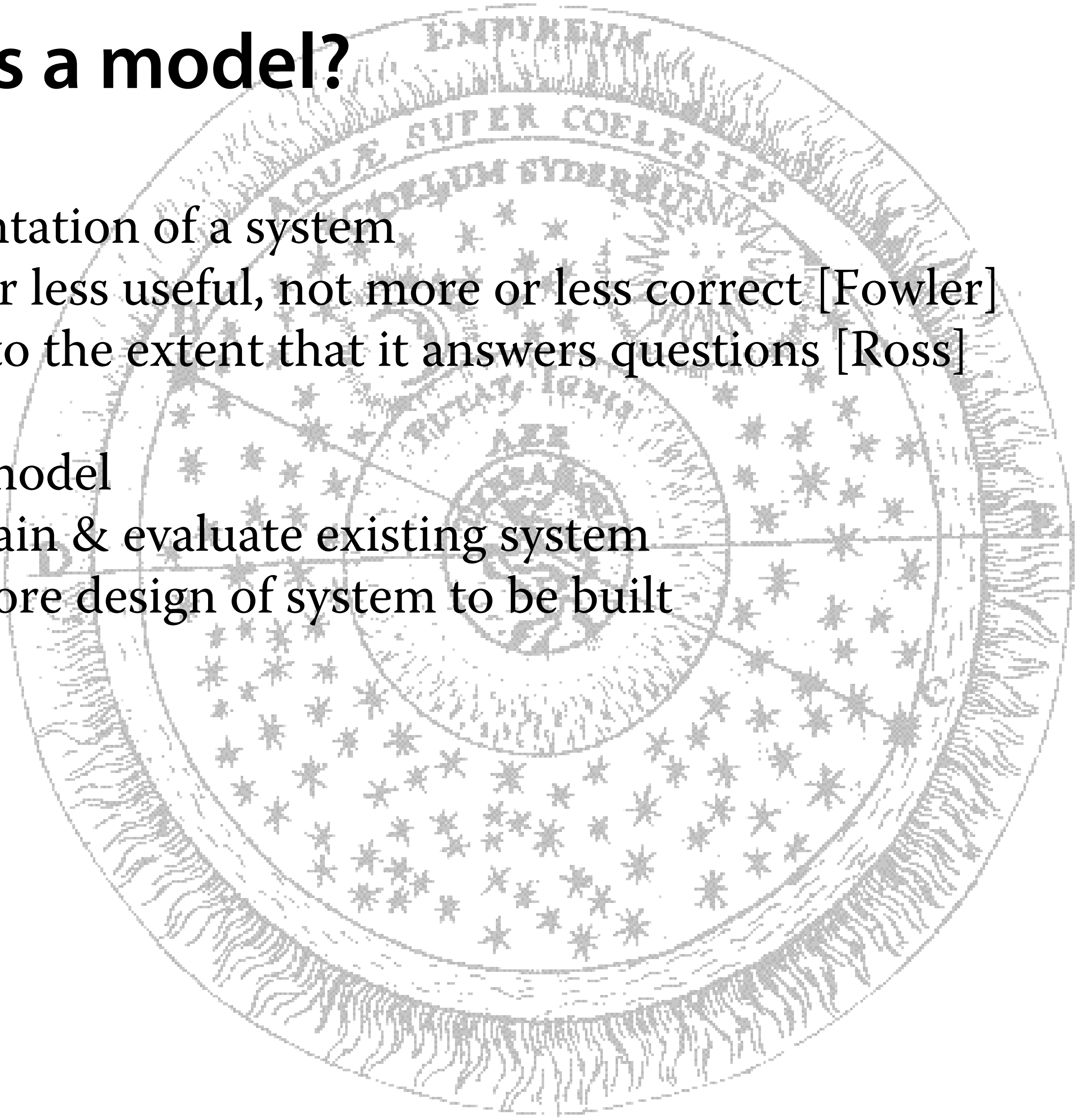
fix the numbering mechanism to handle

- multiple children
  - section and figure have parent chapter
- multiple parents
  - section has parent chapter and appendix

what is a model?

a representation of a system
- › more or less useful, not more or less correct [Fowler]
- › useful to the extent that it answers questions [Ross]

role of a model
- › to explain & evaluate existing system
- › to explore design of system to be built



why model?

‘plan to throw one away’ [Brooks]
- › 100 line model or 100k lines of code?
- › nasty surprises happen sooner

designs with clear conceptual models
- › easier to use and implement
- › allow delegation & division of labour

separation of concerns
- › conceptual flaws get mired in code
- › not a good use of testing

lightweight formal methods

elements
- small & simple notations
- partial models & analyses
- full automation

focus on risky aspects
- hard to get right, or to check
- structure-determining
- high cost of failure