Reglog – the game

David I. Spivak
(joint with Brendan Fong)

2019/06/24

Compose :: Conference

Outline

Minority Report

Minority Report

The following has very little direct relation to the movie Minority Report; it's just an analogy for color.

Minority Report... regular logic style

The 2002 movie *Minority report* showed detective Tom Cruise playing seamlessly with logic.

- A computer database held relevant information.
- Cruise could pull it up and manipulate it to solve crimes.

Minority Report... regular logic style

The 2002 movie *Minority report* showed detective Tom Cruise playing seamlessly with logic.

- A computer database held relevant information.
- Cruise could pull it up and manipulate it to solve crimes.

Let's imagine our own version of a detective scenario.

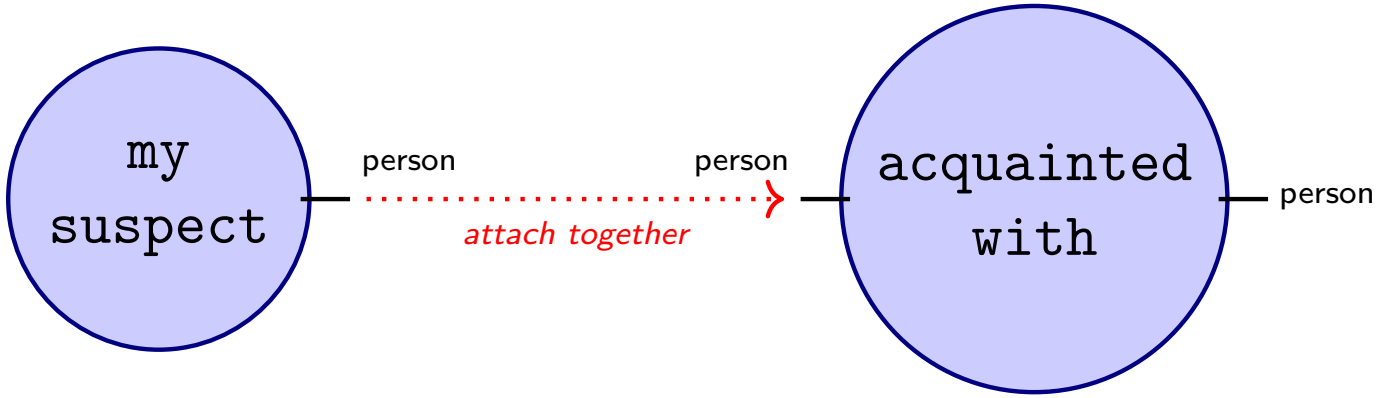
Brought to you by... regular logic – the game!

Working with logic

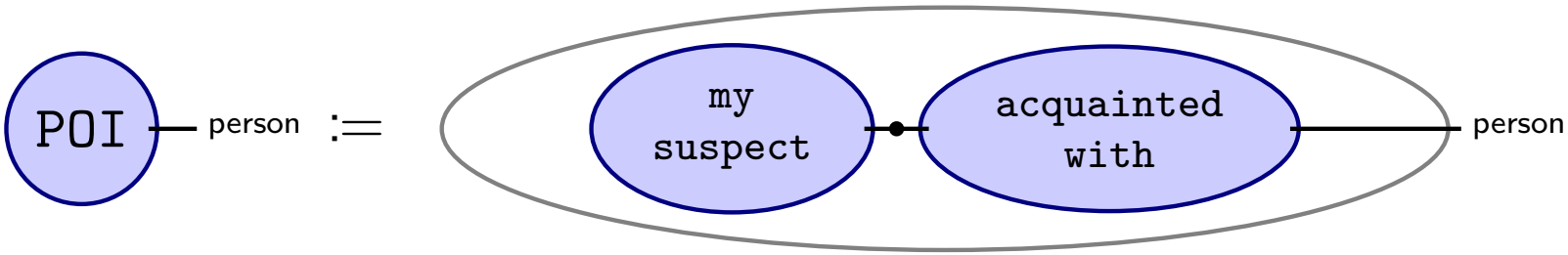
- Imagine you are an investigator on a case.
- You're adding to and narrowing down your set of suspects.
- You can pull up cells from the computer's database.

Working with logic

- Imagine you are an investigator on a case.
- You're adding to and narrowing down your set of suspects.
- You can pull up cells from the computer's database.

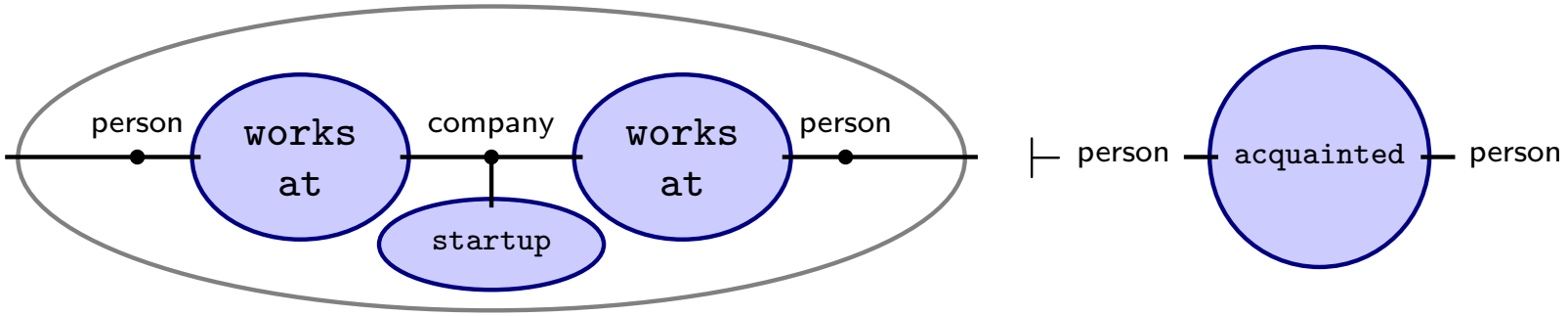


- and define `POI` (person of interest) as the result:



Adding beliefs

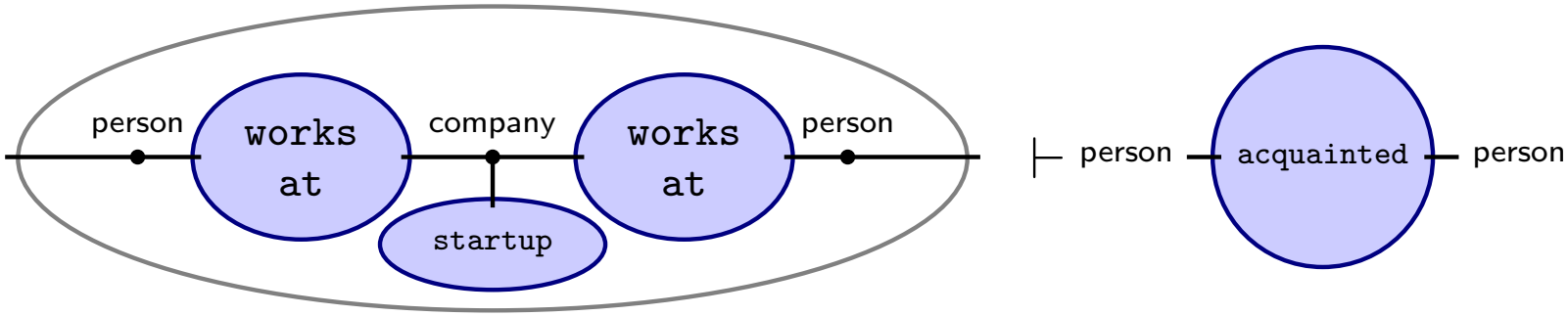
You can add beliefs about the world.



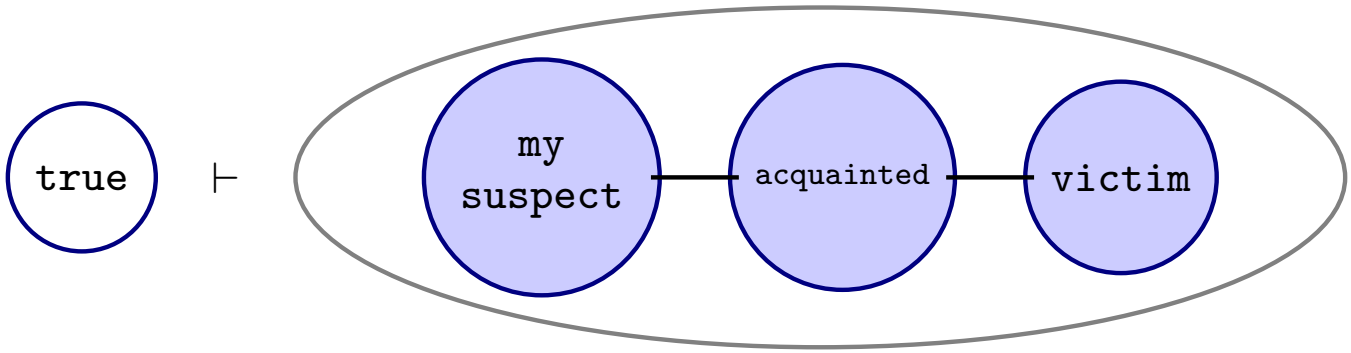
Belief: "If two persons work at the same startup, they are acquainted."

Adding beliefs

You can add beliefs about the world.



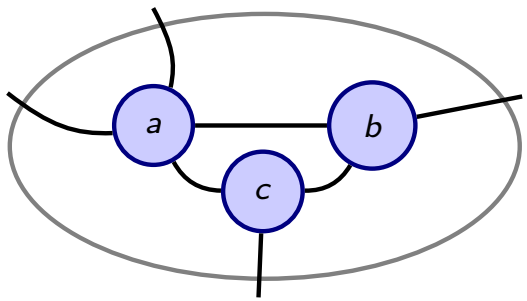
Belief: "If two persons work at the same startup, they are acquainted."



Belief: "In any case, my suspect is acquainted with the victim."

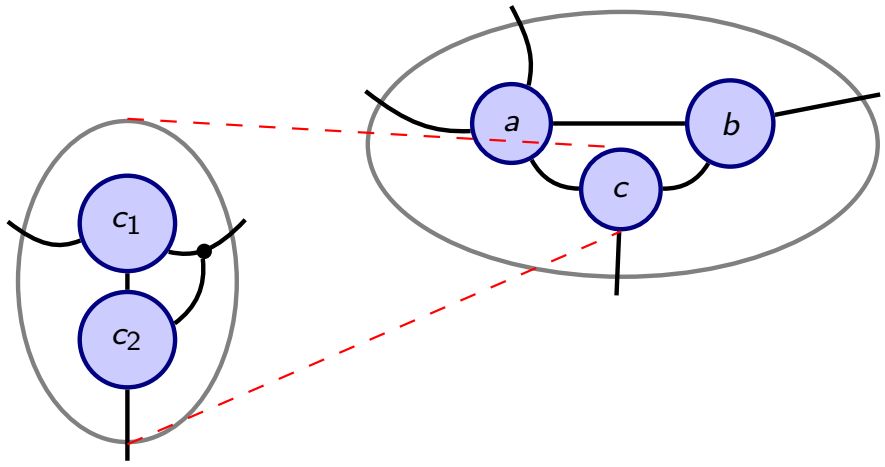
Compositionality

Of course, concepts can be arbitrarily nested.



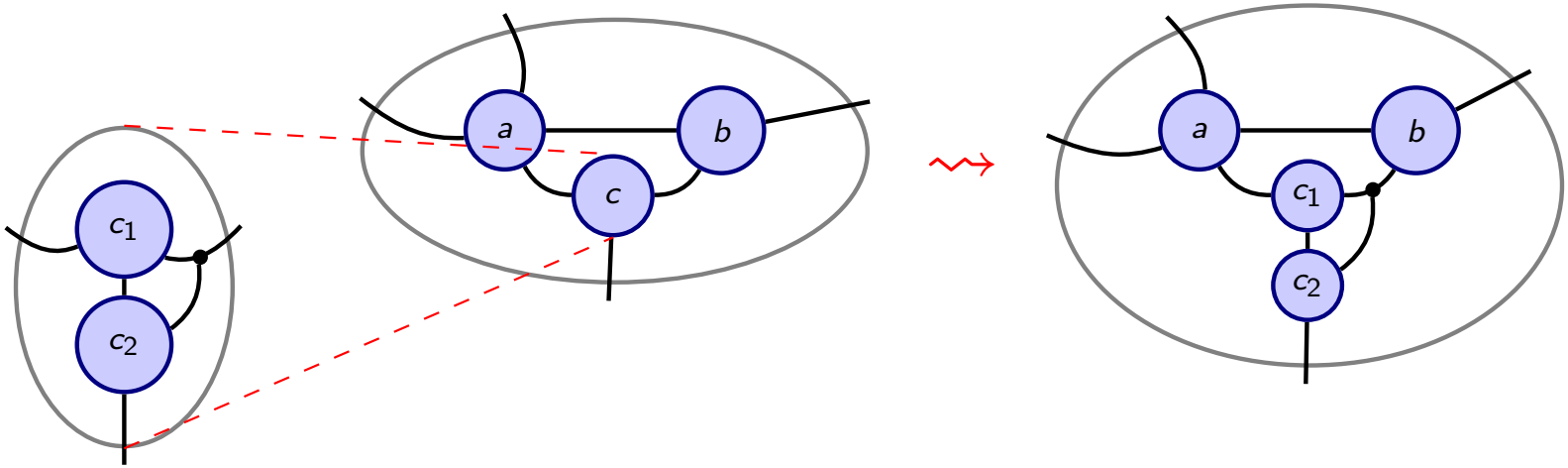
Compositionality

Of course, concepts can be arbitrarily nested.



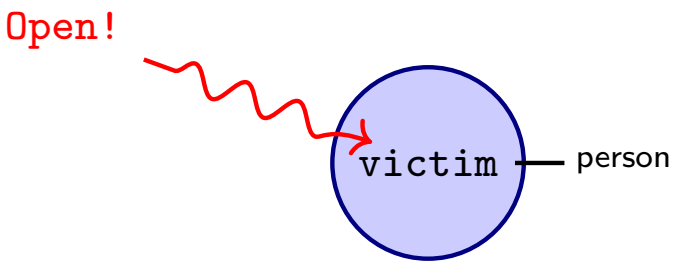
Compositionality

Of course, concepts can be arbitrarily nested.



Accessing data

You can click a cell to see what's inside:

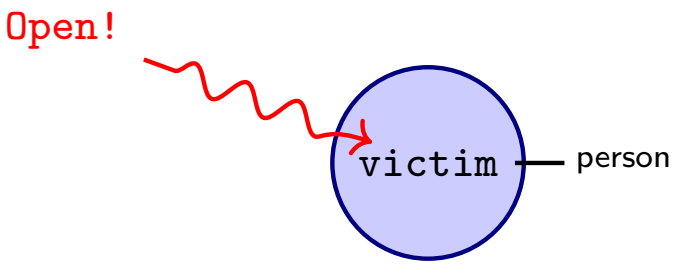


victim
person
P105
P820

Persons are internal identifiers; we want to see facts about the victims.

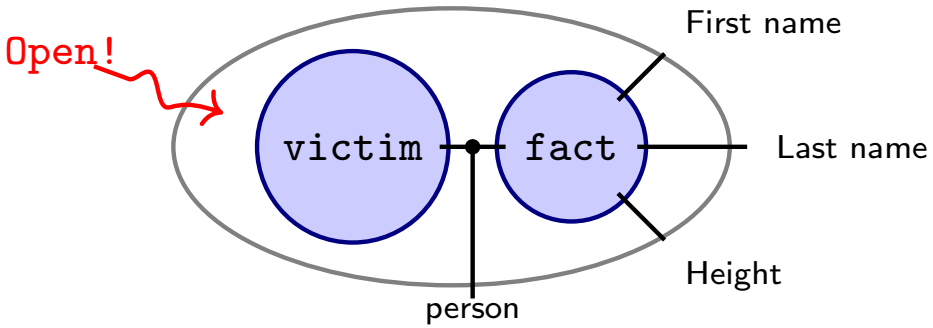
Accessing data

You can click a cell to see what's inside:



victim
person
P105
P820

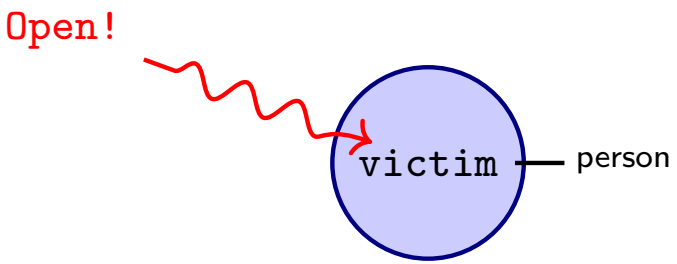
Persons are internal identifiers; we want to see facts about the victims.



victim fact			
person	First	Last	Height
P105	Alice	Hu	5'10" – 6'00"
P820	Bob	Barr	P820.Height

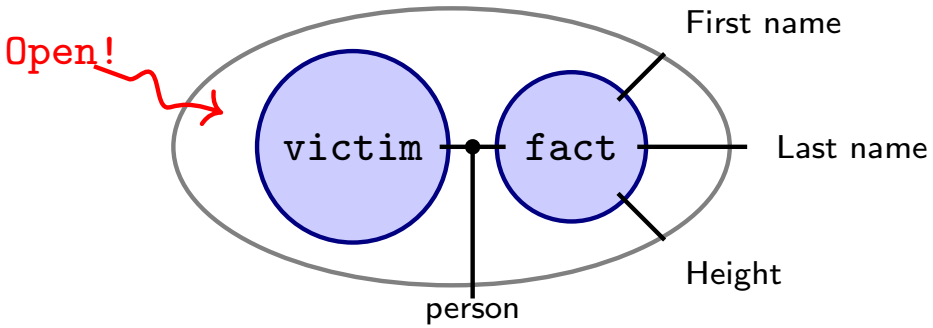
Accessing data

You can click a cell to see what's inside:



victim
person
P105
P820

Persons are internal identifiers; we want to see facts about the victims.

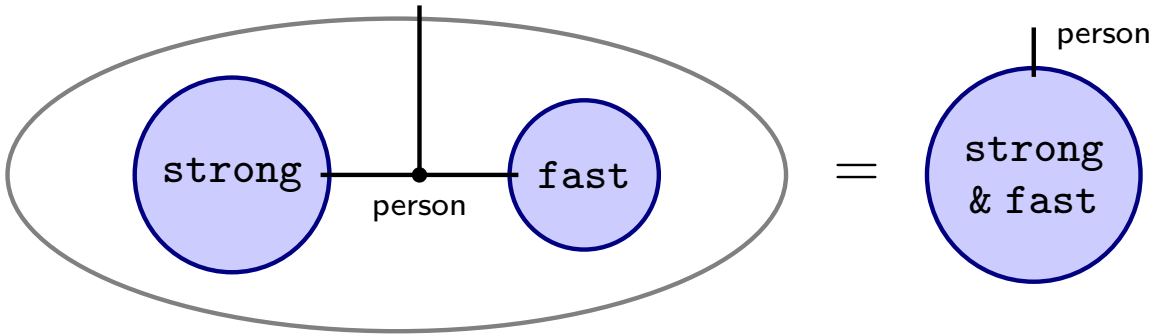


victim fact			
person	First	Last	Height
P105	Alice	Hu	5'10" – 6'00"
P820	Bob	Barr	P820.Height

Some knowledge may be missing or otherwise imperfect.

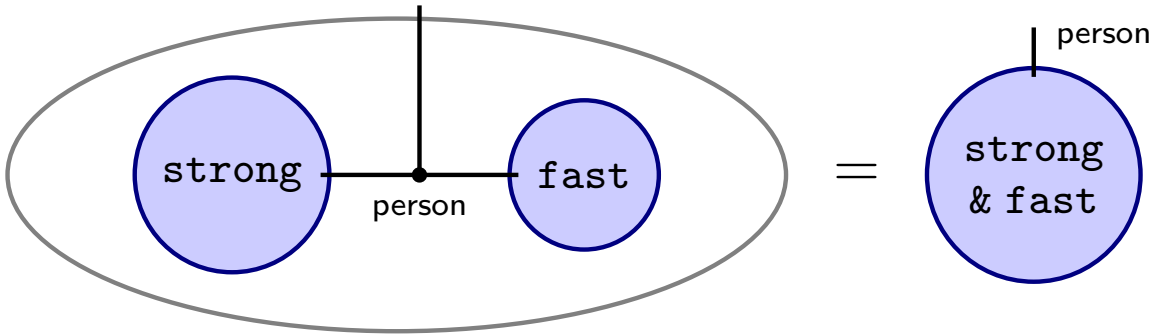
Reasoning

The machine knows basic logical reasoning.



Reasoning

The machine knows basic logical reasoning.

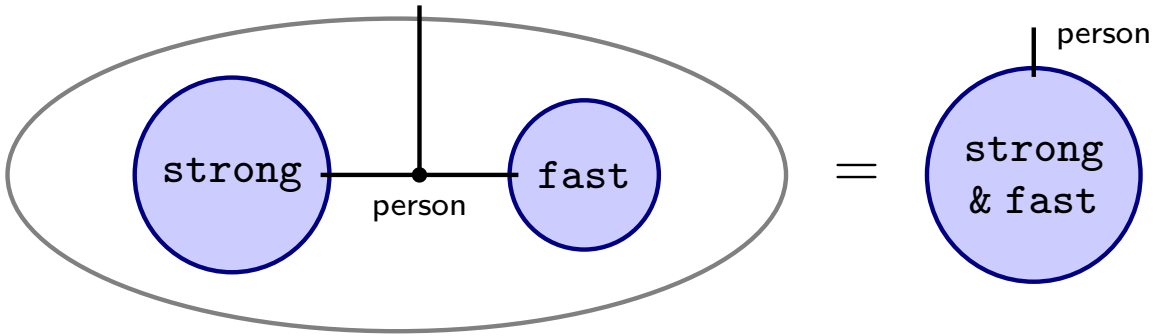


You can manipulate diagrams by ...

- ... combining or breaking up intersectionalities as above;

Reasoning

The machine knows basic logical reasoning.



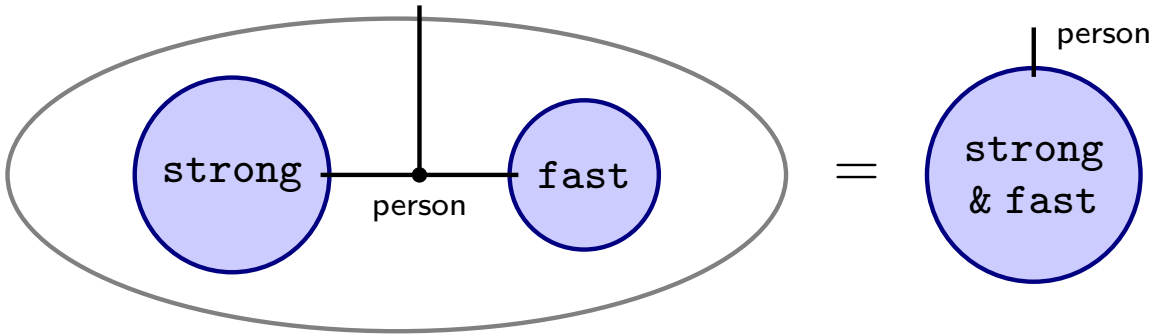
You can manipulate diagrams by ...

- ... combining or breaking up intersectionalities as above;
- ... breaking dots to drop equality constraints:

$$\{(x, y, z) \mid x = y = z\} \subseteq \{(x, y, z) \mid x = y\}$$

Reasoning

The machine knows basic logical reasoning.



You can manipulate diagrams by ...

- ... combining or breaking up intersectionalities as above;
- ... breaking dots to drop equality constraints:

$$\{(x, y, z) \mid x = y = z\} \subseteq \{(x, y, z) \mid x = y\}$$

Reasoning: regular "old" logic, with a shiny new math-specified GUI.

The chase is on

You've identified certain sources of and constraints on your suspect

- Sue is a suspect.
- The suspect is acquainted with the victim.
- (and so on.)

The chase is on

You've identified certain sources of and constraints on your suspect

- Sue is a suspect.
- The suspect is acquainted with the victim.
- (and so on.)

To catch your suspect, you must chase...

The chase is on

You've identified certain sources of and constraints on your suspect

- Sue is a suspect.
- The suspect is acquainted with the victim.
- (and so on.)

To catch your suspect, you must chase...

- This is the name of an algorithm from database theory.
- Start with a database instance I with knowns and unknowns.

The chase is on

You've identified certain sources of and constraints on your suspect

- Sue is a suspect.
- The suspect is acquainted with the victim.
- (and so on.)

To catch your suspect, you must chase...

- This is the name of an algorithm from database theory.
- Start with a database instance I with knowns and unknowns.
- Add axioms such as:

  Every two people who work at a startup together were acquainted at some time.

The chase is on

You've identified certain sources of and constraints on your suspect

- Sue is a suspect.
- The suspect is acquainted with the victim.
- (and so on.)

To catch your suspect, you must chase...

- This is the name of an algorithm from database theory.
- Start with a database instance I with knowns and unknowns.
- Add axioms such as:

  Every two people who work at a startup together were acquainted at some time.

  $$\forall(p, p' : P).\, \exists(c : C).\, W(p, c) \wedge W(p', c) \wedge S(c) \vdash \exists(t : T).\, A(t, p, p').$$

The chase is on

You've identified certain sources of and constraints on your suspect

- Sue is a suspect.
- The suspect is acquainted with the victim.
- (and so on.)

To catch your suspect, you must chase...

- This is the name of an algorithm from database theory.
- Start with a database instance I with knowns and unknowns.
- Add axioms such as:

  Every two people who work at a startup together were acquainted at some time.

  $$\forall(p, p' : P).\, \exists(c : C).\, W(p, c) \wedge W(p', c) \wedge S(c) \vdash \exists(t : T).\, A(t, p, p').$$

- Axioms AKA: regular logic sequents, embedded dependencies, existential horn clauses, lifting problems.

The chase is on

You've identified certain sources of and constraints on your suspect

- Sue is a suspect.
- The suspect is acquainted with the victim.
- (and so on.)

To catch your suspect, you must chase...

- This is the name of an algorithm from database theory.
- Start with a database instance I with knowns and unknowns.
- Add axioms such as:

  Every two people who work at a startup together were acquainted at some time.

  $$\forall(p, p' : P).\, \exists(c : C).\, W(p, c) \wedge W(p', c) \wedge S(c) \vdash \exists(t : T).\, A(t, p, p').$$

- Axioms AKA: regular logic sequents, embedded dependencies, existential horn clauses, lifting problems.

The chase minimally "repairs" $I \to I'$, with I' conforming to axioms.

Let's hook it up

To complete the detective story:

- You have created an important cell: locations the suspect may be in.
- You connect its location port to your autonomous car, and off you go.

Let's hook it up

To complete the detective story:

- You have created an important cell: locations the suspect may be in.
- You connect its location port to your autonomous car, and off you go.
- You find the suspect and get promoted to Tom Cruise.

Let's hook it up

To complete the detective story:

- You have created an important cell: locations the suspect may be in.
- You connect its location port to your autonomous car, and off you go.
- You find the suspect and get promoted to Tom Cruise.

The End.

Let's hook it up

To complete the detective story:

- You have created an important cell: locations the suspect may be in.
- You connect its location port to your autonomous car, and off you go.
- You find the suspect and get promoted to Tom Cruise.

The End.

Can we make this real?

Plan for rest of talk

"Detective" is not the only game in reglog – the game.

- I'll briefly discuss the mathematics involved.
- I'll talk about how the math connects to the GUI described above.
- I'll end by describing several other games, besides "detective".

Plan for rest of talk

"Detective" is not the only game in reglog – the game.

- I'll briefly discuss the mathematics involved.
- I'll talk about how the math connects to the GUI described above.
- I'll end by describing several other games, besides "detective".

To keep in mind: *Category Theory as specification language*

- Category theory helped in the development of Haskell, etc.
- But it can also be used to specify DSLs.

Plan for rest of talk

"Detective" is not the only game in reglog – the game.

- I'll briefly discuss the mathematics involved.
- I'll talk about how the math connects to the GUI described above.
- I'll end by describing several other games, besides "detective".

To keep in mind: *Category Theory as specification language*

- Category theory helped in the development of Haskell, etc.
- But it can also be used to specify DSLs.
- E.g. Tom Ellis' Haskell package *Opaleye* is based on similar work.
- Today's talk is a generalization of that story.

Outline

Regular logic and regular categories

There is a strong connection between category theory and logic.

Regular logic and regular categories

There is a strong connection between category theory and logic.

- A regular category is a category $\mathcal{R}$ with
  - finite limits (terminal object 1 and pullbacks), and
  - pullback-stable image factorizations.

Regular logic and regular categories

There is a strong connection between category theory and logic.

- A regular category is a category $\mathcal{R}$ with
  - finite limits (terminal object 1 and pullbacks), and
  - pullback-stable image factorizations.
- Regular logic is the internal logic of regular categories.

Regular logic and regular categories

There is a strong connection between category theory and logic.

- A regular category is a category $\mathcal{R}$ with
  - finite limits (terminal object 1 and pullbacks), and
  - pullback-stable image factorizations.
- Regular logic is the internal logic of regular categories.
- Examples of regular categories:
  - Set, and more generally any topos;
  - $\mathrm{Set}^{\mathrm{op}}$, opposite of any topos, also $\mathrm{TopSp}^{\mathrm{op}}$;
  - The category of models of any Lawvere theory (Groups, Rings, ...);
  - The slice (also the coslice) of any regular category over any object;
  - Exponential ideal: if $\mathcal{R}$ regular and $\mathcal{C}$ a category, then $\mathcal{R}^{\mathcal{C}}$ is regular.

Regular categories have well-behaved relations

Regular categories $\mathcal{R}$ are those with a good *2-category of relations*.

- A relation in $\mathcal{R}$ is a subobject $S \subseteq A \times B$.
- When $\mathcal{R}$ is regular, pullbacks and images play nicely...

Regular categories have well-behaved relations

Regular categories $\mathcal{R}$ are those with a good *2-category of relations*.

- A relation in $\mathcal{R}$ is a subobject $S \subseteq A \times B$.
- When $\mathcal{R}$ is regular, pullbacks and images play nicely...
- ... so that relations form a locally-posetal 2-category $\mathbb{R}\mathrm{el}_{\mathcal{R}}$.
  - That is, relations can be composed (WDs) and compared ($\vdash$).
  - One can recover $\mathcal{R}$ as the category of adjunctions in $\mathbb{R}\mathrm{el}_{\mathcal{R}}$!

Regular categories have well-behaved relations

Regular categories $\mathcal{R}$ are those with a good *2-category of relations*.

- A relation in $\mathcal{R}$ is a subobject $S \subseteq A \times B$.
- When $\mathcal{R}$ is regular, pullbacks and images play nicely...
- ... so that relations form a locally-posetal 2-category $\mathbb{R}\mathrm{el}_{\mathcal{R}}$.
  - That is, relations can be composed (WDs) and compared ($\vdash$).
  - One can recover $\mathcal{R}$ as the category of adjunctions in $\mathbb{R}\mathrm{el}_{\mathcal{R}}$!
- Every novice category theorist should prove to themselves that Set is the category of adjunctions in Rel.

Regular categories have well-behaved relations

Regular categories $\mathcal{R}$ are those with a good *2-category of relations*.

- A relation in $\mathcal{R}$ is a subobject $S \subseteq A \times B$.
- When $\mathcal{R}$ is regular, pullbacks and images play nicely...
- ... so that relations form a locally-posetal 2-category $\mathbb{R}\mathrm{el}_{\mathcal{R}}$.
  - That is, relations can be composed (WDs) and compared ($\vdash$).
  - One can recover $\mathcal{R}$ as the category of adjunctions in $\mathbb{R}\mathrm{el}_{\mathcal{R}}$!
- Every novice category theorist should prove to themselves that Set is the category of adjunctions in Rel.

Regular categories have enough structure to do regular logic.

Graphical regular logic

Think of regular categories as having good notion of *relation*.

Graphical regular logic

Think of regular categories as having good notion of *relation*.

- Regular logic is the logic of relations in regular categories.

Graphical regular logic

Think of regular categories as having good notion of *relation*.

- Regular logic is the logic of relations in regular categories.
- Given rel'ns $R(x, y)$ and $S(y, z)$, can interpret: $\exists y.\, R(x, y) \wedge S(y, z)$.

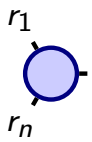
Graphical regular logic

Think of regular categories as having good notion of *relation*.

- Regular logic is the logic of relations in regular categories.
- Given rel'ns $R(x, y)$ and $S(y, z)$, can interpret: $\exists y.\, R(x, y) \wedge S(y, z)$.
  - The AND ($\wedge$), together with variable sharing, is given by pullback.
  - The EXISTS ($\exists y$) quantifier is given by taking an image.
  - The result $\exists y.\, R(x, y) \wedge S(y, z)$ is the *composite* $S \circ R$.

Graphical regular logic

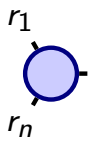
Think of regular categories as having good notion of *relation*.

- Regular logic is the logic of relations in regular categories.
- Given rel'ns $R(x, y)$ and $S(y, z)$, can interpret: $\exists y.\, R(x, y) \wedge S(y, z)$.
  - The AND ($\wedge$), together with variable sharing, is given by pullback.
  - The EXISTS ($\exists y$) quantifier is given by taking an image.
  - The result $\exists y.\, R(x, y) \wedge S(y, z)$ is the *composite* $S \circ R$.
- Graphical approach: suppose $\mathcal{R}$ is a regular category.
  - Have a shell 

for every *context* $\Gamma = (r_1, \ldots, r_n)$, where $r_i \in \mathcal{R}$.

Graphical regular logic

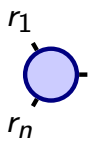
Think of regular categories as having good notion of *relation*.

- Regular logic is the logic of relations in regular categories.
- Given rel'ns $R(x,y)$ and $S(y,z)$, can interpret: $\exists y.\, R(x,y) \wedge S(y,z)$.
  - The AND ($\wedge$), together with variable sharing, is given by pullback.
  - The EXISTS ($\exists y$) quantifier is given by taking an image.
  - The result $\exists y.\, R(x,y) \wedge S(y,z)$ is the *composite* $S \circ R$.
- Graphical approach: suppose $\mathcal{R}$ is a regular category.
  - Have a shell 

 for every *context* $\Gamma = (r_1, \ldots, r_n)$, where $r_i \in \mathcal{R}$.
  - Have a cell (filling shell Γ) for every subobject $c \subseteq r_1 \times \cdots \times r_n$.
  - Wiring diagrams denote combinations of finite limits and images.

Graphical regular logic

Think of regular categories as having good notion of *relation*.

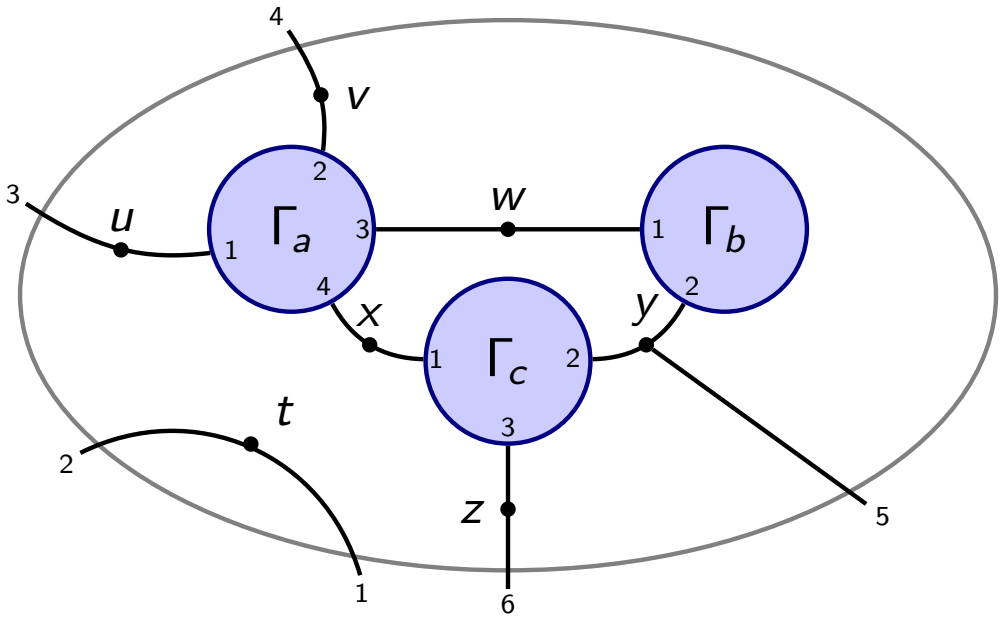
- Regular logic is the logic of relations in regular categories.
- Given rel'ns $R(x, y)$ and $S(y, z)$, can interpret: $\exists y.\, R(x, y) \wedge S(y, z)$.
  - The AND ($\wedge$), together with variable sharing, is given by pullback.
  - The EXISTS ($\exists y$) quantifier is given by taking an image.
  - The result $\exists y.\, R(x, y) \wedge S(y, z)$ is the *composite* $S \circ R$.
- Graphical approach: suppose $\mathcal{R}$ is a regular category.
  - Have a shell 

 for every *context* $\Gamma = (r_1, \ldots, r_n)$, where $r_i \in \mathcal{R}$.
  - Have a cell (filling shell Γ) for every subobject $c \subseteq r_1 \times \cdots \times r_n$.
  - Wiring diagrams denote combinations of finite limits and images.

Let's discuss wiring diagrams.

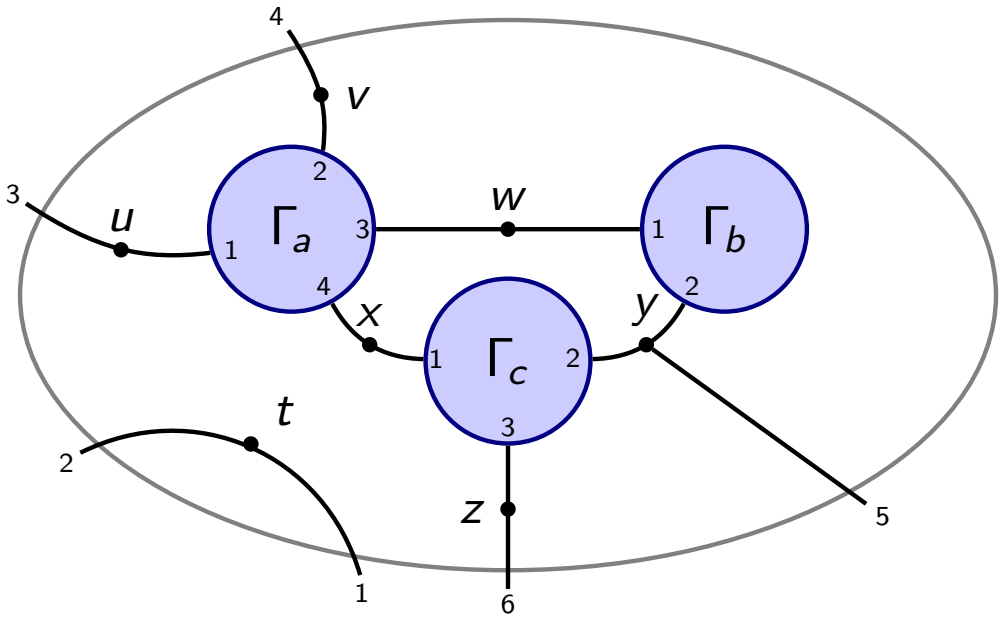
Mathematical specification of wiring diagrams

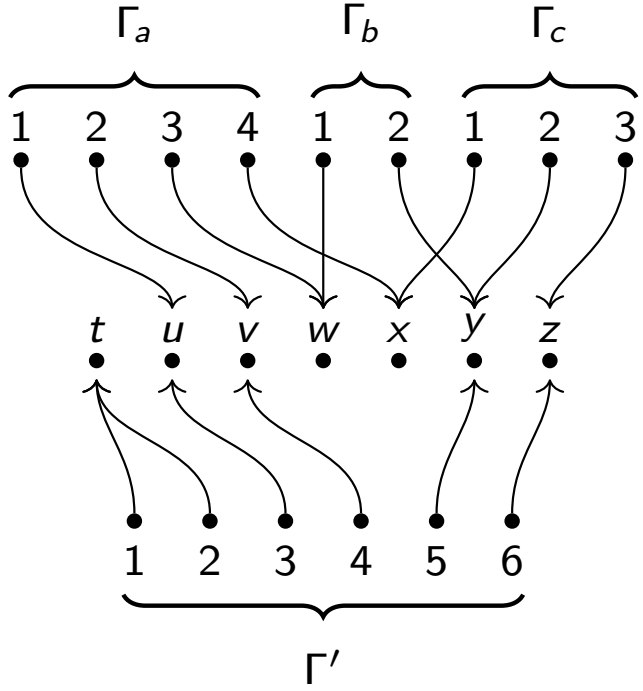
Mathematically, what is a wiring diagram $(\Gamma_a, \Gamma_b, \Gamma_c) \to \Gamma'$?



Mathematical specification of wiring diagrams

Mathematically, what is a wiring diagram $(\Gamma_a, \Gamma_b, \Gamma_c) \to \Gamma'$?





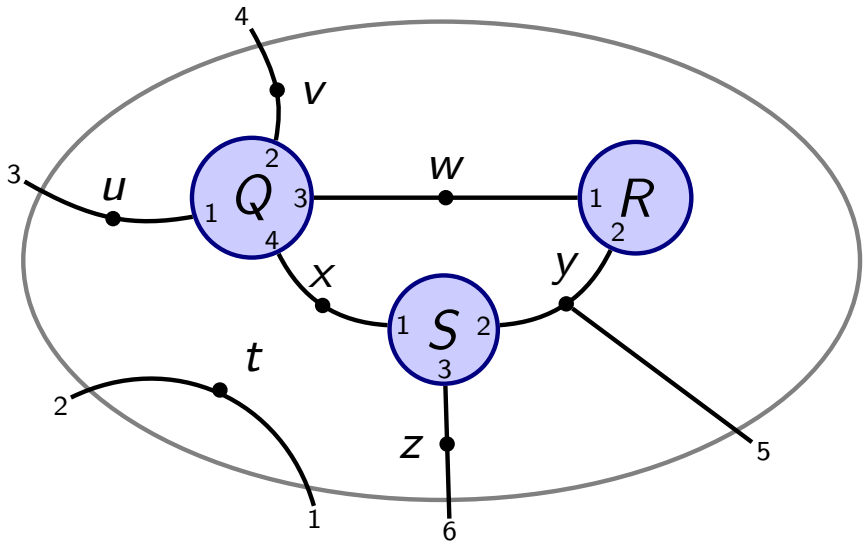
It is a morphism $(\Gamma_a, \Gamma_b, \Gamma_c) \to \Gamma'$ in the operad of cospans.

- That is, it's a *cospan*: a pair of functions with common codomain

$$\Gamma_a \sqcup \Gamma_b \sqcup \Gamma_c \to \{t, \ldots, z\} \leftarrow \Gamma'$$

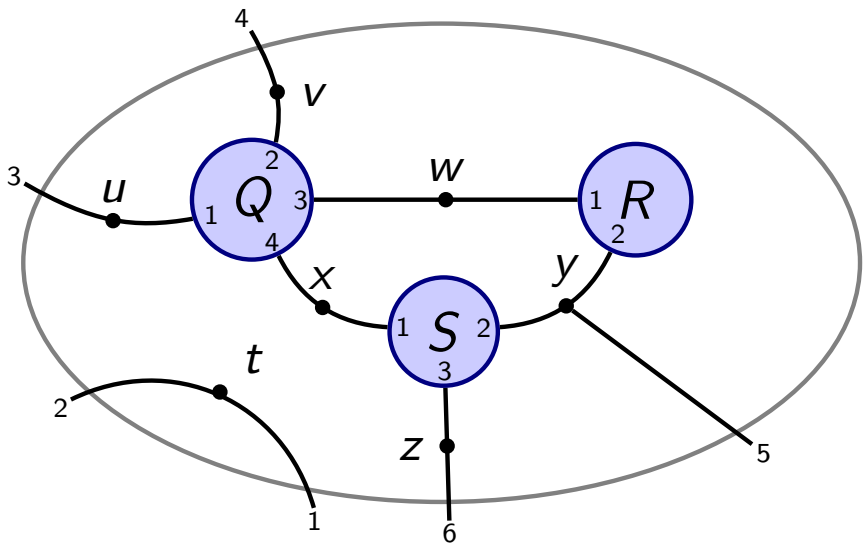
Wiring diagrams as logical expressions

We can convert a wiring diagram like this into a logical expression:



Wiring diagrams as logical expressions

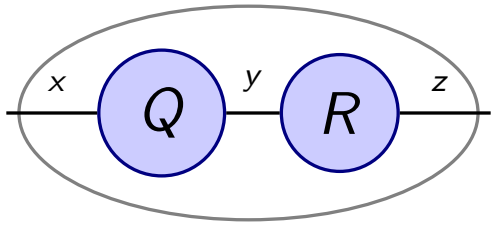
We can convert a wiring diagram like this into a logical expression:



- Write type of exterior shell Γ', naming each port by a distinct variable.
- Write quantifier $\exists(x : X)$ for each unexposed wire of type X.
- AND together internal cells, with established var. names from above.
- Equate variables for exposed ports that are connected.

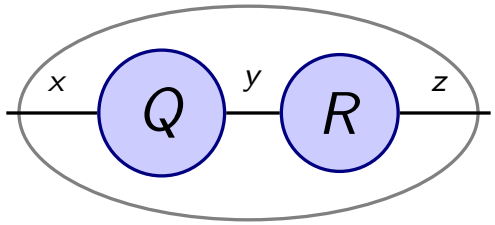
$$O(t_1,t_2,u_3,v_4,y_5,z_6) := \exists(w{:}W, x{:}X).\, Q(u_3,v_4,w,x) \wedge R(w,y_5) \wedge S(x,y_5,z_6) \wedge (t_1 = t_2)$$

Simpler example



$$O(x, z) :=$$

Simpler example

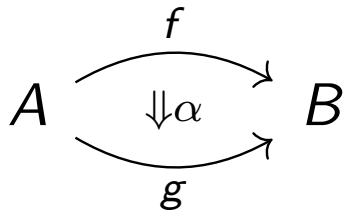


$$O(x,z) \coloneqq \exists y.\, Q(x,y) \wedge R(y,z)$$

Primer on monoidal 2-categories

What's a 2-category?

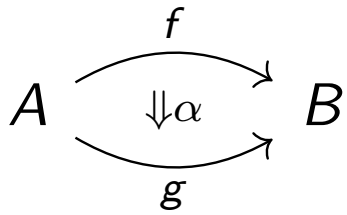
- A (1-) category has objects, 1-morphisms, and composition.
- A 2-category adds 2-morphisms between morphisms.



Primer on monoidal 2-categories

What's a 2-category?

- A (1-) category has objects, 1-morphisms, and composition.
- A 2-category adds 2-morphisms between morphisms.

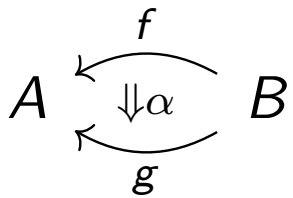


2-categories can be op'd or co'd.

Primer on monoidal 2-categories

What's a 2-category?

- A (1-) category has objects, 1-morphisms, and composition.
- A 2-category adds 2-morphisms between morphisms.



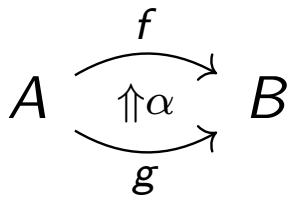
2-categories can be op'd or co'd.

- The 1-opposite category $\mathcal{C}^{\mathrm{op}}$ reverses the 1-arrows.

Primer on monoidal 2-categories

What's a 2-category?

- A (1-) category has objects, 1-morphisms, and composition.
- A 2-category adds 2-morphisms between morphisms.



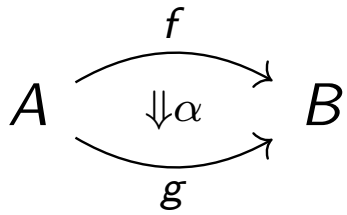
2-categories can be op'd or co'd.

- The 1-opposite category $\mathcal{C}^{\mathrm{op}}$ reverses the 1-arrows.
- The 2-opposite category $\mathcal{C}^{\mathrm{co}}$ reverses the 2-arrows.

Primer on monoidal 2-categories

What's a 2-category?

- A (1-) category has objects, 1-morphisms, and composition.
- A 2-category adds 2-morphisms between morphisms.



2-categories can be op'd or co'd.

- The 1-opposite category $\mathcal{C}^{\mathrm{op}}$ reverses the 1-arrows.
- The 2-opposite category $\mathcal{C}^{\mathrm{co}}$ reverses the 2-arrows.

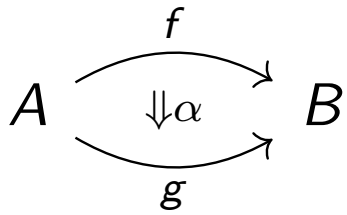
In monoidal 2-categories, you can combine things.

- Given objects A_1, A_2, you can form $A_1 \odot A_2$.
- Think about product types or sum types.

Primer on monoidal 2-categories

What's a 2-category?

- A (1-) category has objects, 1-morphisms, and composition.
- A 2-category adds 2-morphisms between morphisms.



2-categories can be op'd or co'd.

- The 1-opposite category $\mathcal{C}^{\mathrm{op}}$ reverses the 1-arrows.
- The 2-opposite category $\mathcal{C}^{\mathrm{co}}$ reverses the 2-arrows.

In monoidal 2-categories, you can combine things.

- Given objects A_1, A_2, you can form $A_1 \odot A_2$.
- Think about product types or sum types.
- These operate on morphisms too (think $f_1 \odot f_2 \colon A_1 \odot A_2 \to B_1 \odot B_2$).
- There's also a monoidal unit (think unit or void).

Formal specification of graphical calculi I

Our "minority report" detective GUI can be understood as follows.

- Fix a set T (each $t \in \mathsf{T}$ is a string label: person, height, etc.).
- Consider the symmetric monoidal 2-category $\mathbb{C}\mathrm{ospan}_{\mathsf{T}}^{\mathrm{co}}$.

Formal specification of graphical calculi I

Our "minority report" detective GUI can be understood as follows.

- Fix a set T (each $t \in \mathsf{T}$ is a string label: person, height, etc.).
- Consider the symmetric monoidal 2-category $\mathbb{C}\mathrm{ospan}^{\mathrm{co}}_{\mathsf{T}}$.
  - Objects: arities $\underline{n} \xrightarrow{t} \mathsf{T}$, i.e. lists $\Gamma = (t(1), \ldots, t(n)) \in \mathsf{T}^n$.
  - 1-morphisms $(n_1, t_1) \to (n_2, t_2)$: cospans $n_1 \to W \leftarrow n_2$, pres'ing types.
  - 2-morphisms: maps $W' \to W$ making two triangles commute.
  - Monoidal structure: concatenate lists.

Formal specification of graphical calculi I

Our "minority report" detective GUI can be understood as follows.

- Fix a set T (each $t \in \mathsf{T}$ is a string label: person, height, etc.).
- Consider the symmetric monoidal 2-category $\mathbb{C}\mathrm{ospan}^{\mathrm{co}}_{\mathsf{T}}$.
  - Objects: arities $\underline{n} \xrightarrow{t} \mathsf{T}$, i.e. lists $\Gamma = (t(1), \ldots, t(n)) \in \mathsf{T}^n$.
  - 1-morphisms $(n_1, t_1) \to (n_2, t_2)$: cospans $n_1 \to W \leftarrow n_2$, pres'ing types.
  - 2-morphisms: maps $W' \to W$ making two triangles commute.
  - Monoidal structure: concatenate lists.
- Consider the monoidal 2-category $\mathbb{P}\mathrm{oset}$.
  - Obj: posets; 1-morphisms: monotone maps; 2-morphisms: nat. trans.
  - Monoidal structure: $(1, \times)$.

Formal specification of graphical calculi I

Our "minority report" detective GUI can be understood as follows.

- Fix a set T (each $t \in \mathsf{T}$ is a string label: person, height, etc.).
- Consider the symmetric monoidal 2-category $\mathbb{C}\mathrm{ospan}^{\mathrm{co}}_{\mathsf{T}}$.
  - Objects: arities $\underline{n} \xrightarrow{t} \mathsf{T}$, i.e. lists $\Gamma = (t(1), \ldots, t(n)) \in \mathsf{T}^n$.
  - 1-morphisms $(n_1, t_1) \to (n_2, t_2)$: cospans $n_1 \to W \leftarrow n_2$, pres'ing types.
  - 2-morphisms: maps $W' \to W$ making two triangles commute.
  - Monoidal structure: concatenate lists.
- Consider the monoidal 2-category $\mathbb{P}\mathrm{oset}$.
  - Obj: posets; 1-morphisms: monotone maps; 2-morphisms: nat. trans.
  - Monoidal structure: $(1, \times)$.
- Detective will be a certain kind of 2-functor $P\colon \mathbb{C}\mathrm{ospan}^{\mathrm{co}}_{\mathsf{T}} \to \mathbb{P}\mathrm{oset}$.
  - To each shell $\Gamma \in \mathbb{C}\mathrm{ospan}^{\mathrm{co}}_{\mathsf{T}}$, a poset $P(\Gamma)$.
  - We denote the order in $P(\Gamma)$ using the logical *entailment* symbol $\vdash$.

Formal specification of graphical calculi II

We have monoidal 2-categories $\mathbb{C}\text{ospan}^{\text{co}}$ and $\mathbb{P}\text{oset}$.

Definition

An *regular calculus* is a lax monoidal 2-functor

$$P\colon \mathbb{C}\text{ospan}_{\mathsf{T}}^{\text{co}} \to \mathbb{P}\text{oset}$$

such that the laxators are right adjoints.

Formal specification of graphical calculi II

We have monoidal 2-categories $\mathbb{C}\mathrm{ospan}^{\mathrm{co}}$ and $\mathbb{P}\mathrm{oset}$.

Definition

An *regular calculus* is a lax monoidal 2-functor

$$P\colon \mathbb{C}\mathrm{ospan}_{\mathsf{T}}^{\mathrm{co}} \to \mathbb{P}\mathrm{oset}$$

such that the laxators are right adjoints.

Our terminology: *ajax* monoidal functors: the laxators

$$1 \xrightarrow{\rho_1} P(0) \qquad \text{and} \qquad P(v) \times P(v') \xrightarrow{\rho_{v,v'}} P(v+v').$$

Formal specification of graphical calculi II

We have monoidal 2-categories $\mathbb{C}\mathrm{ospan}^{\mathrm{co}}$ and $\mathbb{P}\mathrm{oset}$.

Definition

An *regular calculus* is a lax monoidal 2-functor

$$P\colon \mathbb{C}\mathrm{ospan}_{\mathsf{T}}^{\mathrm{co}} \to \mathbb{P}\mathrm{oset}$$

such that the laxators are right adjoints.

Our terminology: *ajax* monoidal functors: the laxators are adjoints

$$1 \underset{\lambda_1}{\overset{\rho_1}{\rightleftarrows}} P(0) \qquad \text{and} \qquad P(v)\times P(v') \underset{\lambda_{v,v'}}{\overset{\rho_{v,v'}}{\rightleftarrows}} P(v+v').$$

Regular calculi and regular categories

Denote by RegCalc the category of regular calculi

$$\mathsf{RegCalc} := \big((\mathsf{T}, P \colon \mathbb{C}\mathrm{ospan}_{\mathsf{T}}^{\mathrm{co}} \to \mathbb{P}\mathrm{oset})\big).$$

Regular calculi and regular categories

Denote by RegCalc the category of regular calculi

$$\mathsf{RegCalc} := \big((\mathsf{T}, P \colon \mathbb{C}\mathrm{ospan}_{\mathsf{T}}^{\mathrm{co}} \to \mathbb{P}\mathrm{oset})\big).$$

Theorem

There is an adjunction

$$\mathsf{RegCalc} \underset{\mathsf{prd}}{\overset{\mathsf{syn}}{\rightleftarrows}} \mathsf{RegCat} \ ,$$

such that for any regular category $\mathcal{R}$, the counit $\mathsf{syn} \circ \mathsf{prd}(\mathcal{R}) \to \mathcal{R}$ *is an equivalence of categories.*

Mathematical spec of the GUI

We want software to do the detective work; call it "Reglog – the game."

Mathematical spec of the GUI

We want software to do the detective work; call it "Reglog – the game."

- Each 'game': a set T and an ajax 2-functor $P : \mathbb{C}\mathrm{ospan}_{\mathsf{T}}^{\mathsf{co}} \to \mathbb{P}\mathrm{oset}$.
- Repeating a bit... what is $\mathbb{C}\mathrm{ospan}_{\mathsf{T}}^{\mathsf{co}}$?

Mathematical spec of the GUI

We want software to do the detective work; call it "Reglog – the game."

- Each 'game': a set T and an ajax 2-functor $P\colon \mathbb{C}\mathrm{ospan}_{\mathsf{T}}^{\mathrm{co}} \to \mathbb{P}\mathrm{oset}$.
- Repeating a bit... what is $\mathbb{C}\mathrm{ospan}_{\mathsf{T}}^{\mathrm{co}}$?
  - Objects: drawn as shells  , encoded as lists $(t_1, \ldots, t_n)$.
  - Monoidal product: drawn as multiple shells, encoded as list concat.
  - Morphisms: drawn as wiring diagrams, encoded as cospans.
  - 2-structure: drawn as breaking dots, encoded as cospan maps.

Mathematical spec of the GUI

We want software to do the detective work; call it "Reglog – the game."

- Each 'game': a set T and an ajax 2-functor $P\colon \mathbb{C}\mathrm{ospan}_{\mathsf{T}}^{\mathrm{co}} \to \mathbb{P}\mathrm{oset}$.
- Repeating a bit... what is $\mathbb{C}\mathrm{ospan}_{\mathsf{T}}^{\mathrm{co}}$?
  - Objects: drawn as shells  , encoded as lists $(t_1, \ldots, t_n)$.
  - Monoidal product: drawn as multiple shells, encoded as list concat.
  - Morphisms: drawn as wiring diagrams, encoded as cospans.
  - 2-structure: drawn as breaking dots, encoded as cospan maps.
- Each game developer must supply their own P.
  - To each shell Γ, specify elements of $P(\Gamma)$, drawn as cells (fillers).

Mathematical spec of the GUI

We want software to do the detective work; call it "Reglog – the game."

- Each 'game': a set T and an ajax 2-functor $P\colon \mathbb{C}\mathrm{ospan}_{\mathsf{T}}^{\mathsf{co}} \to \mathbb{P}\mathrm{oset}$.
- Repeating a bit... what is $\mathbb{C}\mathrm{ospan}_{\mathsf{T}}^{\mathsf{co}}$?
  - Objects: drawn as shells  , encoded as lists $(t_1, \ldots, t_n)$.
  - Monoidal product: drawn as multiple shells, encoded as list concat.
  - Morphisms: drawn as wiring diagrams, encoded as cospans.
  - 2-structure: drawn as breaking dots, encoded as cospan maps.
- Each game developer must supply their own P.
  - To each shell Γ, specify elements of $P(\Gamma)$, drawn as cells (fillers).
  - Specify definition of $\varphi_1 \leq \varphi_2$, drawn perhaps as $\varphi_1 \vdash \varphi_2$.
  - To each wiring diagram w, supply monotonic function $P(w)\colon P(\Gamma_1) \times \cdots \times P(\Gamma_n) \to P(\Gamma')$.
  - Ensure P preserves composition, identity, and dot-breaking.

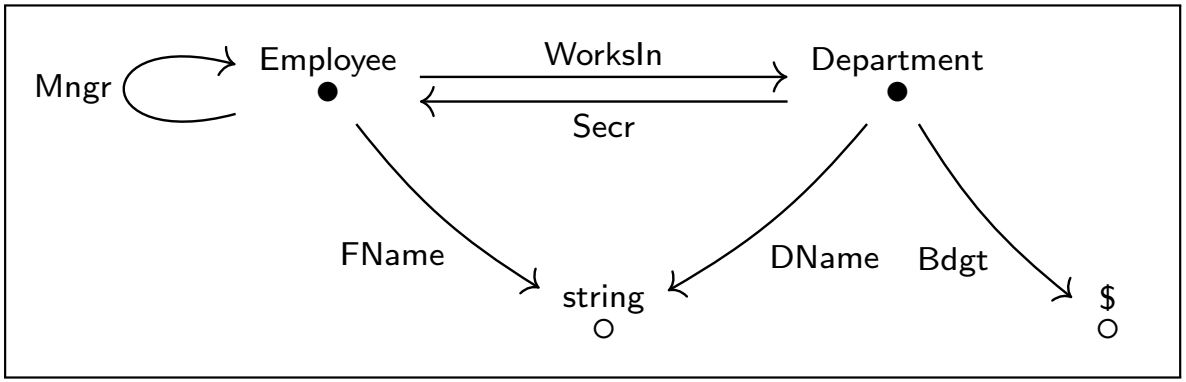
Backend: open source CQL

Backend: categorical query language, another approach to databases.

Backend: open source CQL

Backend: categorical query language, another approach to databases.

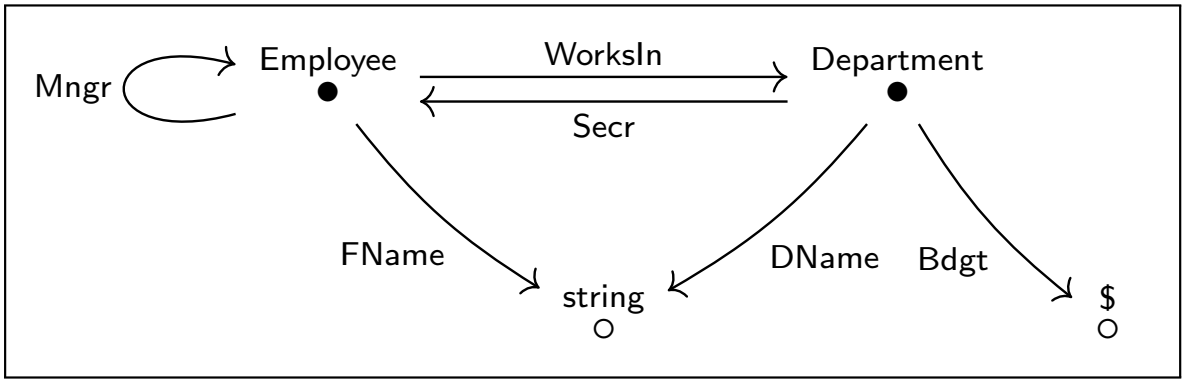
- A schema S is roughly a category and looks like this:



Backend: open source CQL

Backend: categorical query language, another approach to databases.

- A schema S is roughly a category and looks like this:

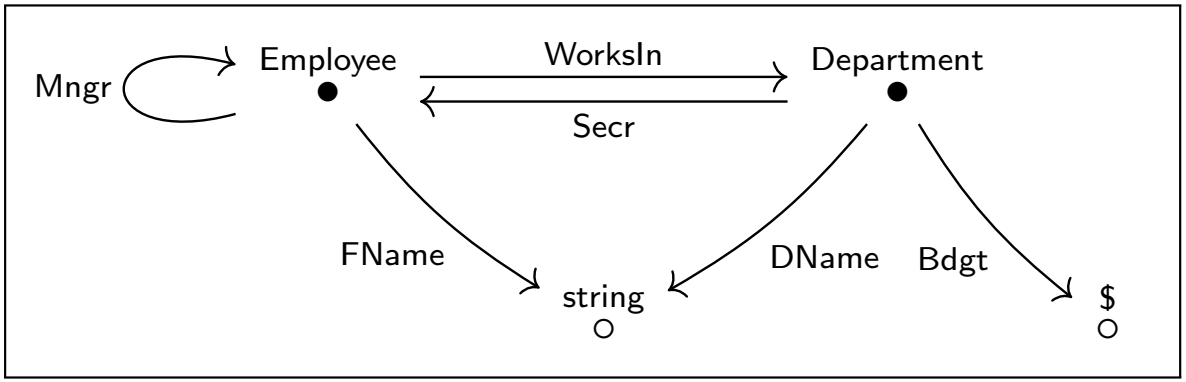


- An instance is roughly a functor $I : S \to \mathrm{Set}$.

Backend: open source CQL

Backend: categorical query language, another approach to databases.

- A schema S is roughly a category and looks like this:

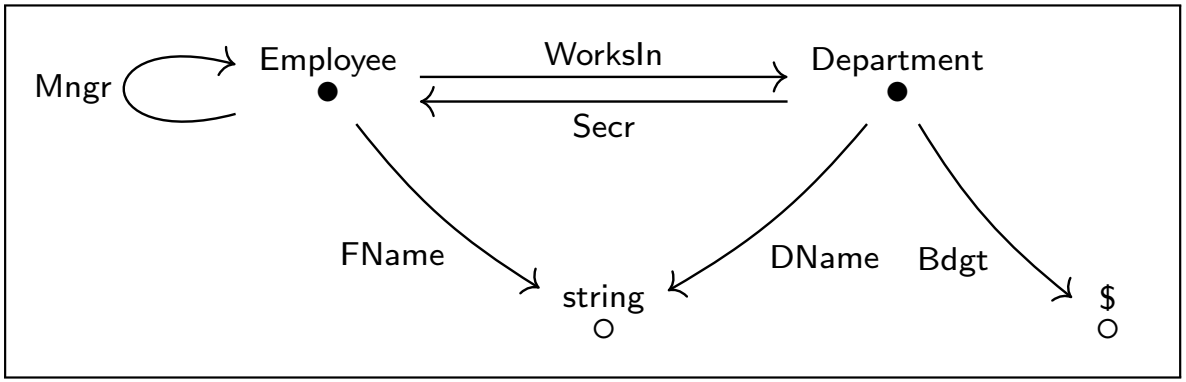


- An instance is roughly a functor $I : S \to \mathrm{Set}$.
- Given schema S, consider each dot $s \in S$ as a shell and as a type.
  - Ports of shell s: its outgoing arrows $s \to t$ typed by target t.
  - If I is an S-instance, elements of $I(s)$ fill shell s, as in detective game.

Backend: open source CQL

Backend: categorical query language, another approach to databases.

- A schema S is roughly a category and looks like this:



- An instance is roughly a functor $I : S \to \mathrm{Set}$.
- Given schema S, consider each dot $s \in S$ as a shell and as a type.
  - Ports of shell s: its outgoing arrows $s \to t$ typed by target t.
  - If I is an S-instance, elements of $I(s)$ fill shell s, as in detective game.
- Each regular sequent $\phi(\vec{x}) \vdash \psi(\vec{x})$ is called an embedded dependency.
- Chase these EDs to force axioms to hold with minimal other assumptions.

Outline

Common game platform

"Reglog – the game" is actually a bunch of games.

- In common: same GUI and same sort of interaction:
  - Create new shells, attach shells, compose wiring diagrams
  - Zoom in and out of wiring diagrams.
  - Fill shells with content of a certain form, to make "cells".
  - Click on cells to interact with content.
  - Do reasoning (entailment, substitution, dot-breaking).

Common game platform

"Reglog – the game" is actually a bunch of games.

- In common: same GUI and same sort of interaction:
  - Create new shells, attach shells, compose wiring diagrams
  - Zoom in and out of wiring diagrams.
  - Fill shells with content of a certain form, to make "cells".
  - Click on cells to interact with content.
  - Do reasoning (entailment, substitution, dot-breaking).
- Differences between different games P:
  - Formally, $P : \mathbb{C}\mathrm{ospan}_{\mathsf{T}}^{\mathrm{co}} \longrightarrow \mathbb{P}\mathrm{oset}$ can be any "ajax functor".
  - Different P's specify different sorts of content for shells $\Gamma \in \mathbb{C}\mathrm{ospan}_{\mathsf{T}}^{\mathrm{co}}$.

Common game platform

"Reglog – the game" is actually a bunch of games.

- In common: same GUI and same sort of interaction:
  - Create new shells, attach shells, compose wiring diagrams
  - Zoom in and out of wiring diagrams.
  - Fill shells with content of a certain form, to make "cells".
  - Click on cells to interact with content.
  - Do reasoning (entailment, substitution, dot-breaking).
- Differences between different games P:
  - Formally, $P : \mathbb{C}\mathrm{ospan}_{\mathsf{T}}^{\mathrm{co}} \longrightarrow \mathbb{P}\mathrm{oset}$ can be any "ajax functor".
  - Different P's specify different sorts of content for shells $\Gamma \in \mathbb{C}\mathrm{ospan}_{\mathsf{T}}^{\mathrm{co}}$.
- Some examples:
  - Where are my keys?
  - Smart witter
  - Partitions puzzle
  - Boolean circuits
  - Solve equations
  - etc.

Game: Where are my keys?

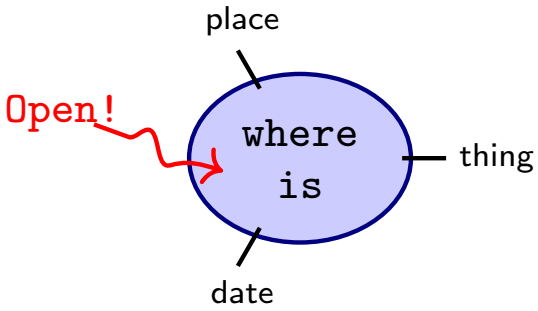
A simpler version of detective game: no reasoning $\vdash$, just query.

- Tell Alexa or Siri facts as you get them, and sort your life.

Game: Where are my keys?

A simpler version of detective game: no reasoning $\vdash$, just query.

- Tell Alexa or Siri facts as you get them, and sort your life.



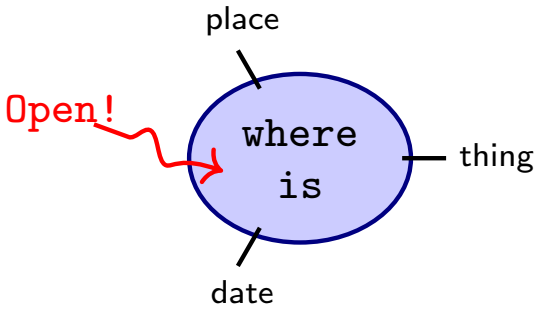
. . . .

where is		
thing	place	date
keys	top drawer	2019/06/23
lease	file cabinet	2019/01/01
gift for Mom	basement	2019/03/30
. . .	. . .	. . .

Game: Where are my keys?

A simpler version of detective game: no reasoning $\vdash$, just query.

- Tell Alexa or Siri facts as you get them, and sort your life.



. . . .

where is		
thing	place	date
keys	top drawer	2019/06/23
lease	file cabinet	2019/01/01
gift for Mom	basement	2019/03/30
. . .	. . .	. . .

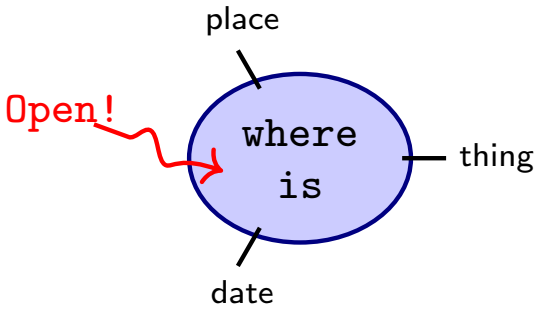
- Query by attaching and clicking
  - Attach 

to find places your keys were yesterday.
  - Similar shell for your calendar, your address book, etc.

Game: Where are my keys?

A simpler version of detective game: no reasoning $\vdash$, just query.

- Tell Alexa or Siri facts as you get them, and sort your life.



where is		
thing	place	date
keys	top drawer	2019/06/23
lease	file cabinet	2019/01/01
gift for Mom	basement	2019/03/30
...	...	...

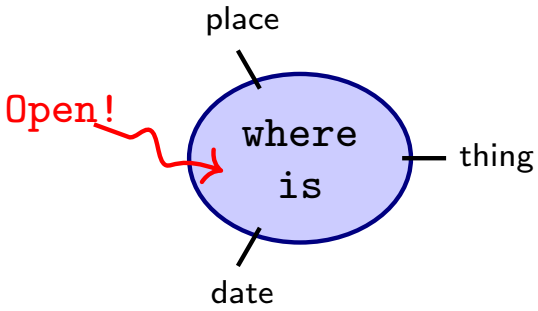
- Query by attaching and clicking
  - Attach 

to find places your keys were yesterday.
  - Similar shell for your calendar, your address book, etc.
- Use public information repos.

Game: Where are my keys?

A simpler version of detective game: no reasoning $\vdash$, just query.

- Tell Alexa or Siri facts as you get them, and sort your life.



where is		
thing	place	date
keys	top drawer	2019/06/23
lease	file cabinet	2019/01/01
gift for Mom	basement	2019/03/30
...	...	...

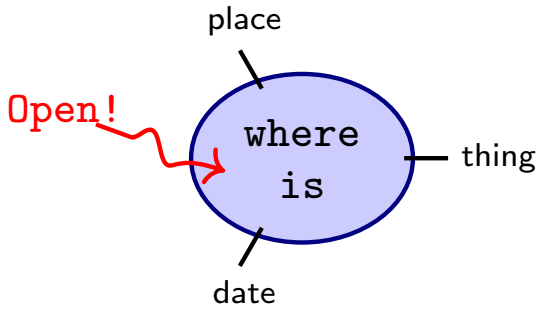
- Query by attaching and clicking
  - Attach 

to find places your keys were yesterday.
  - Similar shell for your calendar, your address book, etc.
- Use public information repos.
  - "Cardiologists who fit my availability and who take my insurance."

Game: Where are my keys?

A simpler version of detective game: no reasoning $\vdash$, just query.

- Tell Alexa or Siri facts as you get them, and sort your life.



. . . .

where is		
thing	place	date
keys	top drawer	2019/06/23
lease	file cabinet	2019/01/01
gift for Mom	basement	2019/03/30
. . .	. . .	. . .

- Query by attaching and clicking
  - Attach thing — keys yesterday — date to find places your keys were yesterday.
  - Similar shell for your calendar, your address book, etc.
- Use public information repos.
  - "Cardiologists who fit my availability and who take my insurance."
  - There is an internet-published *doctor* cell doctor — specialty .
  - Connect specialty — cardiologist to doctor's speciality port.
  - Connect doctor's availability and insurance ports to your own.
  - Output doctor's name and phone number.

Game: Smart witter

A smarter social network.

- This is another simplification of the detective game.
- Difference: cells are inhabited only by wirings of other cells.
- Idea: people blog concepts built from simple pieces.

Game: Smart witter

A smarter social network.

- This is another simplification of the detective game.
- Difference: cells are inhabited only by wirings of other cells.
- Idea: people blog concepts built from simple pieces.
  - Others can disassemble, reassemble, and nick-name concepts
  - Zoom into others' concepts to see what they really mean.

Game: Smart witter

A smarter social network.

- This is another simplification of the detective game.
- Difference: cells are inhabited only by wirings of other cells.
- Idea: people blog concepts built from simple pieces.
  - Others can disassemble, reassemble, and nick-name concepts
  - Zoom into others' concepts to see what they really mean.
- What's trending? Concepts.
  - Maybe the machine can find concepts that people keep reusing.

Game: Smart witter

A smarter social network.

- This is another simplification of the detective game.
- Difference: cells are inhabited only by wirings of other cells.
- Idea: people blog concepts built from simple pieces.
  - Others can disassemble, reassemble, and nick-name concepts
  - Zoom into others' concepts to see what they really mean.
- What's trending? Concepts.
  - Maybe the machine can find concepts that people keep reusing.
    - Logic gates are particular wirings of transistors.
    - Adder circuits are particular wirings of logic gates.
    - Out of all configurations, some are very popular, i.e. reused.

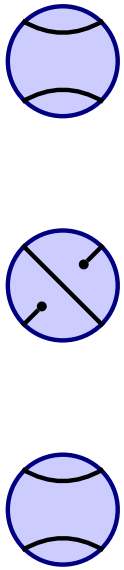
Game: Smart witter

A smarter social network.

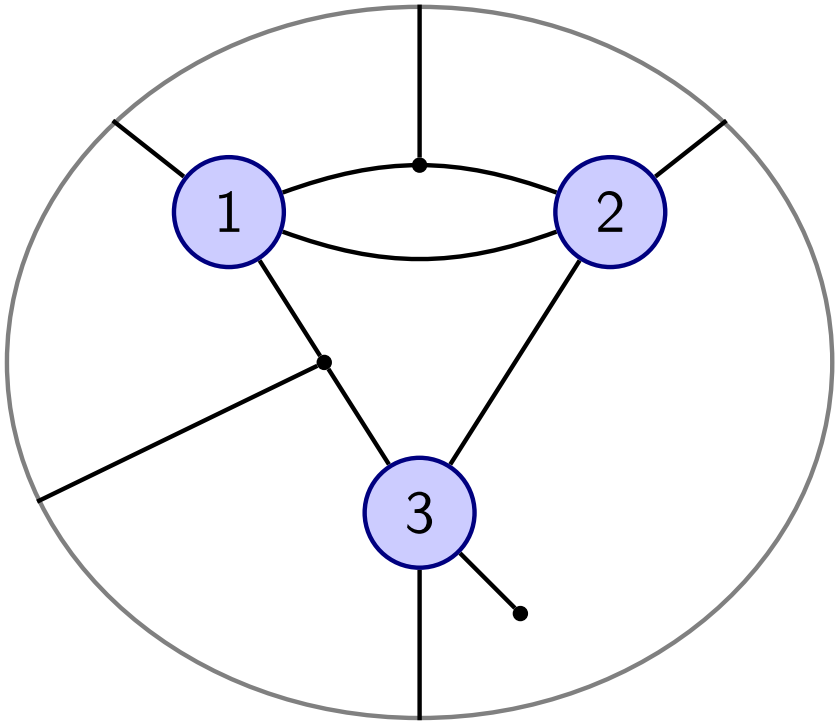
- This is another simplification of the detective game.
- Difference: cells are inhabited only by wirings of other cells.
- Idea: people blog concepts built from simple pieces.
  - Others can disassemble, reassemble, and nick-name concepts
  - Zoom into others' concepts to see what they really mean.
- What's trending? Concepts.
  - Maybe the machine can find concepts that people keep reusing.
    - Logic gates are particular wirings of transistors.
    - Adder circuits are particular wirings of logic gates.
    - Out of all configurations, some are very popular, i.e. reused.
  - Same idea for human concepts; find reusable ideas (memes).

Game: Partitions puzzle

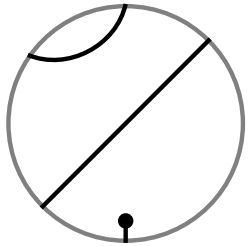
Put pieces:



into puzzle:

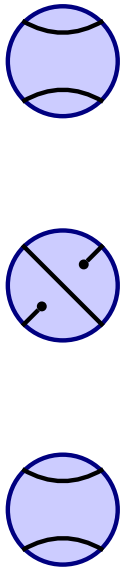


to obtain:

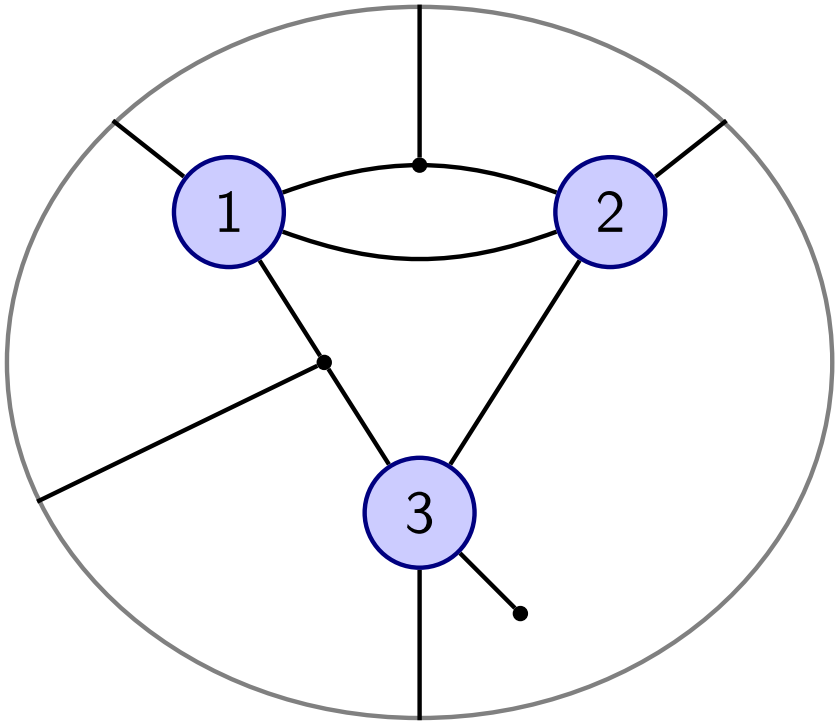


Game: Partitions puzzle

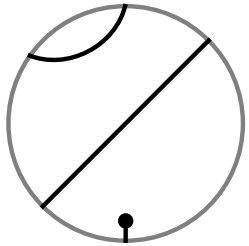
Put pieces:



into puzzle:

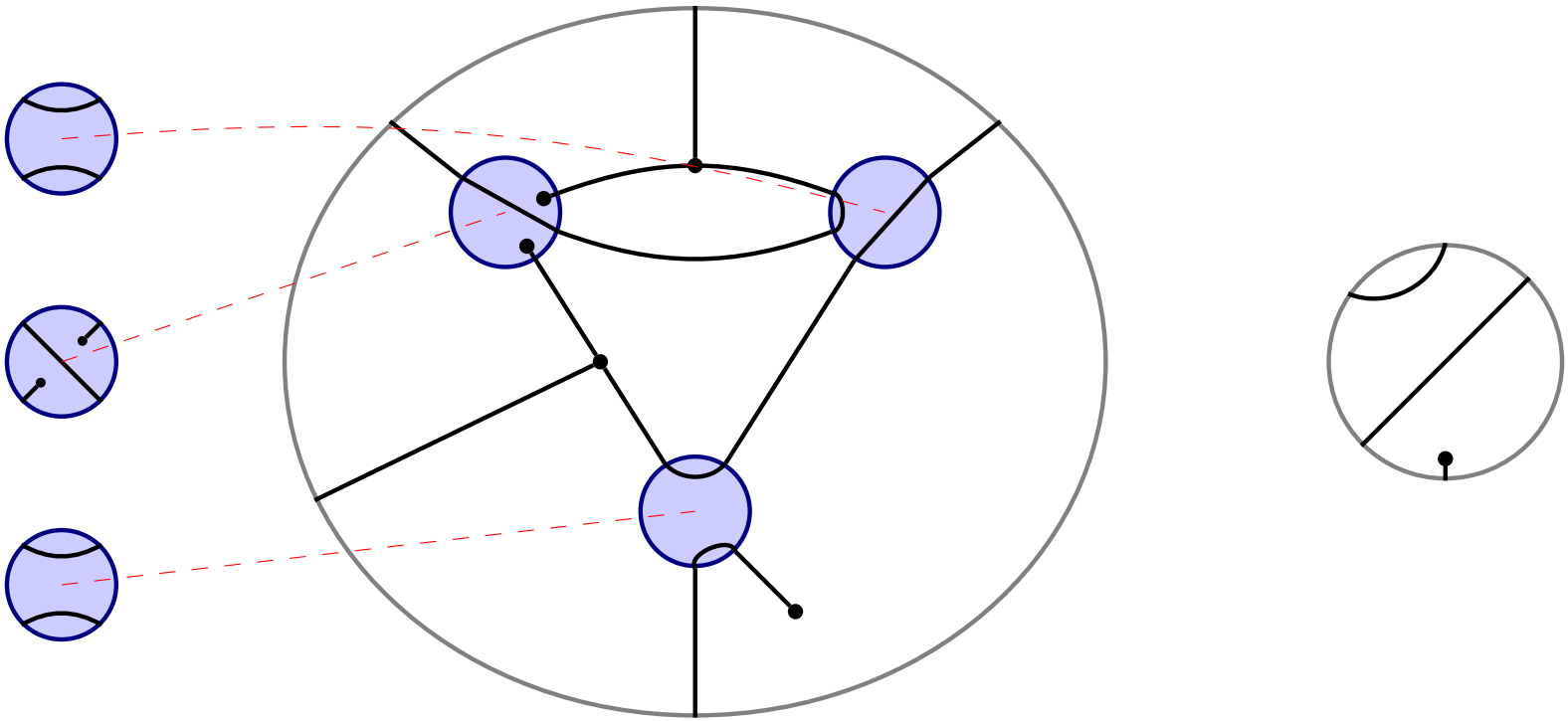


to obtain:

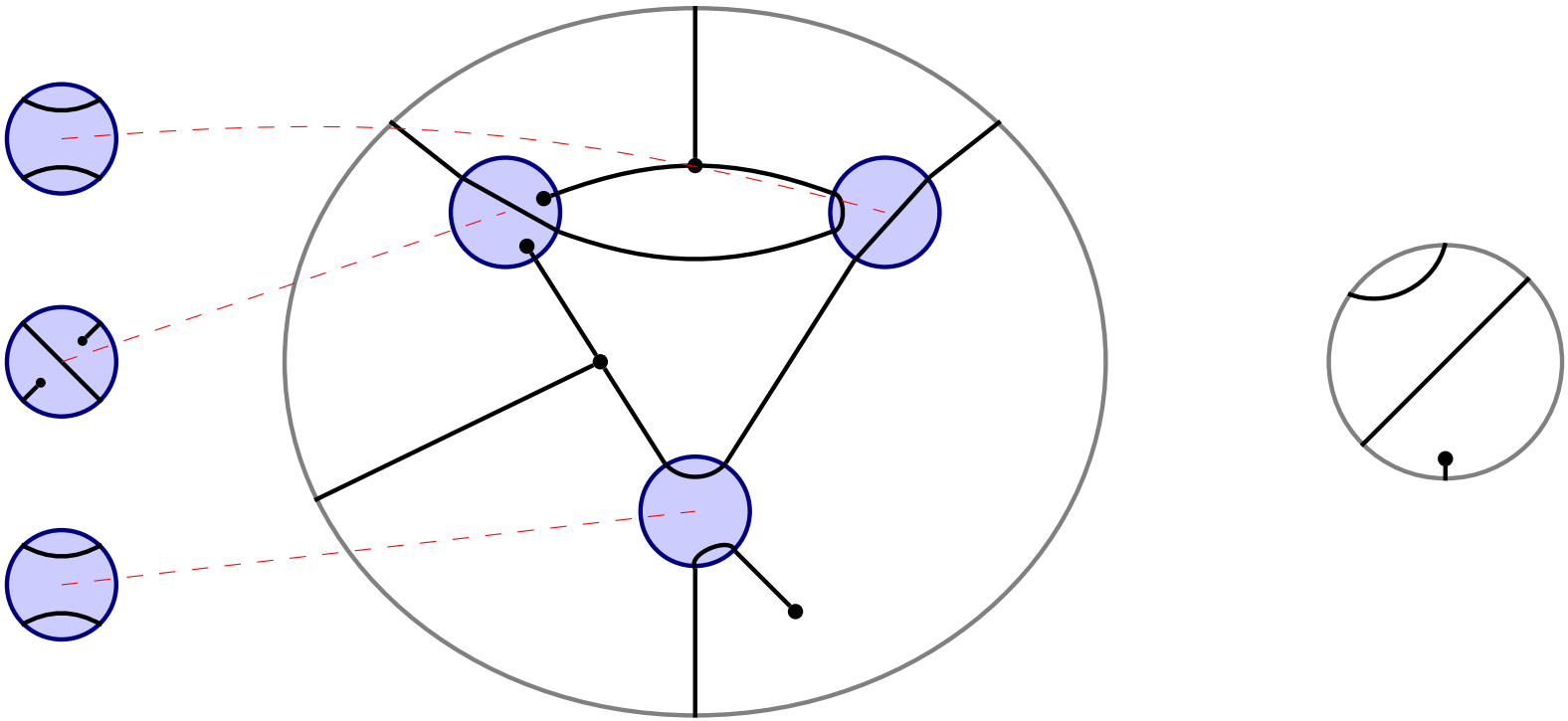


I'll give you a minute to solve it.

Game: partitions puzzle (solution)



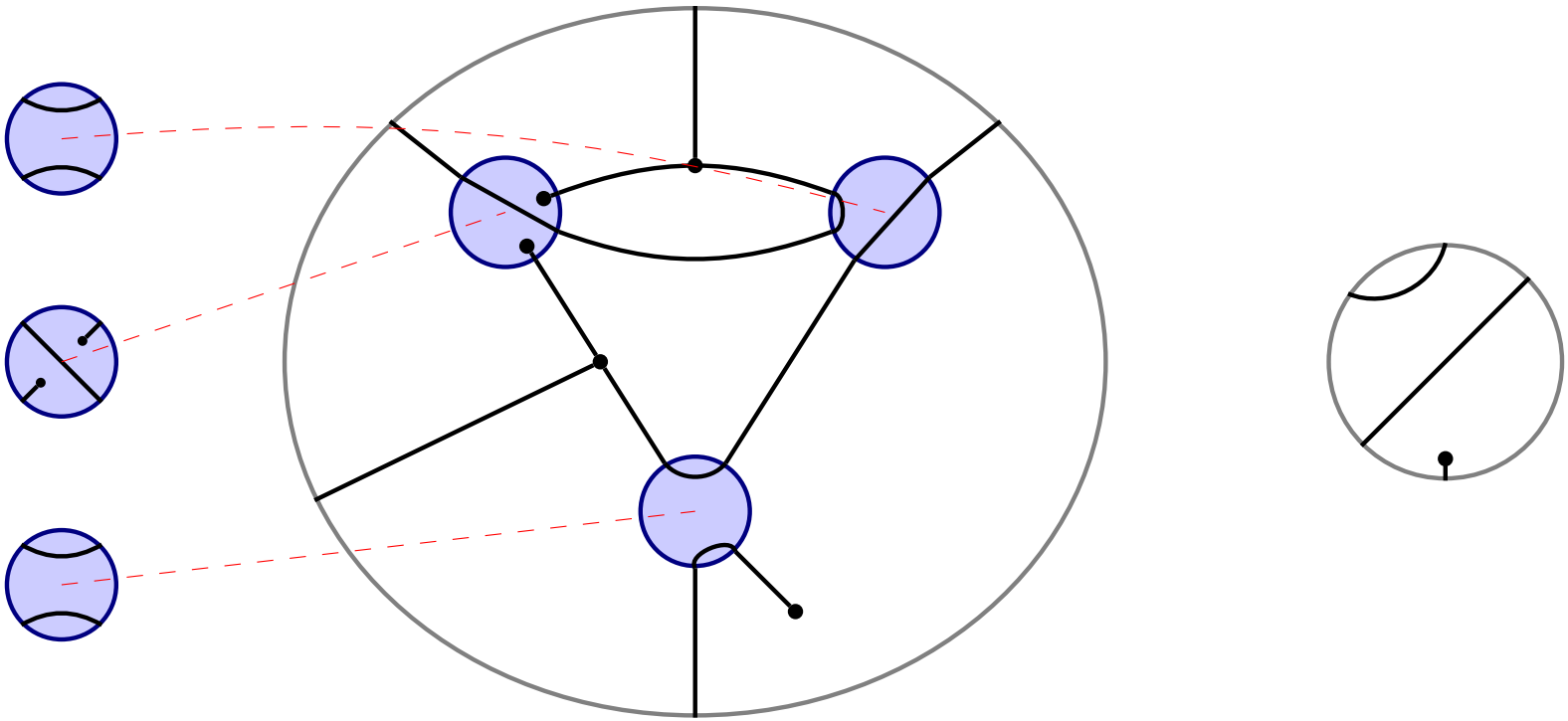
Game: partitions puzzle (solution)



Pretty easy to describe as an ajax functor $P\colon \mathbb{C}\mathrm{ospan}^{\mathrm{co}} \to \mathbb{P}\mathrm{oset}$.

- Since wires are unlabeled, we see that $\mathsf{T} = ()$ is unit.
- And for each $n \in \mathbb{C}\mathrm{ospan}^{\mathrm{co}}$, put $P(n) := \{\text{partitions of } n\}$.

Game: partitions puzzle (solution)



Pretty easy to describe as an ajax functor $P\colon \mathbb{C}\mathrm{ospan}^{\mathrm{co}} \to \mathbb{P}\mathrm{oset}$.

- Since wires are unlabeled, we see that $\mathsf{T} = ()$ is unit.
- And for each $n \in \mathbb{C}\mathrm{ospan}^{\mathrm{co}}$, put $P(n) := \{\text{partitions of } n\}$.
  - $P(\_)$ is just the "poset reflection" of $\mathbb{C}\mathrm{ospan}^{\mathrm{co}}(0, \_)$.

Game: Boolean circuits

Boolean circuits are special cases of boolean relations.

Game: Boolean circuits

Boolean circuits are special cases of boolean relations.

- What are boolean relations?
  - A boolean relation is a subset of $\mathbb{B}^n = \{\texttt{true}, \texttt{false}\}^n$ for some n.
  - Familiar: AND, OR, IMPLIES, NOT, TRUE, FALSE.

Game: Boolean circuits

Boolean circuits are special cases of boolean relations.

- What are boolean relations?
  - A boolean relation is a subset of $\mathbb{B}^n = \{\texttt{true}, \texttt{false}\}^n$ for some n.
  - Familiar: AND, OR, IMPLIES, NOT, TRUE, FALSE.
  - Those are basic circuits; they're functions.
  - Consider also relations, like $\leq$.

AND		
true	true	true
true	false	false
false	true	false
false	false	false

IMPLIES		
true	true	true
true	false	false
false	true	true
false	false	true

$\leq$	
true	true
false	true
false	false

Game: Boolean circuits

Boolean circuits are special cases of boolean relations.

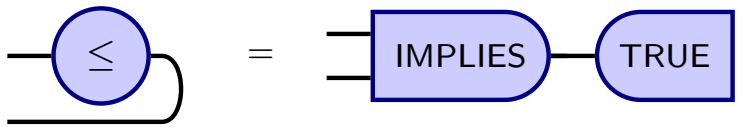
- What are boolean relations?
  - A boolean relation is a subset of $\mathbb{B}^n = \{\texttt{true}, \texttt{false}\}^n$ for some n.
  - Familiar: AND, OR, IMPLIES, NOT, TRUE, FALSE.
  - Those are basic circuits; they're functions.
  - Consider also relations, like $\leq$.

AND		
true	true	true
true	false	false
false	true	false
false	false	false

IMPLIES		
true	true	true
true	false	false
false	true	true
false	false	true

$\leq$	
true	true
false	true
false	false

- Puzzles: build up complex relation S using simple parts $R_1, \ldots, R_n$.
  - Example:



Game: Boolean circuits

Boolean circuits are special cases of boolean relations.

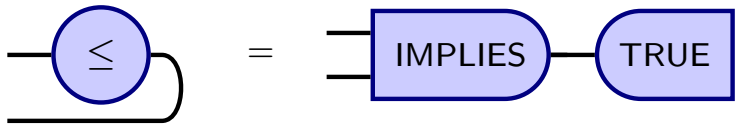
- What are boolean relations?
  - A boolean relation is a subset of $\mathbb{B}^n = \{\texttt{true}, \texttt{false}\}^n$ for some n.
  - Familiar: AND, OR, IMPLIES, NOT, TRUE, FALSE.
  - Those are basic circuits; they're functions.
  - Consider also relations, like $\leq$.

AND		
true	true	true
true	false	false
false	true	false
false	false	false

IMPLIES		
true	true	true
true	false	false
false	true	true
false	false	true

$\leq$	
true	true
false	true
false	false

- Puzzles: build up complex relation S using simple parts $R_1, \ldots, R_n$.
  - Example:



  - Example: every boolean circuit can be built out of NAND gates.

Q: Are there kids who might have fun playing with something like this?

Game: Solving equations

Consider an arbitrary system of equations having the following form:

$$f_1(\mathbf{t}, u, \mathbf{v}) = 0$$
$$f_2(\mathbf{v}, w, x) = 0$$
$$f_3(u, w, x, y) = 0$$
$$f_4(x, \mathbf{z}) = 0$$

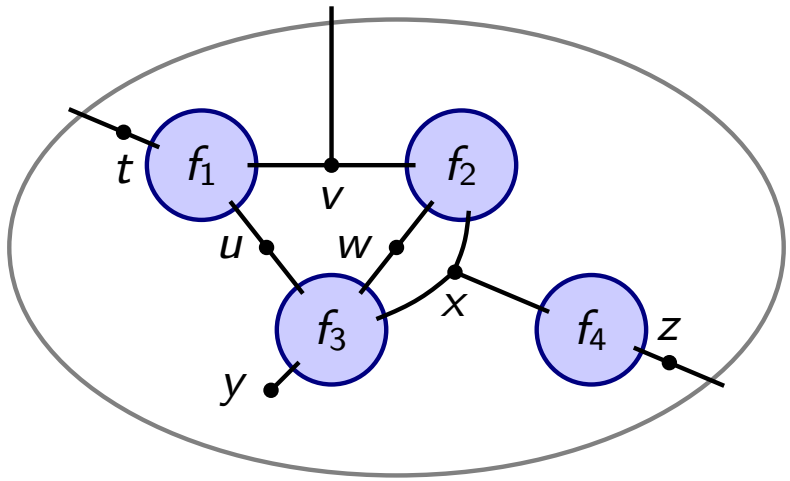
Bold variables are those we want to *expose*; others are latent or *unexposed*.

Game: Solving equations

Consider an arbitrary system of equations having the following form:

$$f_1(\mathbf{t}, u, \mathbf{v}) = 0$$
$$f_2(\mathbf{v}, w, x) = 0$$
$$f_3(u, w, x, y) = 0$$
$$f_4(x, \mathbf{z}) = 0$$

Bold variables are those we want to *expose*; others are latent or *unexposed*.



Said another way, we want $\{(t, v, z) \mid \exists u, w, x, y : f_1 = f_2 = f_3 = f_4 = 0\}$.

Systems of equations via pixel arrays

Consider just two equations $f(x, w) = 0$ and $g(w, y) = 0$.

- Plot each in its own bounding box, say in the range $[-1.5, 1.5]$.
- Consider the plots as matrices M, N whose entries are on/off pixels.
- That is, M and N are *boolean matrices* corresponding to f and g.

Multiplying these two matrices MN yields the simultaneous solution.

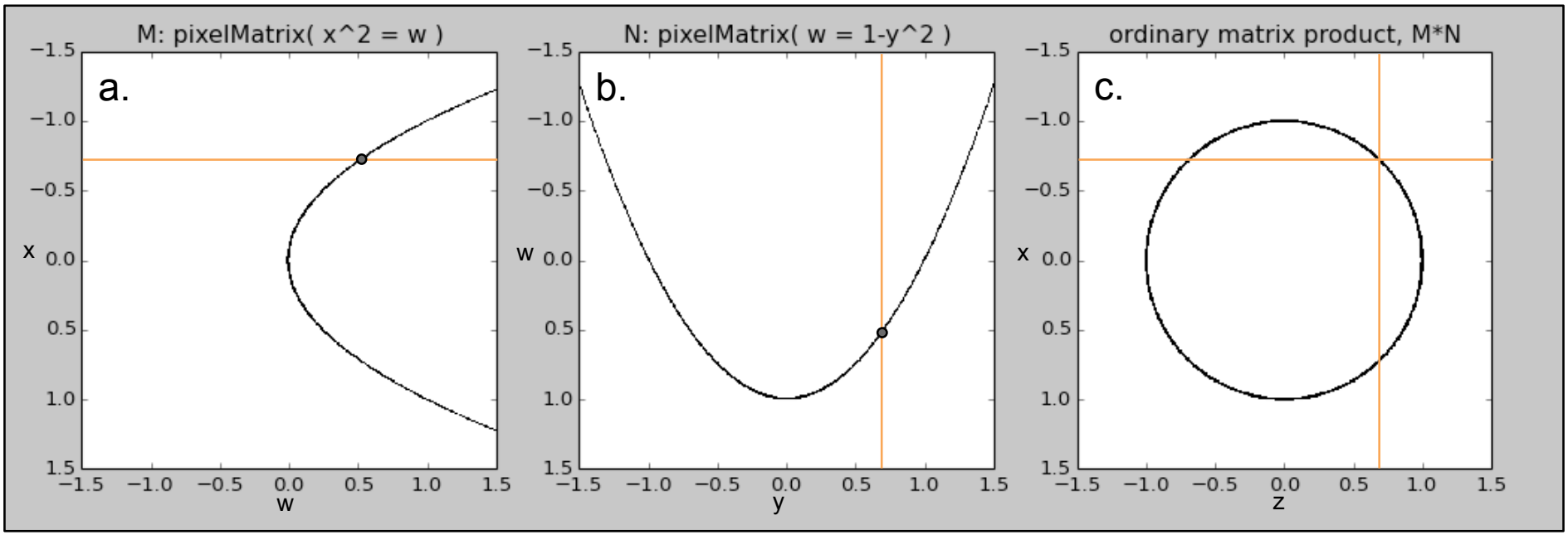
Systems of equations via pixel arrays

Consider just two equations $f(x, w) = 0$ and $g(w, y) = 0$.

- Plot each in its own bounding box, say in the range $[-1.5, 1.5]$.
- Consider the plots as matrices M, N whose entries are on/off pixels.
- That is, M and N are *boolean matrices* corresponding to f and g.

Multiplying these two matrices MN yields the simultaneous solution.

- For example, plot equations $x^2 = w$ and $w = 1 - y^2$, and multiply.



A more complex example

The following eq's are not differentiable, nor even defined everywhere.

$$\cos\left(\ln(z^2 + 10^{-3}x)\right) - x + 10^{-5}z^{-1} = 0 \qquad \text{(Equation 1)}$$

$$\cosh(w + 10^{-3}y) + y + 10^{-4}w = 2 \qquad \text{(Equation 2)}$$

$$\tan(x + y)(x - 2)^{-1}(x + 3)^{-1}y^{-2} = 1 \qquad \text{(Equation 3)}$$

Q: For what values of w and z does a simultaneous solution exist?

A more complex example

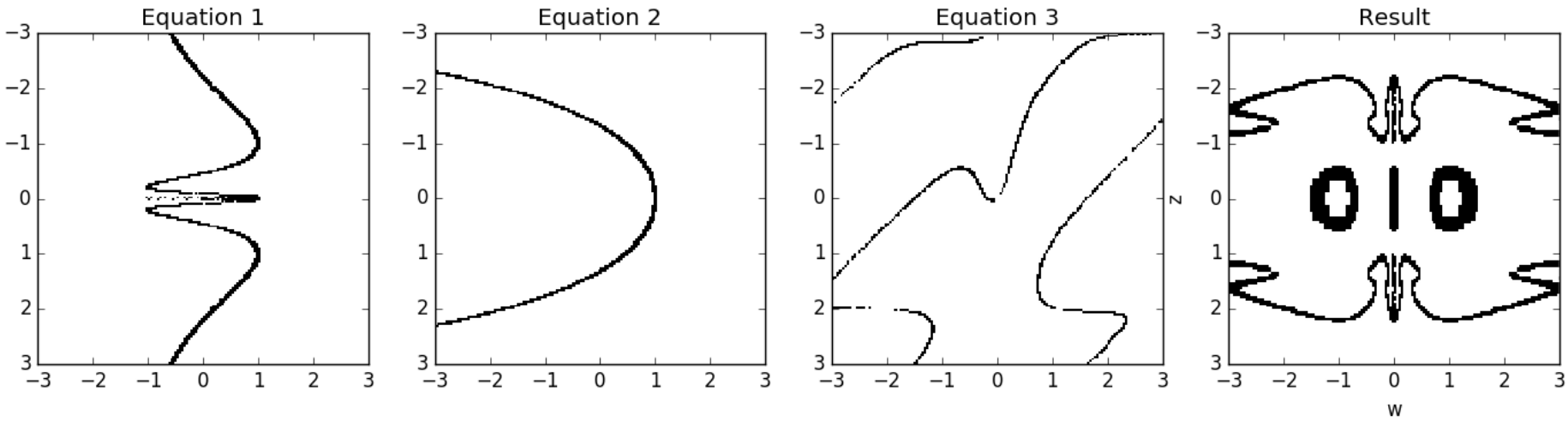
The following eq's are not differentiable, nor even defined everywhere.

$$\cos\left(\ln(z^2 + 10^{-3}x)\right) - x + 10^{-5}z^{-1} = 0 \qquad \text{(Equation 1)}$$

$$\cosh(w + 10^{-3}y) + y + 10^{-4}w = 2 \qquad \text{(Equation 2)}$$

$$\tan(x + y)(x - 2)^{-1}(x + 3)^{-1}y^{-2} = 1 \qquad \text{(Equation 3)}$$

Q: For what values of w and z does a simultaneous solution exist?



A more complex example

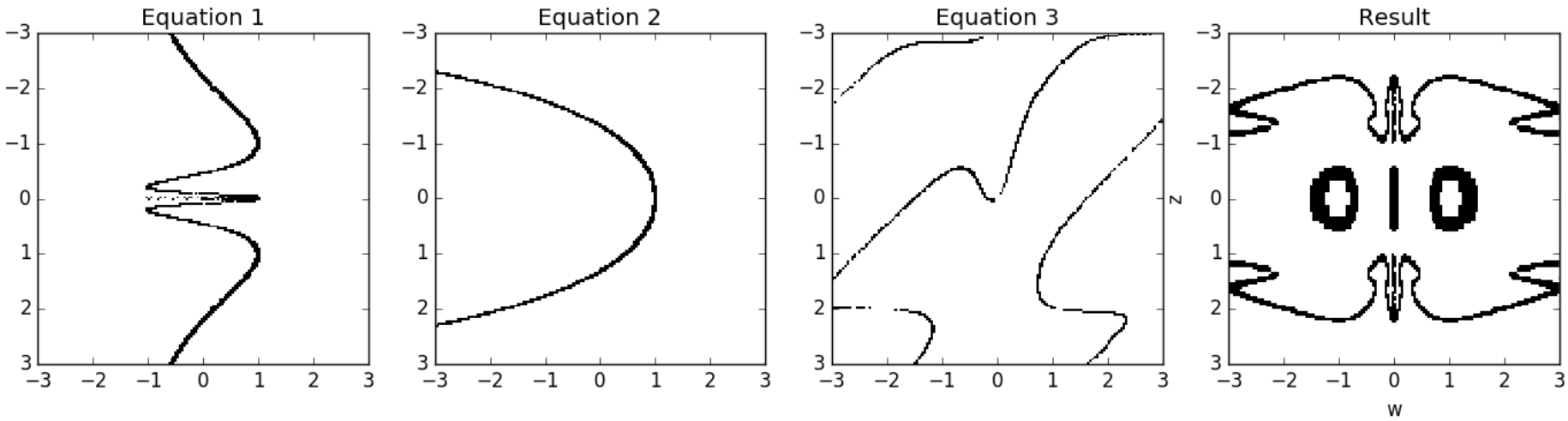
The following eq's are not differentiable, nor even defined everywhere.

$$\cos\left(\ln(z^2 + 10^{-3}x)\right) - x + 10^{-5}z^{-1} = 0 \qquad \text{(Equation 1)}$$

$$\cosh(w + 10^{-3}y) + y + 10^{-4}w = 2 \qquad \text{(Equation 2)}$$

$$\tan(x + y)(x - 2)^{-1}(x + 3)^{-1}y^{-2} = 1 \qquad \text{(Equation 3)}$$

Q: For what values of w and z does a simultaneous solution exist?



Q: Are there kids who might have fun playing with something like *this*?

Game: theory of a group, ring, etc.

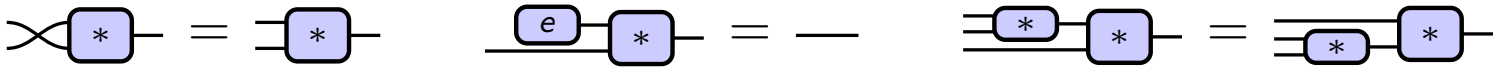
Monoids, groups, rings, modules: each has an associated algebraic theory.

- Choose any algebraic theory, or more generally regular theory.

Game: theory of a group, ring, etc.

Monoids, groups, rings, modules: each has an associated algebraic theory.

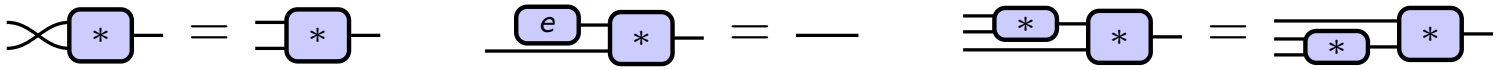
- Choose any algebraic theory, or more generally regular theory.
  - Let's say the theory of commutative monoids. What does that mean?
    - Have a unit cell e and a multiplication cell $*$
    - These satisfy equations (more generally, regular axioms)



Game: theory of a group, ring, etc.

Monoids, groups, rings, modules: each has an associated algebraic theory.

- Choose any algebraic theory, or more generally regular theory.
  - Let's say the theory of commutative monoids. What does that mean?
    - Have a unit cell e and a multiplication cell $*$
    - These satisfy equations (more generally, regular axioms)

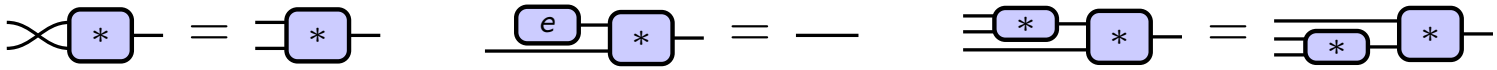


  - Like in Smart witter, maybe machines can learn common moves (?)
    - Note that common proof strategies are often used.
    - Game designer could program in things like $x * e = x$ simplifications.

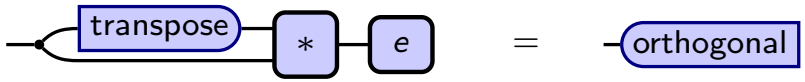
Game: theory of a group, ring, etc.

Monoids, groups, rings, modules: each has an associated algebraic theory.

- Choose any algebraic theory, or more generally regular theory.
  - Let's say the theory of commutative monoids. What does that mean?
    - Have a unit cell e and a multiplication cell $*$
    - These satisfy equations (more generally, regular axioms)



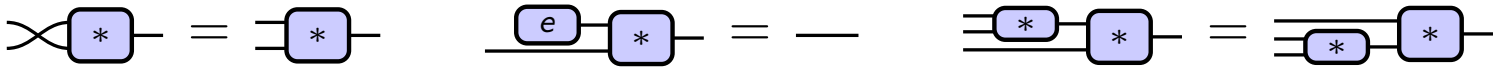
  - Like in Smart witter, maybe machines can learn common moves (?)
    - Note that common proof strategies are often used.
    - Game designer could program in things like $x * e = x$ simplifications.
- Within the game, create new cells and manipulate them.
  - E.g. add orthogonal and transpose and axiomatize them.



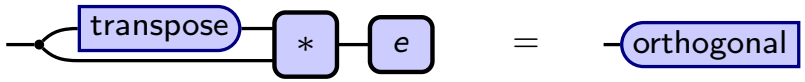
Game: theory of a group, ring, etc.

Monoids, groups, rings, modules: each has an associated algebraic theory.

- Choose any algebraic theory, or more generally regular theory.
  - Let's say the theory of commutative monoids. What does that mean?
    - Have a unit cell e and a multiplication cell $*$
    - These satisfy equations (more generally, regular axioms)



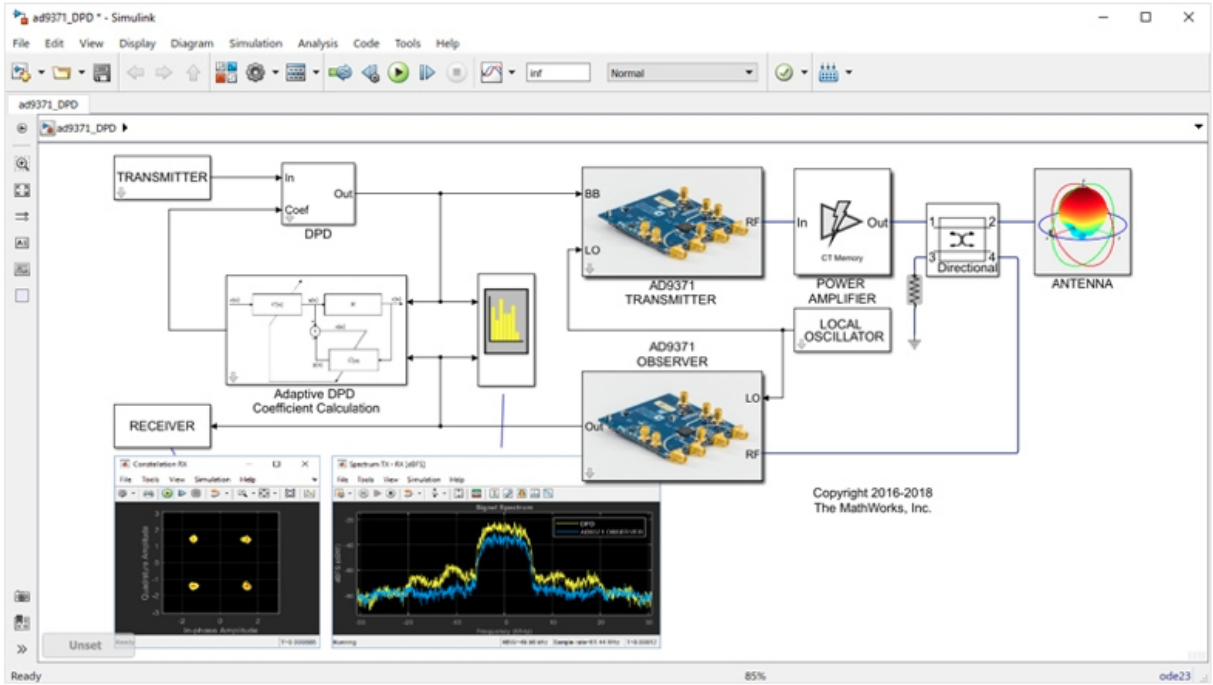
  - Like in Smart witter, maybe machines can learn common moves (?)
    - Note that common proof strategies are often used.
    - Game designer could program in things like $x * e = x$ simplifications.
- Within the game, create new cells and manipulate them.
  - E.g. add orthogonal and transpose and axiomatize them.



Like "fold-it" for protein folding, players can help w/o understanding.

Game: Simulink

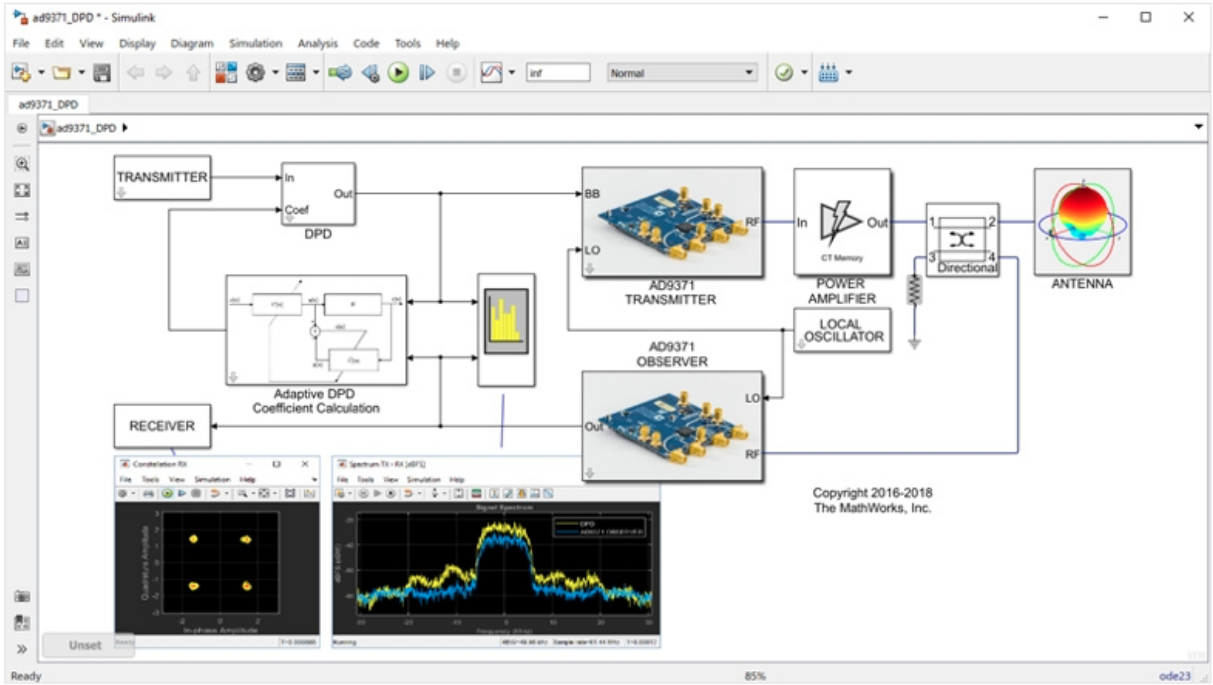
- Simulink (makers of Matlab): model and simulate dynamic systems.



- Connect up smaller dynamic systems.

Game: Simulink

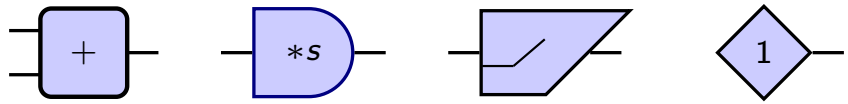
- Simulink (makers of Matlab): model and simulate dynamic systems.



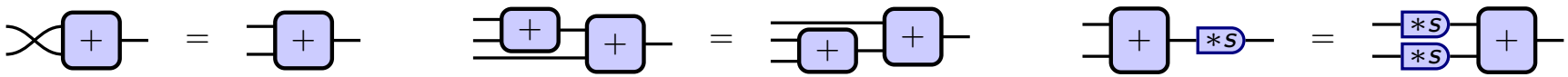
- Connect up smaller dynamic systems.
- Each can be understood as a relation in the temporal topos (TTT).
- Use reglog – the game as a shell for Simulink.

Game: neural relations

Neural networks are formed out of addition, scalar mult, ReLU, and 1.

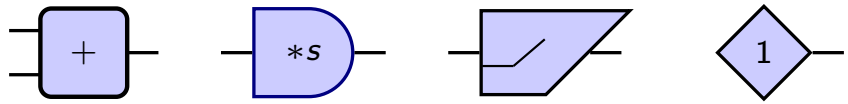


These satisfy various relations, e.g.

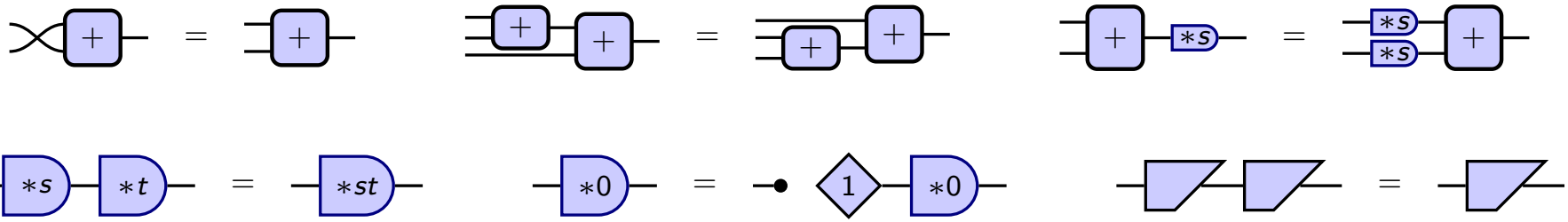


Game: neural relations

Neural networks are formed out of addition, scalar mult, ReLU, and 1.

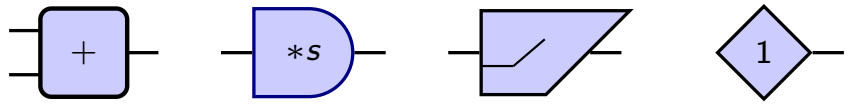


These satisfy various relations, e.g.

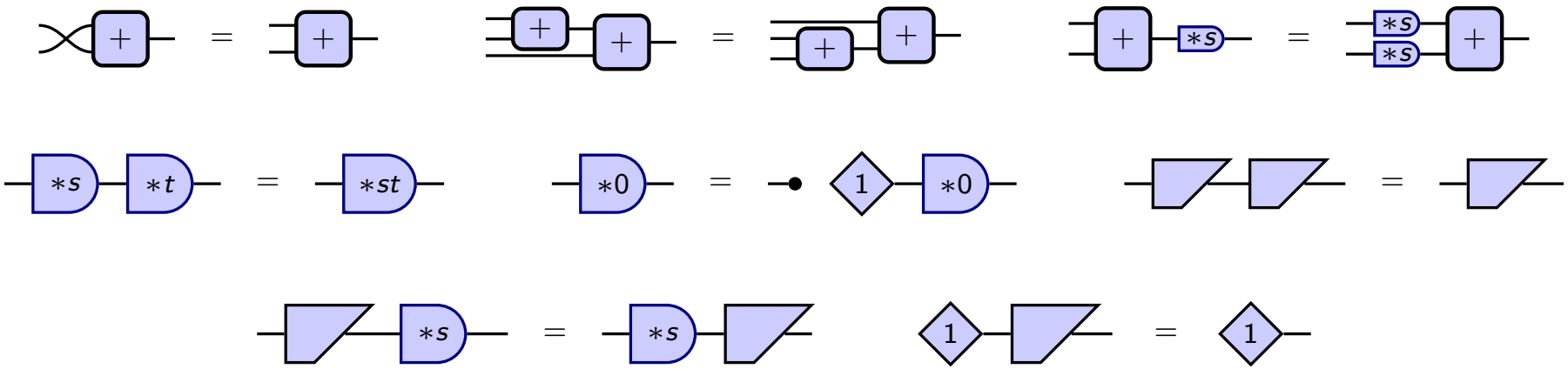


Game: neural relations

Neural networks are formed out of addition, scalar mult, ReLU, and 1.

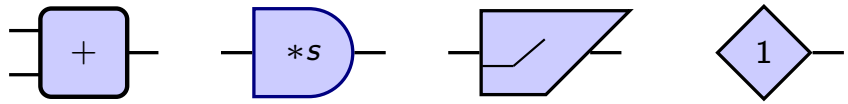


These satisfy various relations, e.g.

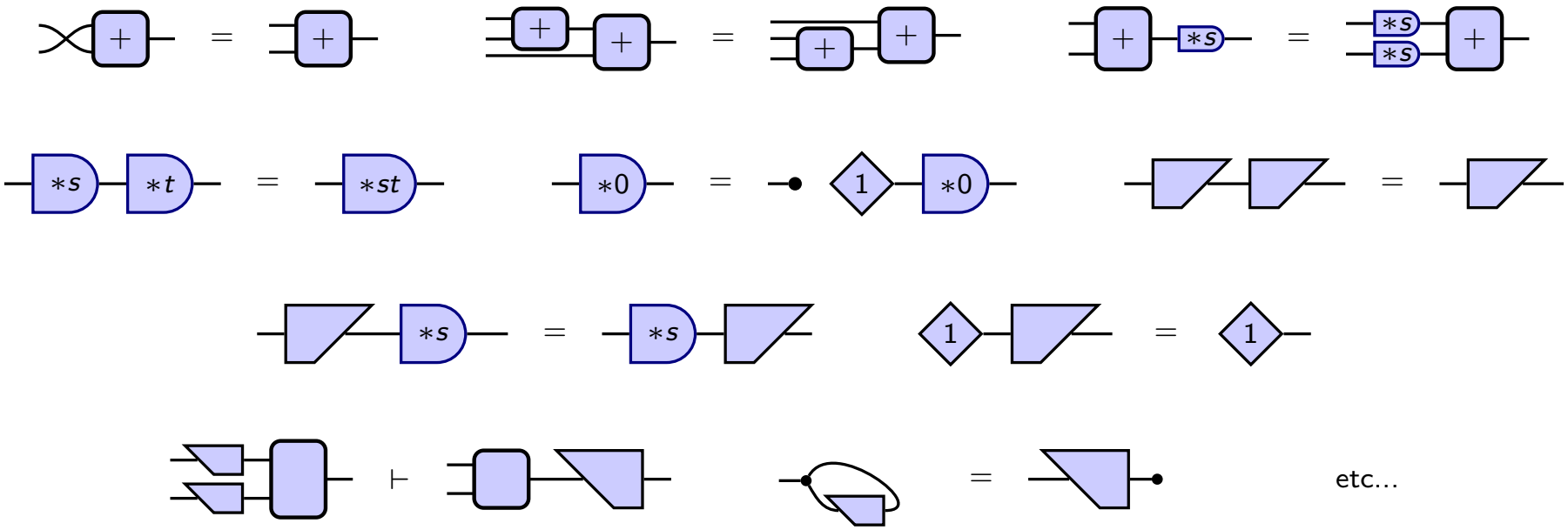


Game: neural relations

Neural networks are formed out of addition, scalar mult, ReLU, and 1.

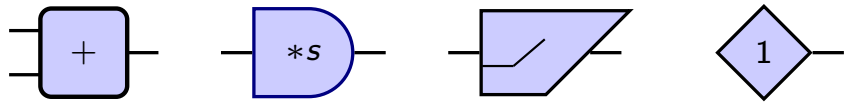


These satisfy various relations, e.g.

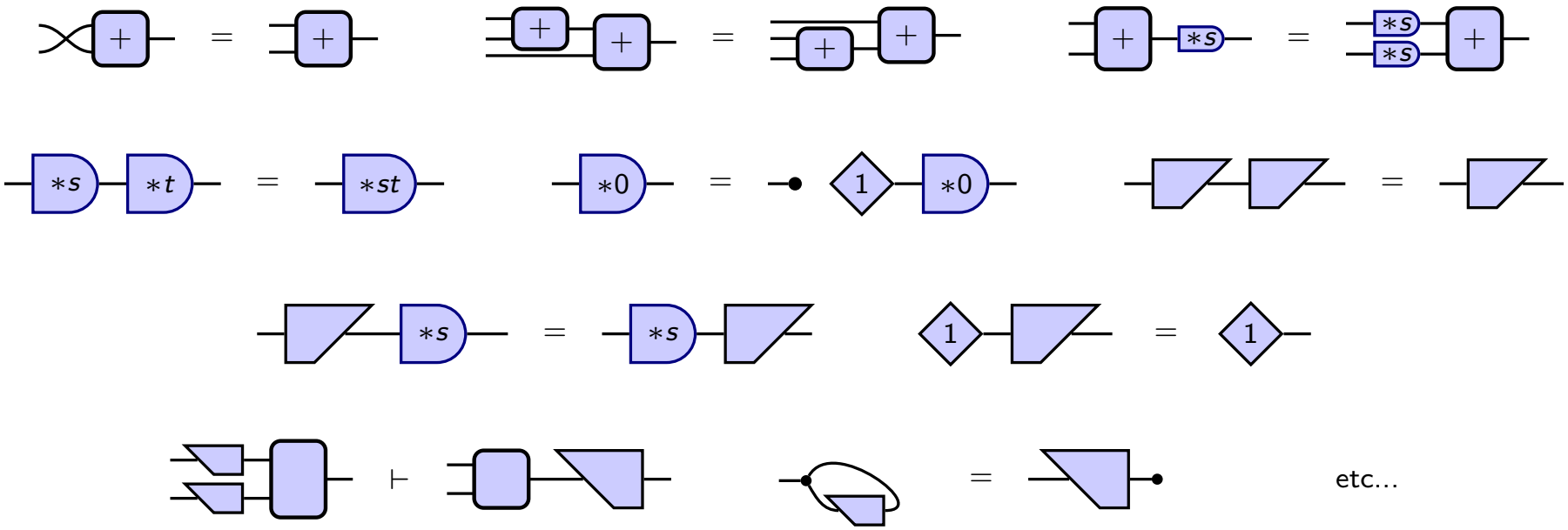


Game: neural relations

Neural networks are formed out of addition, scalar mult, ReLU, and 1.



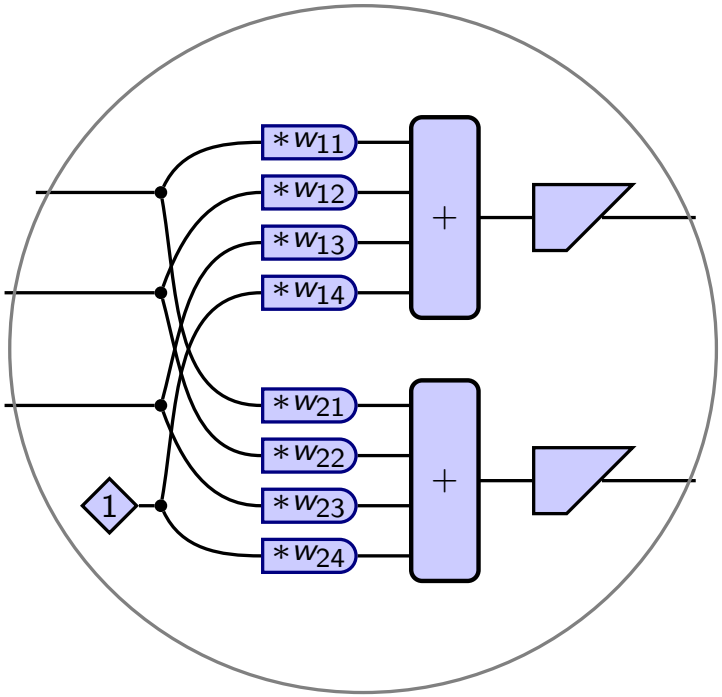
These satisfy various relations, e.g.



One could presumably work out a sound and complete set of axioms.

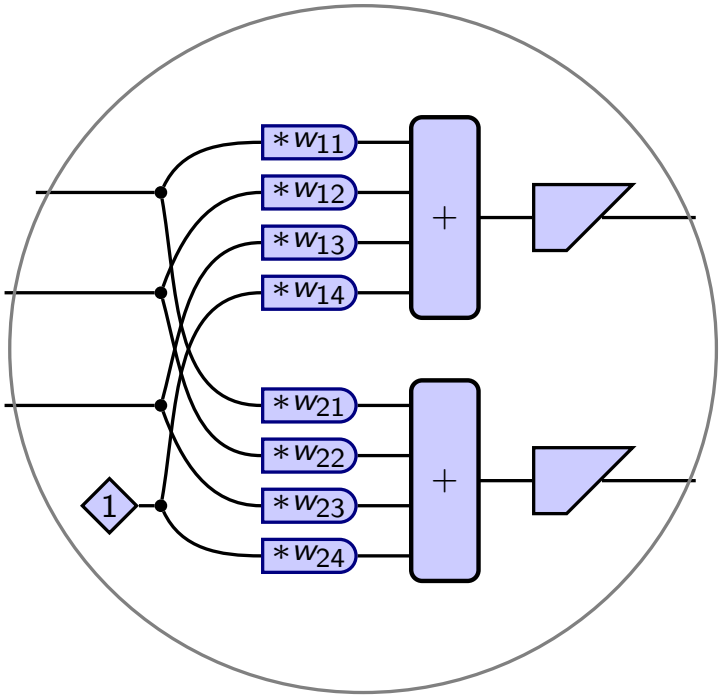
Game: neural relations

Users play with neural networks:

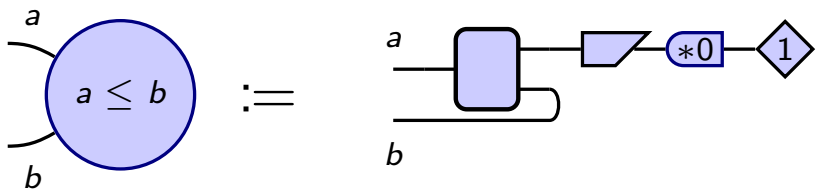


Game: neural relations

Users play with neural networks:



And invent other relations if they feel like it:



Outline

Can we make this real?

I'd love to see something like this be implemented.

- Users can get comfortable with one platform for many "games".
- The background math is complete.
  - Pick types T, pick ajax functor $P\colon \mathbb{C}\mathrm{ospan}_{\mathsf{T}}^{\mathrm{co}} \longrightarrow \mathbb{P}\mathrm{oset}$.
  - Each pair (T, P) constitutes a different game.
  - Playing the game involves wiring up cells and seeing how they relate.

Can we make this real?

I'd love to see something like this be implemented.

- Users can get comfortable with one platform for many "games".
- The background math is complete.
  - Pick types T, pick ajax functor $P\colon \mathbb{C}\mathrm{ospan}_{\mathsf{T}}^{\mathrm{co}} \longrightarrow \mathbb{P}\mathrm{oset}$.
  - Each pair (T, P) constitutes a different game.
  - Playing the game involves wiring up cells and seeing how they relate.
  - There are as many games as their are regular categories.
  - There are of course morphisms of games $(F\colon \mathsf{T} \to \mathsf{T}', F^\sharp\colon P \to P' \circ F)$.

Can we make this real?

I'd love to see something like this be implemented.

- Users can get comfortable with one platform for many "games".
- The background math is complete.
  - Pick types T, pick ajax functor $P\colon \mathbb{C}\mathrm{ospan}_{\mathsf{T}}^{\mathrm{co}} \longrightarrow \mathbb{P}\mathrm{oset}$.
  - Each pair (T, P) constitutes a different game.
  - Playing the game involves wiring up cells and seeing how they relate.
  - There are as many games as their are regular categories.
  - There are of course morphisms of games $(F\colon \mathsf{T} \to \mathsf{T}', F^\sharp\colon P \to P' \circ F)$.
- Seems like implementation would benefit a lot from dependent types.

Can we make this real?

I'd love to see something like this be implemented.

- Users can get comfortable with one platform for many "games".
- The background math is complete.
  - Pick types T, pick ajax functor $P\colon \mathbb{C}\mathrm{ospan}_{\mathsf{T}}^{\mathrm{co}} \longrightarrow \mathbb{P}\mathrm{oset}$.
  - Each pair (T, P) constitutes a different game.
  - Playing the game involves wiring up cells and seeing how they relate.
  - There are as many games as their are regular categories.
  - There are of course morphisms of games $(F\colon \mathsf{T} \to \mathsf{T}', F^\sharp\colon P \to P' \circ F)$.
- Seems like implementation would benefit a lot from dependent types.

If in the future you have questions or ideas—about this or anything category-theoretic—feel free to email me (dspivak@gmail.com).

Can we make this real?

I'd love to see something like this be implemented.

- Users can get comfortable with one platform for many "games".
- The background math is complete.
  - Pick types T, pick ajax functor $P\colon \mathbb{C}\mathrm{ospan}_{\mathsf{T}}^{\mathrm{co}} \longrightarrow \mathbb{P}\mathrm{oset}$.
  - Each pair (T, P) constitutes a different game.
  - Playing the game involves wiring up cells and seeing how they relate.
  - There are as many games as their are regular categories.
  - There are of course morphisms of games $(F\colon \mathsf{T} \to \mathsf{T}', F^\sharp\colon P \to P' \circ F)$.
- Seems like implementation would benefit a lot from dependent types.

If in the future you have questions or ideas—about this or anything category-theoretic—feel free to email me (dspivak@gmail.com).

Thanks! Questions and comments welcome!!