Lecture 10
Reasoning about Z Specifications

David Garlan

Carnegie Mellon University

Models of Software Systems

15-671

Fall 1995

© Garlan, 1994

The Story Thus Far

- We have seen a brief intro to Z
- Main features
  - > use of schemas to encapsulate state
  - > use of schemas to describe operations
  - > the delta and xi conventions
  - > use of schema calculus to build up bigger operations out of smaller ones

Relating Birthday Book to State Machines

$S = \{known: NAME; birthday: NAME \pounds DATE \mid known = dom(birthday)\}$

$I = \{known: NAME; birthday: NAME \pounds DATE \mid birthday = \varnothing\}$

$A = \{AddBirthday(date?:DATE, name?:NAME),$
$FindBirthday(date?:DATE)/name!(NAME),$
$Remind(date?:DATE)/names!(P\ Name)\}$

$\delta = ...$

AddBirthday

AddBirthday(date?:DATE, name?:NAME)

pre name? $\notin$ known

post birthday' = birthday $\cup$ { name? ¢ date? }

FindBirthday(date?:DATE)/name!(NAME)

pre name? $/$ known

post birthday' = birthday

known' = known

name! = birthday(date?)

RAddBirthday

RAddBirthday(date?:DATE, name?:NAME)
report!(REPORT)

pre true

post

(name? ∉ known fl

birthday' = birthday ∪ { name? ¢ date? } fl

report = ok)

‡

(name? / known fl

birthday' = birthday fl

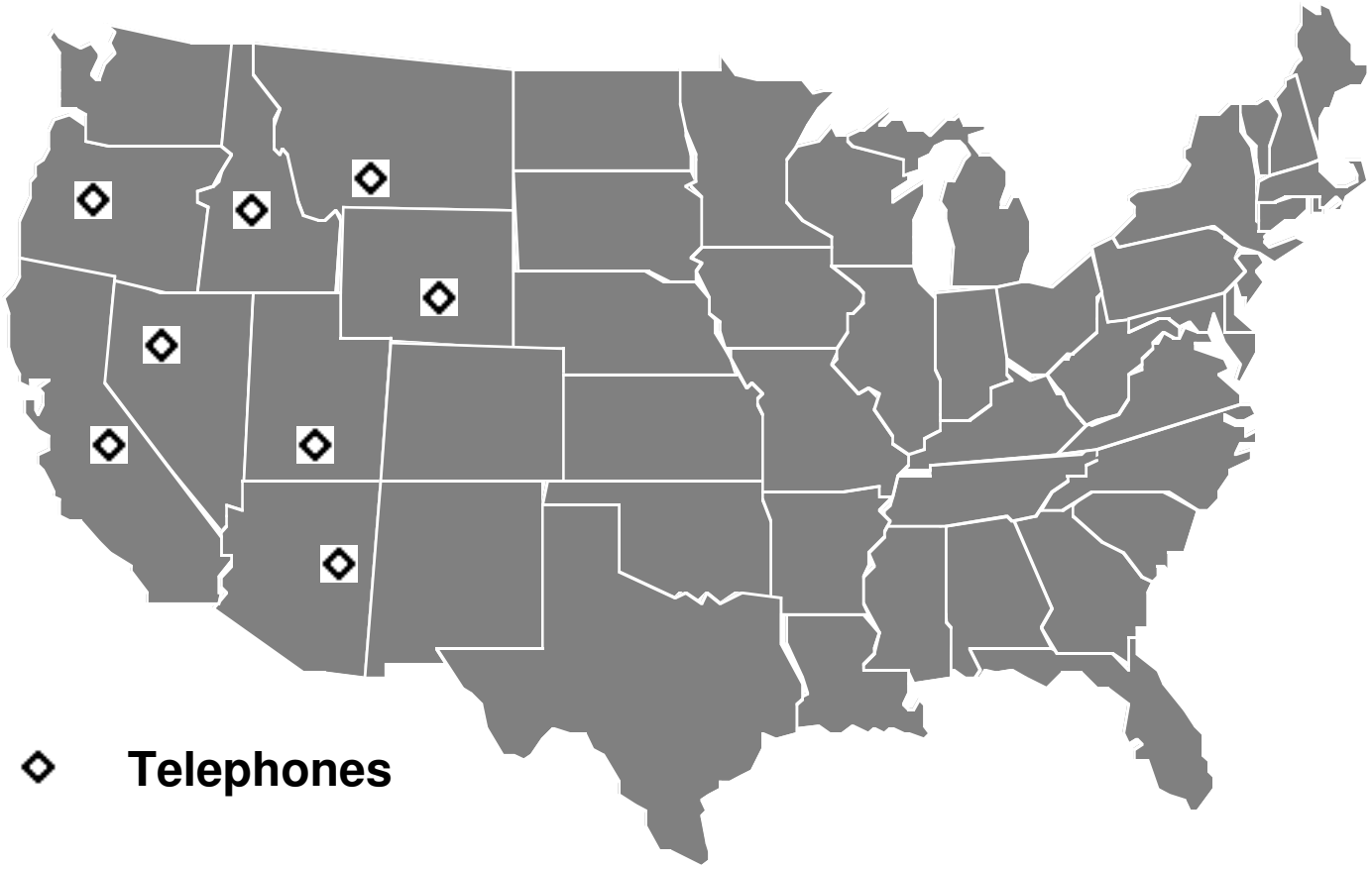
known'= knownfl

report = alreadyknown)

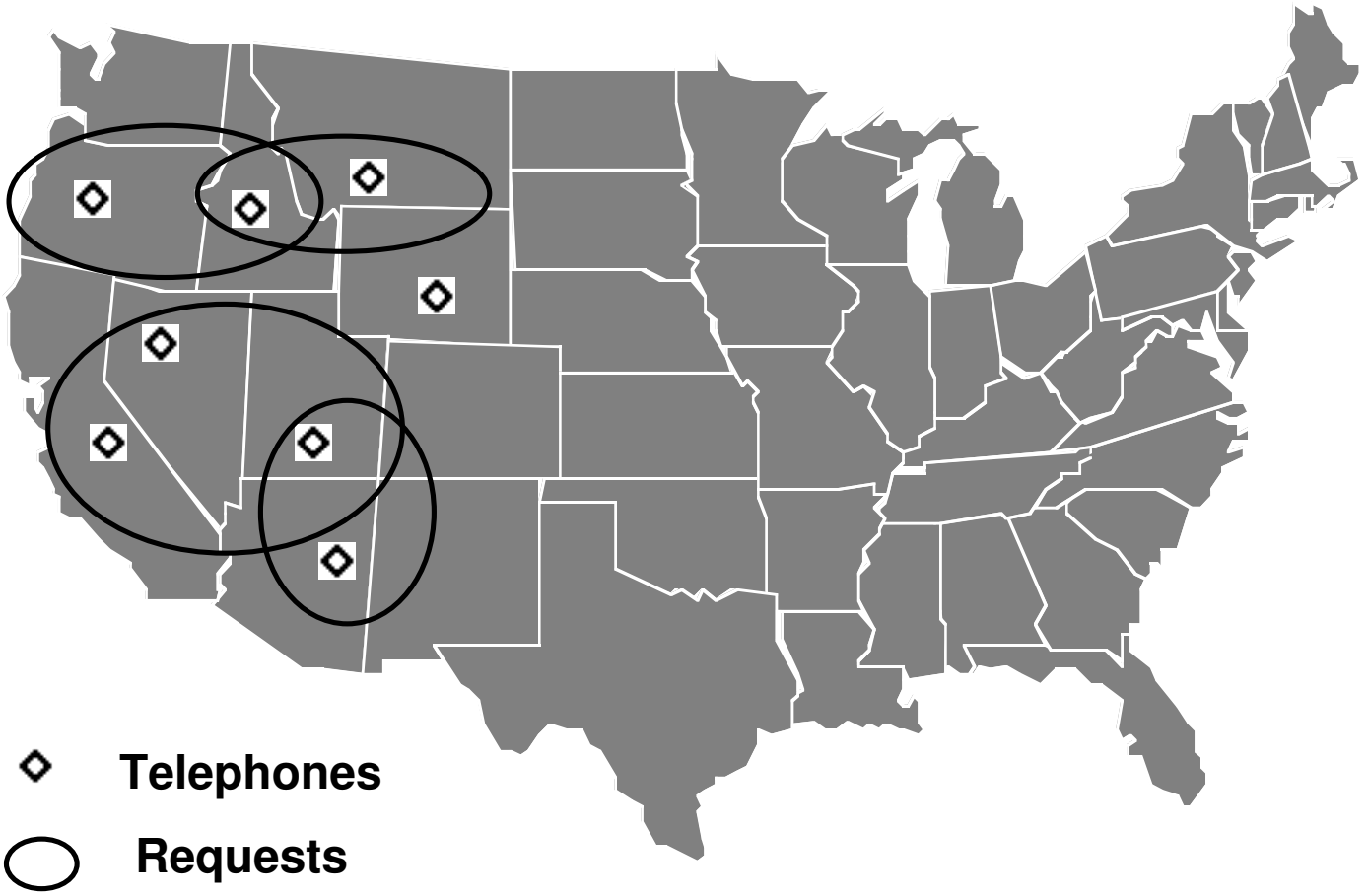
Lecture Outline

- **A Second Example: Telephone Net**
  - > Adapted from "Telephone Network", by Carroll Morgan
  - > In Specification Case Studies, Ian Hayes (ed)
- **Concepts**
  - > Use of strong state invariants to make concise models
  - > Use of schema calculus to simplify descriptions and associated reasoning
  - > Disj, (Generic Definitions)
  - > Framing
  - > Reasoning with state invariants

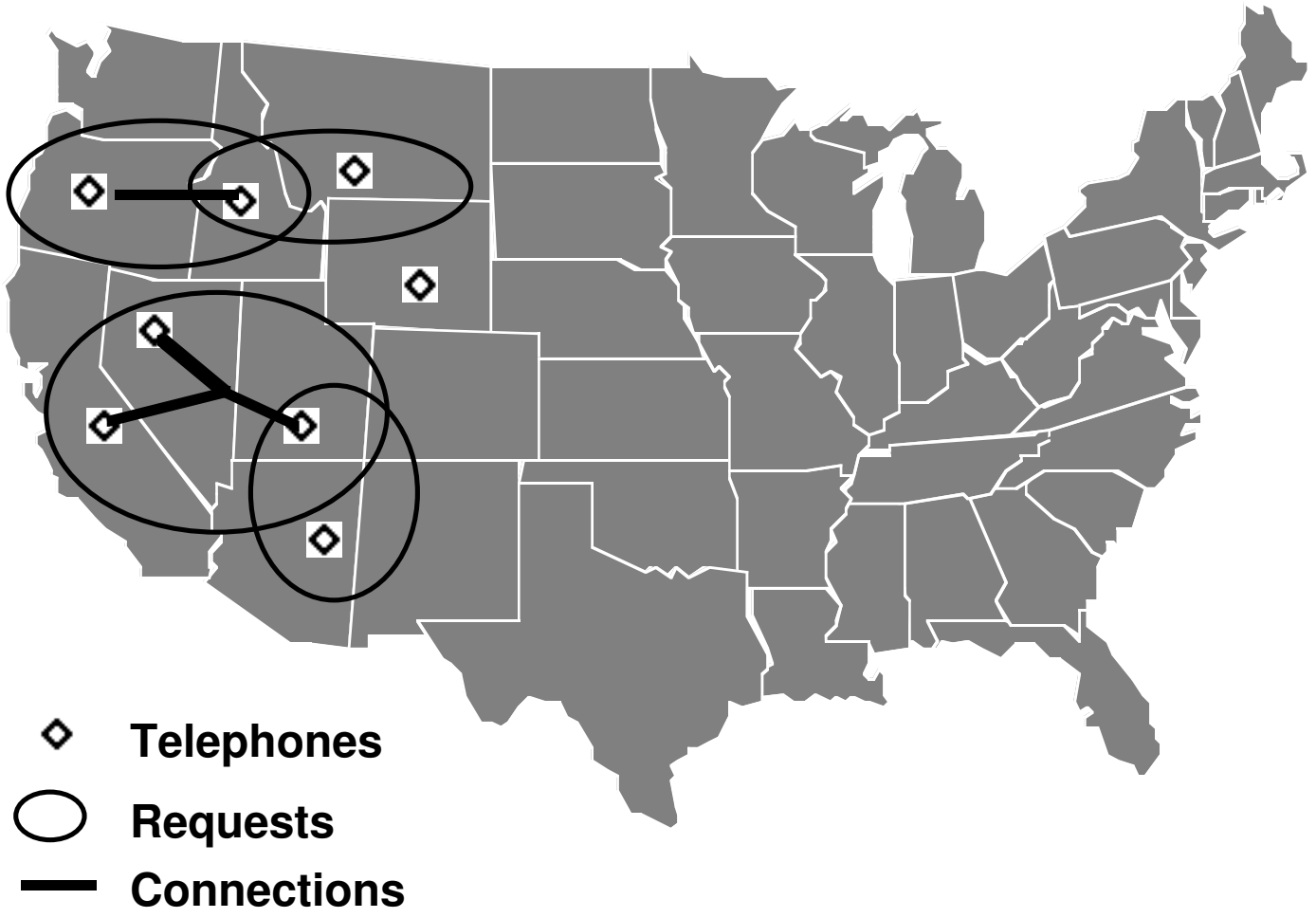
Telephones



Requests for Connection



Connections



Operations

- ***Call*: Request a connection between two phones**
- ***Hangup*: Terminate a connection**
- ***Busy*: Report whether a phone is busy**

State Space

$$[\,PHONE\,]$$

$$CONNECTION == \mathbb{P}\, PHONE$$

TelephoneNet

$$reqs, cons : \mathbb{P}\, CONNECTION$$

$$cons \subseteq reqs$$

$$disj\ cons$$

Example

PHONE = {1, 2, 3 }

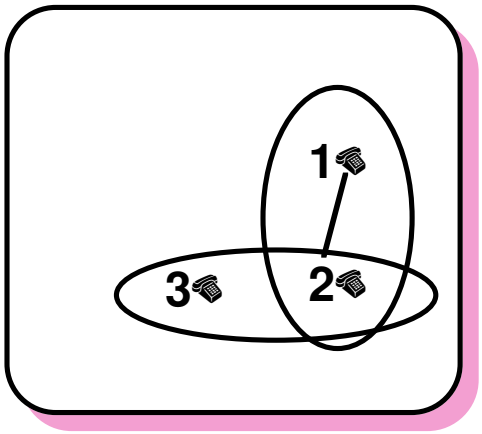
CONNECTIONS =

{ {}, {1}, {2}, {3}, {1,2} {1,3}, {2,3}, {1,2,3} }

TelephoneNet

reqs: { {1,2},{2,3} }

cons: { {1,2} }



Pairwise disjoint sets

Definition: (informal) A set of sets is disjoint iff no pair of those sets has elements in common.

Definition: (formal)

$$disj\ cons \Leftrightarrow$$

$$\forall\ c1, c2 : cons \bullet c1 \neq c2 \Rightarrow c1 \cap c2 = \varnothing$$

Examples: Which sets are *disj*?

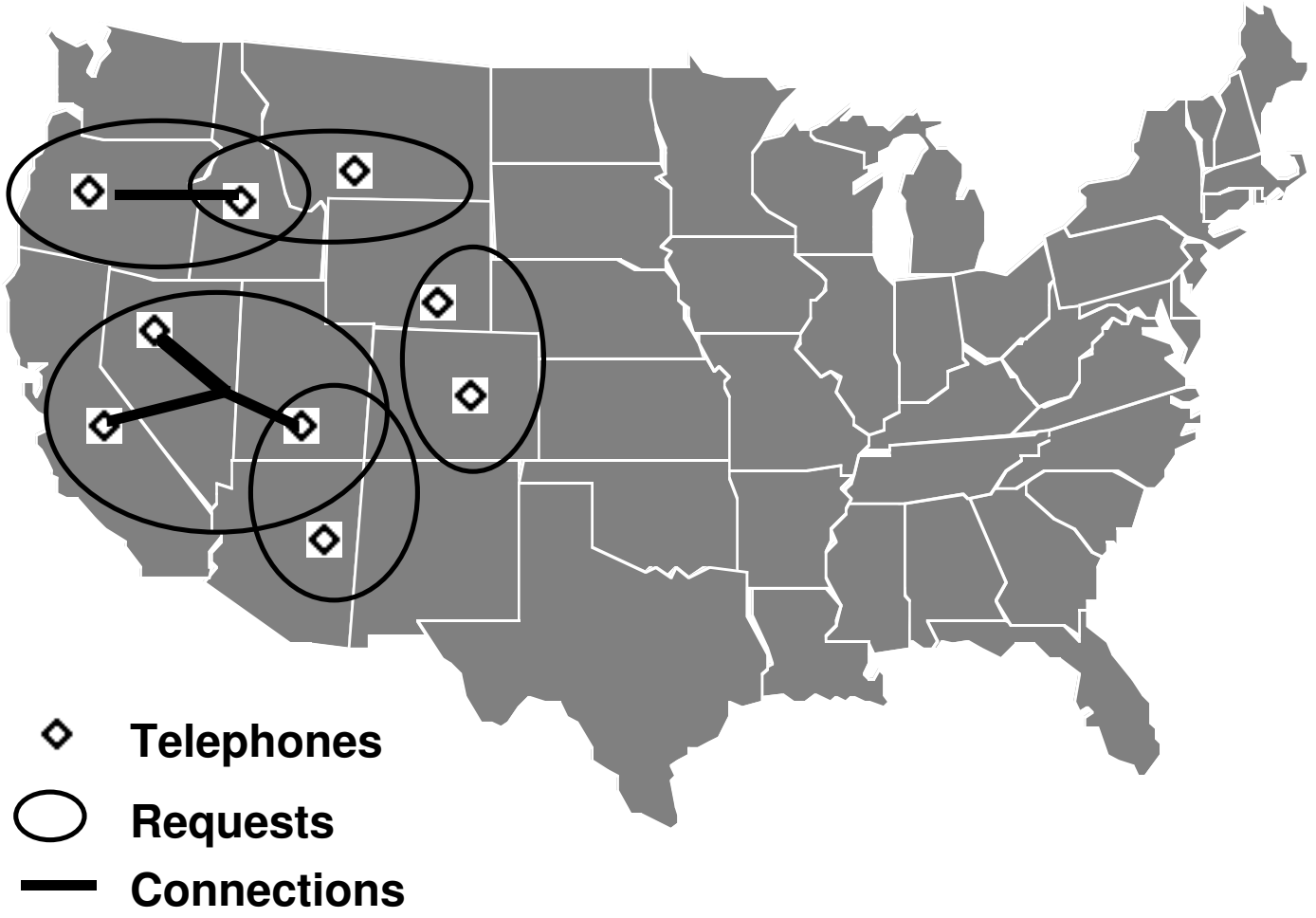
{ {a}, {b,c}, {d} }

{ {a,b}, {c,d}, {a,c} }

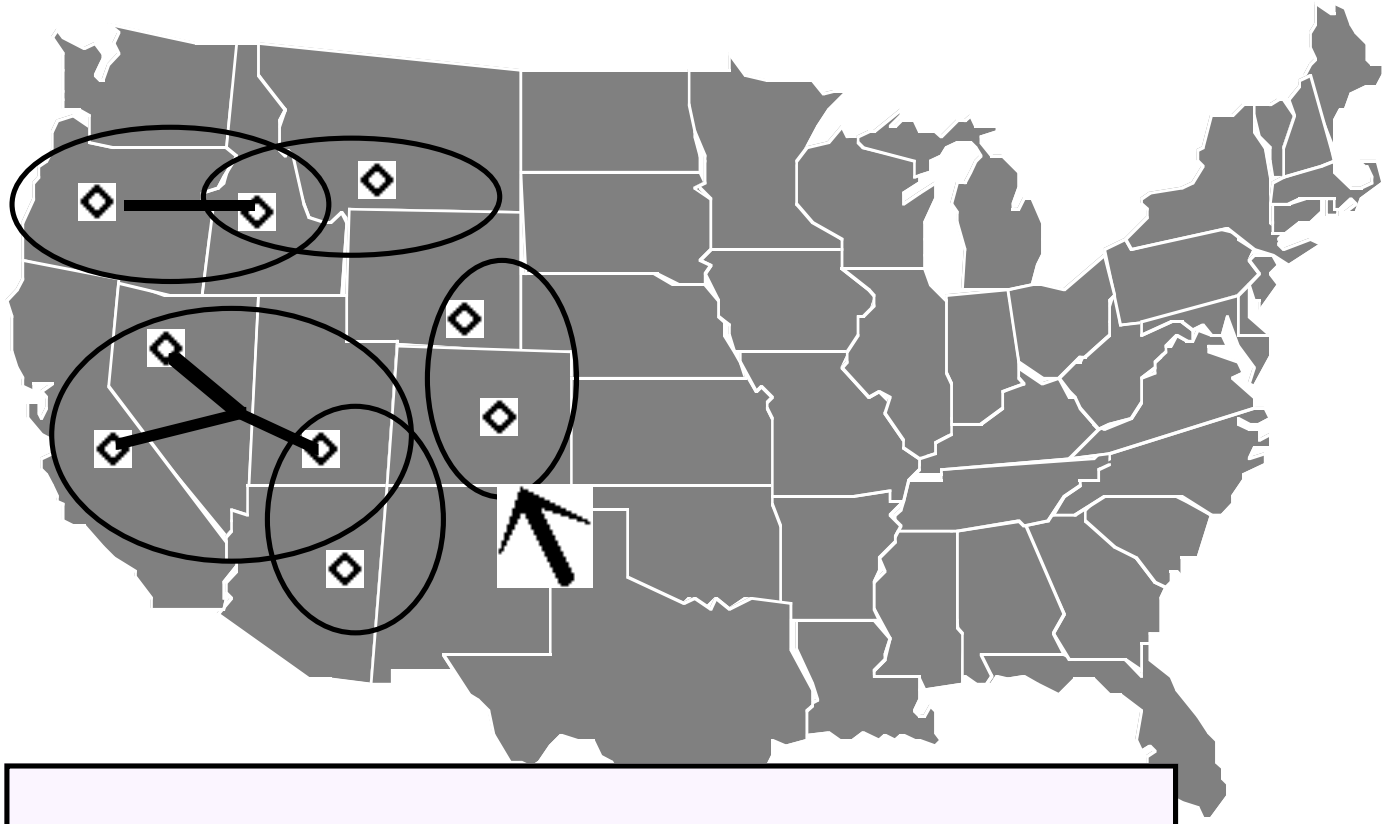
{ ∅, {a} }

{ {a,b}, {b, a}}

What's Wrong With This Net?



An Inefficient Net



InefficientNet :

$\exists$ r: reqs \ cons • disj (cons $\cup$ {r})

Efficient Networks

EfficientTN

TelephoneNet

$$\sim (\exists\ r: reqs \setminus cons \bullet disj\ (cons \cup \{r\}))$$

EfficientTN

$$reqs, cons: \mathbb{P}\ CONNECTION$$

$$cons \leq reqs$$

$$disj\ cons$$

$$\sim (\exists r: reqs \setminus cons \bullet disj\ (cons \cup \{r\}))$$

Initial Telephone Net

InitTN

EfficientTN'

reqs' = cons' = ∅

Framing: Event

Event

Δ **EfficientTN**

$\forall$ **c: CONNECTION •**

c $\in$ **(cons** $\cap$ **reqs')** $\Rightarrow$ **c** $\in$ **cons'**

This Really Means ...

Event

reqs, cons: $\mathbb{P}$ **PHONE**

reqs', cons': $\mathbb{P}$ **PHONE**

But we would not be expected to write it out this way.

---

cons reqs

***disj* cons**

cons' reqs'

***disj* cons'**

~(∃ r: reqs \ cons • disj (cons ∪ {r}))

~(∃ r: reqs' \ cons' • disj (cons' ∪ {r}))

∀ c: CONNECTION • c ∈ (cons ∩ reqs') ⇒ c ∈ cons'

Call

Call

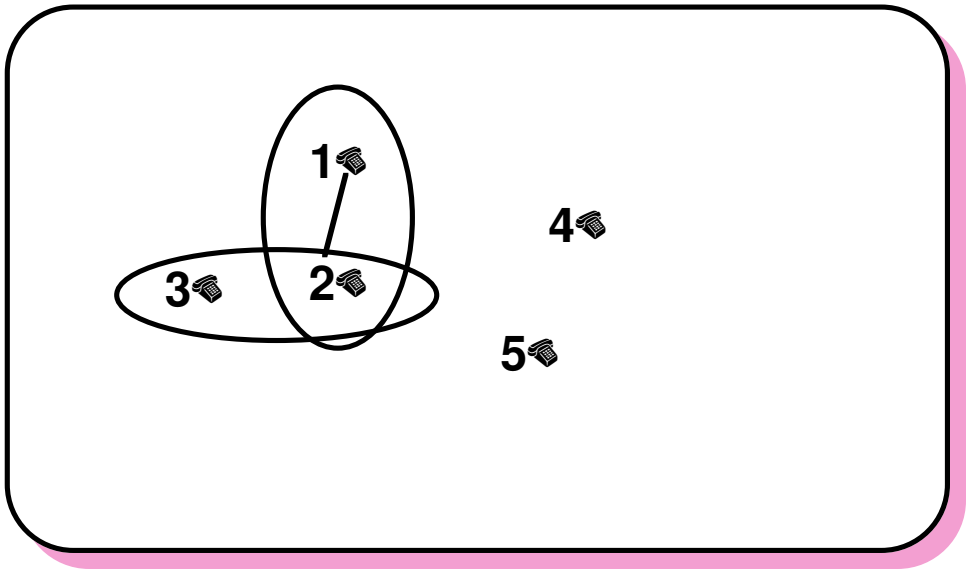
Event

ph?, dialled? : PHONE

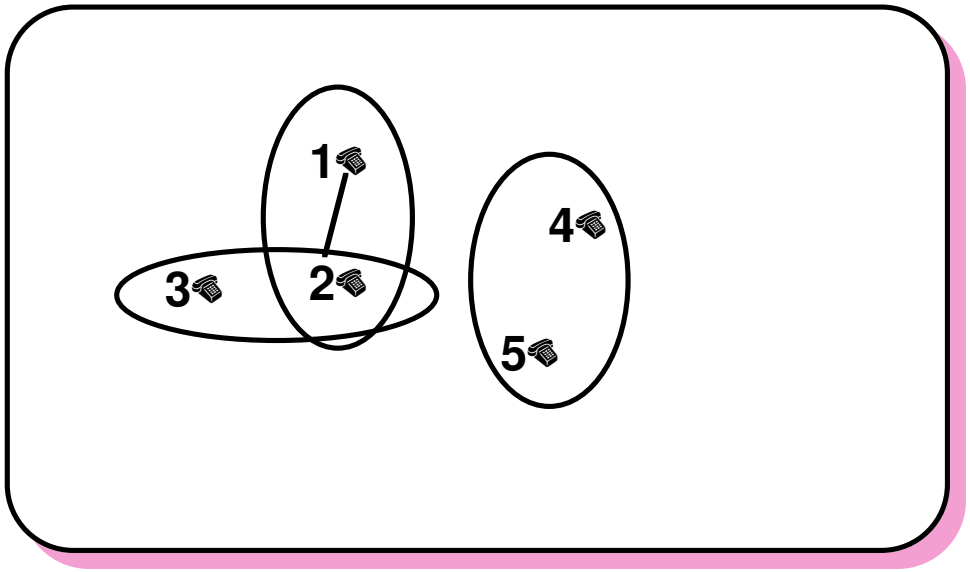
---

reqs' = reqs ∪ { {ph?, dialled?} }

Example



Example



Must the new request become a connection?

Hangup

Hangup

Event

ph? : PHONE

---

reqs' = reqs \ { c: cons | ph? ∈ c }

Is this realistic?

Busy

YesOrNo ::== Yes | No

Busy

Event

ph?: PHONE

busy!: YesOrNo

---

reqs' = reqs

busy! = Yes $\Leftrightarrow$ ph? $\in$ $\bigcup$ cons

What happens to cons?

Reasoning about the Specification

- **Theorem: (Informal) If an operation doesn't change any of the requests then it doesn't change any of the connections.**
- **Proof: (Informal)**
  1. **The constraint on Event tells us that any original connection won't be broken if it is still requested.**
  2. **If an operation doesn't change the requests, all original connections will still be requested.**
  3. **Hence, no connections are broken.**
  4. **Also, no new connections are added because if a connection could be added afterwards, it could have been added before.**
  5. **So connections aren't changed.**

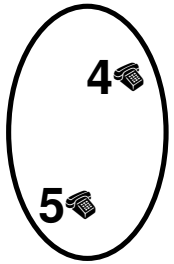
The Fuzz Type Checker

- One of the benefits of being precise about types is that we can type-check our descriptions
- We will use the Fuzz Type Checker to do this.
- It takes Latex descriptions of Z specifications as input and produces error reports and other useful type information.
- To type check a file spec.tex, invoke

  fuzz spec.tex
- To get type information about spec.tex, invoke

  fuzz -t spec.tex

Last Slide



Must the new request become a connection?

What happens to cons?