Reglog – the game

David I. Spivak

2019/04/11

MIT ACT seminar

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.

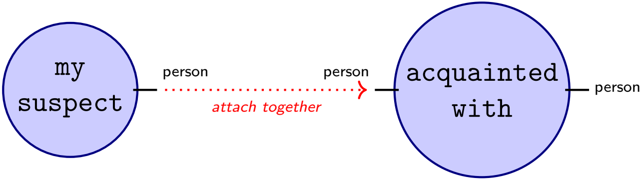
Brought to you by... regular logic – the game!

Working with logic

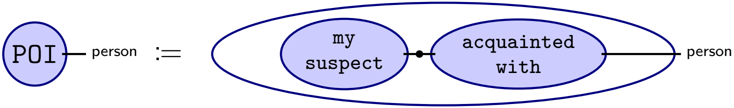
- Imagine you are an investigator on a case.
- You're adding to and narrowing down your set of suspects.
- You can pull up cells from the computer's database.

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.

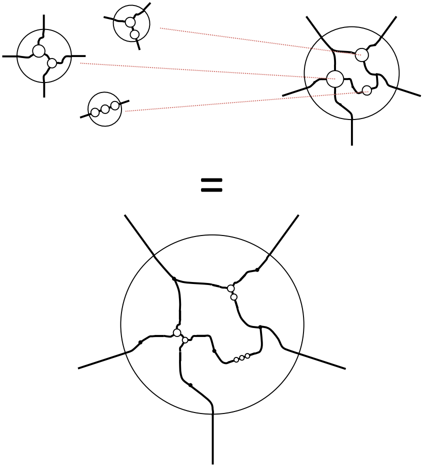


- and define POI (person of interest) as the result:



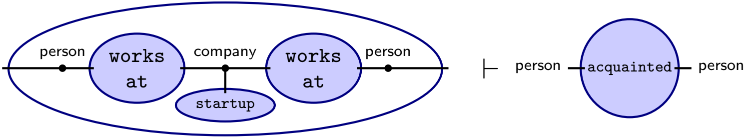
Compositionality

Of course, these concepts can be nested.



Adding beliefs

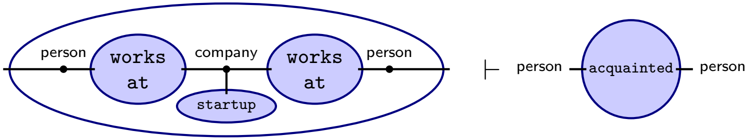
You can add beliefs about the world.



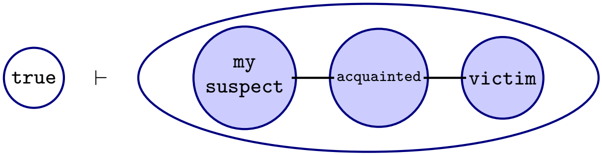
Belief: “If two persons work at the same startup, they are acquainted.”

Adding beliefs

You can add beliefs about the world.



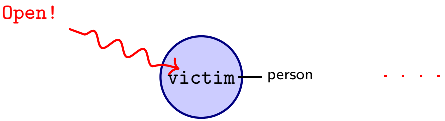
Belief: "If two persons work at the same startup, they are acquainted."



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

Accessing data

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

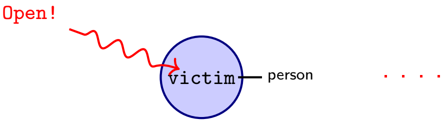


victim
person
P105
P820

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

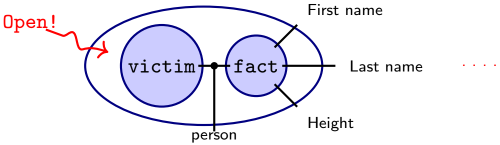
Accessing data

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



victim
person
P105
P820

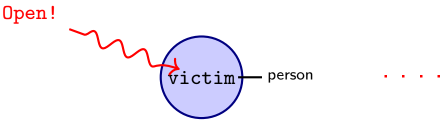
Persons are internal identifiers; we want to see facts about the victims.



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

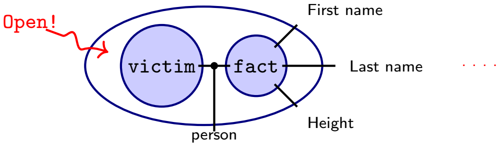
Accessing data

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.

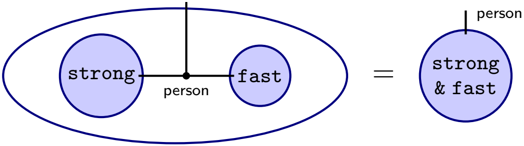


victim fact			
person	First	Last	Height
P105	Alice	Hu	5'10" – 6'00"
P820	Bob	Barr	P820.Height

Some knowledge is missing or otherwise imperfect.

Reasoning

The machine knows basic logical reasoning.

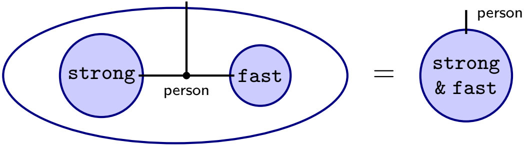


Manipulate diagrams by ...

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

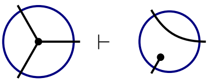
Reasoning

The machine knows basic logical reasoning.

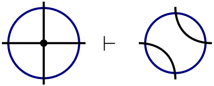


Manipulate diagrams by ...

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

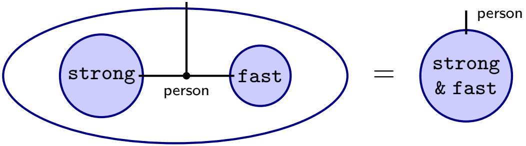


$\{(x, y, z) \mid x = y = z\} \subseteq \{(x, y, z) \mid x = y\}$



Reasoning

The machine knows basic logical reasoning.



Manipulate diagrams by ...

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

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

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 use the venerable 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 use the venerable chase...

- ... 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 use the venerable chase...

- ... 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 use the venerable chase...

- ... 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 use the venerable chase...

- ... 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."

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

- also known as: 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 use the venerable chase...

- ... 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."

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

- also known as: 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 hook it up to your car's autonomous driver, 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 hook it up to your car's autonomous driver, 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 hook it up to your car's autonomous driver, 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 hook it up to your car's autonomous driver, 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 it connects to the GUI described above.
- I'll end by giving several other games, besides "detective".

Outline

Regular logic and regular categories

Regular logic is the internal logic of regular categories.

- Regular categories are categories $\mathcal{R}$ with
  - finite limits (terminal object 1 and pullbacks), and
  - pullback-stable image factorizations.

Regular logic and regular categories

Regular logic is the internal logic of regular categories.

- Regular categories are categories $\mathcal{R}$ with
  - finite limits (terminal object 1 and pullbacks), and
  - pullback-stable image factorizations.
- Say $\mathcal{R}$ is *fully-inhabited* if $\mathcal{R}(1, r) \neq \varnothing$ for each $r \in \mathcal{R}$.
  - Set is a regular category, but not fully inhabited.
  - The category of pointed sets is fully inhabited.
  - Have categories $\mathsf{RegCat}_* \subseteq \mathsf{RegCat}$ of (fully-inhabited) regular cats.

Examples of regular categories:

- Set, and more generally any topos;
- $\mathsf{Set}^{\mathsf{op}}$, opposite of any topos, also $\mathsf{TopSp}^{\mathsf{op}}$;
- 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.

How to think of regular categories

Regular categories $\mathcal{R}$ are those with a good *bicategory 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...

How to think of regular categories

Regular categories $\mathcal{R}$ are those with a good *bicategory 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 bicategory $\mathbb{R}\mathrm{el}_{\mathcal{R}}$.
  - That is, relations can be composed (WDs) and compared ($\vdash$).
  - One can recover $\mathcal{R}$ as the category of adjunctions in $\mathbb{R}\mathrm{el}_{\mathcal{R}}$!

How to think of regular categories

Regular categories $\mathcal{R}$ are those with a good *bicategory 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 bicategory $\mathbb{R}\mathrm{el}_{\mathcal{R}}$.
  - 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.

How to think of regular categories

Regular categories $\mathcal{R}$ are those with a good *bicategory 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 bicategory $\mathbb{R}\mathrm{el}_{\mathcal{R}}$.
  - 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: $\exists y.\, R(x, y) \wedge S(y, z)$.

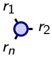
Graphical regular logic

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.

Graphical regular logic

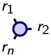
Think of regular categories as having good notion of *relation*.

- 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.
- 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

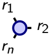
Think of regular categories as having good notion of *relation*.

- 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.
- 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 (filled Γ-shaped shell) for every subobject $c \subseteq r_1 \times \cdots \times r_n$.
  - Wiring diagrams denote combinations of finite limits and images.

Graphical regular logic

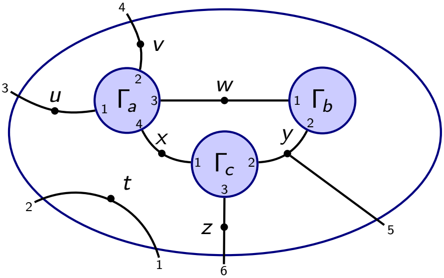
Think of regular categories as having good notion of *relation*.

- 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.
- 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 (filled Γ-shaped shell) 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.

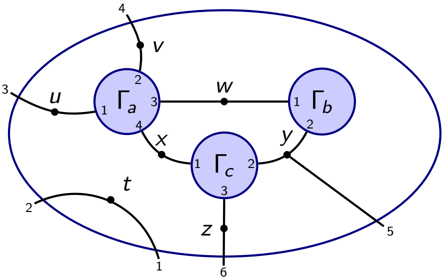
Mathematical specification of wiring diagrams

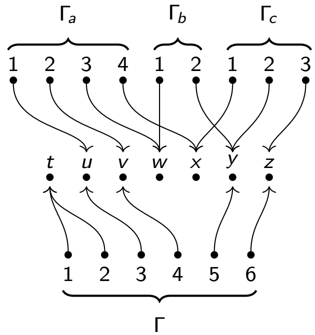
Mathematically, what is a wiring diagram $(\Gamma_a, \Gamma_b, \Gamma_c) \to \Gamma$?.



Mathematical specification of wiring diagrams

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



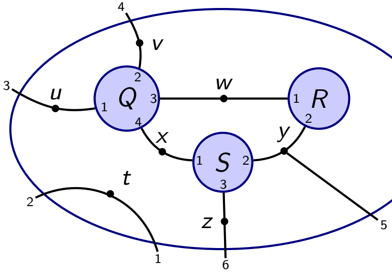


It is a morphism $(\Gamma_a, \Gamma_b, \Gamma_c) \to \Gamma$ in the operad of corelations.

- Another viewpoint: it is an equivalence relation on $\Gamma_a \sqcup \Gamma_b \sqcup \Gamma_c \sqcup \Gamma$.

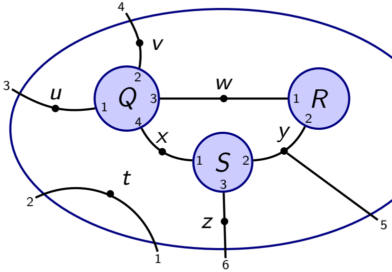
Wiring diagrams as logical expressions

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, 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.

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

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 monoidal 2-category $\mathbb{C}\mathsf{orel}_{\mathsf{T}} = \mathbb{R}\mathsf{el}(\mathsf{Finset}_{\mathsf{T}}^{\mathsf{op}})$.

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 monoidal 2-category $\mathbb{C}\mathrm{orel}_{\mathsf{T}} = \mathbb{R}\mathrm{el}(\mathrm{Finset}_{\mathsf{T}}^{\mathrm{op}})$.
  - Objects: arities $\underline{n} \xrightarrow{t} \mathsf{T}$, i.e. lists $\Gamma = (t(1), \ldots, t(n)) \in \mathsf{T}^n$.
  - Monoidal structure: concatenate lists.
  - 1-morphisms $(n_1, t_1) \to (n_2, t_2)$: partitions of $\underline{n_1 + n_2}$, respecting types
  - 2-morphisms: refinement (discrete partition is largest).

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 monoidal 2-category $\mathbb{C}\mathsf{orel}_{\mathsf{T}} = \mathbb{R}\mathsf{el}(\mathsf{Finset}_{\mathsf{T}}^{\mathsf{op}})$.
  - Objects: arities $\underline{n} \xrightarrow{t} \mathsf{T}$, i.e. lists $\Gamma = (t(1), \ldots, t(n)) \in \mathsf{T}^n$.
  - Monoidal structure: concatenate lists.
  - 1-morphisms $(n_1, t_1) \to (n_2, t_2)$: partitions of $\underline{n_1 + n_2}$, respecting types
  - 2-morphisms: refinement (discrete partition is largest).
- Consider the monoidal 2-category $\mathbb{P}\mathsf{oset}$.
  - 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 monoidal 2-category $\mathbb{C}\mathsf{orel}_{\mathsf{T}} = \mathbb{R}\mathsf{el}(\mathsf{Finset}_{\mathsf{T}}^{\mathsf{op}})$.
  - Objects: arities $\underline{n} \xrightarrow{t} \mathsf{T}$, i.e. lists $\Gamma = (t(1), \ldots, t(n)) \in \mathsf{T}^n$.
  - Monoidal structure: concatenate lists.
  - 1-morphisms $(n_1, t_1) \to (n_2, t_2)$: partitions of $\underline{n_1 + n_2}$, respecting types
  - 2-morphisms: refinement (discrete partition is largest).
- Consider the monoidal 2-category $\mathbb{P}\mathsf{oset}$.
  - Obj: posets; 1-morphisms: monotone maps; 2-morphisms: nat. trans.
  - Monoidal structure: $(1, \times)$.
- We will consider certain functors $\Phi\colon \mathbb{C}\mathsf{orel}_{\mathsf{T}} \to \mathbb{P}\mathsf{oset}$.
  - To each shell $\Gamma \in \mathbb{C}\mathsf{orel}_{\mathsf{T}}$, a poset $\Phi(\Gamma)$.
  - We denote the order in $\Phi(\Gamma)$ using the logical *entailment* symbol $\vdash$.

Formal specification of graphical calculi II

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

Definition

An *(inhabited) regular calculus* is a lax monoidal 2-functor

$$\Phi\colon \mathbb{C}\mathrm{orel}_{\mathsf{T}} \to \mathbb{P}\mathrm{oset}$$

such that the laxators are right adjoints.

Formal specification of graphical calculi II

We have monoidal 2-categories $\mathbb{C}$orel and $\mathbb{P}$oset.

Definition

An *(inhabited) regular calculus* is a lax monoidal 2-functor

$$\Phi\colon \mathbb{C}\mathsf{orel}_{\mathsf{T}} \to \mathbb{P}\mathsf{oset}$$

such that the laxators are right adjoints.

Our terminology: *ajax* monoidal functors: the laxators

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

Formal specification of graphical calculi II

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

Definition

An *(inhabited) regular calculus* is a lax monoidal 2-functor

$$\Phi\colon \mathbb{C}\text{orel}_{\mathsf{T}} \to \mathbb{P}\text{oset}$$

such that the laxators are right adjoints.

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

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

Regular calculi and regular categories

Denote by $\mathbb{C}\mathsf{orel}\text{-}\mathsf{Alg}$ the category of inhabited regular calculi

$$\mathbb{C}\mathsf{orel}\text{-}\mathsf{Alg} := \big(\big(\mathsf{T}, \Phi\colon \mathbb{C}\mathsf{orel}_{\mathsf{T}} \to \mathbb{P}\mathsf{oset}\big)\big).$$

Regular calculi and regular categories

Denote by $\mathbb{C}\mathsf{orel}\text{-}\mathsf{Alg}$ the category of inhabited regular calculi

$$\mathbb{C}\mathsf{orel}\text{-}\mathsf{Alg} := \big(\big(T, \Phi\colon \mathbb{C}\mathsf{orel}_T \to \mathbb{P}\mathsf{oset}\big)\big).$$

Theorem

There is an adjunction

$$\mathbb{C}\mathsf{orel}\text{-}\mathsf{Alg} \underset{\Psi}{\overset{\Phi}{\underset{\longleftarrow}{\overset{\longrightarrow}{\Rightarrow}}}} \mathsf{RegCat}_* \ ,$$

such that for any fully-inhabited regular category $\mathcal{R}$, the counit $\Psi(\Phi(\mathcal{R})) \to \mathcal{R}$ is an equivalence of categories.

Regular calculi and regular categories

Denote by $\mathbb{C}$orel-Alg the category of inhabited regular calculi

$$\mathbb{C}\mathrm{orel}\text{-}\mathrm{Alg} := \big(\big(\mathsf{T}, \Phi \colon \mathbb{C}\mathrm{orel}_{\mathsf{T}} \to \mathbb{P}\mathrm{oset}\big)\big).$$

Theorem

There is an adjunction

$$\mathbb{C}\mathrm{orel}\text{-}\mathrm{Alg} \underset{\Psi}{\overset{\Phi}{\rightleftarrows}} \mathrm{RegCat}_* \; ,$$

such that for any fully-inhabited regular category $\mathcal{R}$, the counit $\Psi(\Phi(\mathcal{R})) \to \mathcal{R}$ is an equivalence of categories.

A similar theorem holds when RegCat_* is replaced by RegCat: `arxiv.org/abs/1812.05765`.

Aside: operadic vs. monoidal

There are two versions of the whole story.

- Operadic version: nice pictures.
- Monoidal version: more familiar, better notation.

Aside: operadic vs. monoidal

There are two versions of the whole story.

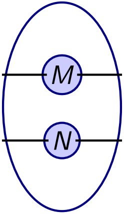
- Operadic version: nice pictures.
- Monoidal version: more familiar, better notation.
- I'll switch back and forth between them.

Aside: operadic vs. monoidal

There are two versions of the whole story.

- Operadic version: nice pictures.
- Monoidal version: more familiar, better notation.
- I'll switch back and forth between them.

The translation uses "shells in a trench coat".



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 $\Phi\colon \mathbb{C}\mathsf{orel}_{\mathsf{T}} \to \mathbb{P}\mathsf{oset}$.
- What is $\mathbb{C}\mathsf{orel}_{\mathsf{T}}$?

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 $\Phi\colon \mathbb{C}\mathrm{orel}_{\mathsf{T}} \to \mathbb{P}\mathrm{oset}$.
- What is $\mathbb{C}\mathrm{orel}_{\mathsf{T}}$?
  - Objects: drawn as shells  , encoded as lists $(t_1, \ldots, t_n)$.
  - Monoidal product: drawn as shells in a trench coat, encoded as list concat.
  - Morphisms: drawn as wiring diagrams, encoded as partitions.
  - 2-structure: drawn as breaking dots, encoded as refinement.

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 $\Phi\colon \mathbb{C}\mathsf{orel}_\mathsf{T} \to \mathbb{P}\mathsf{oset}$.
- What is $\mathbb{C}\mathsf{orel}_\mathsf{T}$?
  - Objects: drawn as shells  , encoded as lists $(t_1, \ldots, t_n)$.
  - Monoidal product: drawn as shells in a trench coat, encoded as list concat.
  - Morphisms: drawn as wiring diagrams, encoded as partitions.
  - 2-structure: drawn as breaking dots, encoded as refinement.
- Each game developer must supply their own Φ.
  - To each shell Γ, supply elts of $\Phi(\Gamma)$, 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 $\Phi\colon \mathbb{C}\mathrm{orel}_\mathsf{T} \to \mathbb{P}\mathrm{oset}$.
- What is $\mathbb{C}\mathrm{orel}_\mathsf{T}$?
  - Objects: drawn as shells  , encoded as lists $(t_1, \ldots, t_n)$.
  - Monoidal product: drawn as shells in a trench coat, encoded as list concat.
  - Morphisms: drawn as wiring diagrams, encoded as partitions.
  - 2-structure: drawn as breaking dots, encoded as refinement.
- Each game developer must supply their own Φ.
  - To each shell Γ, supply elts of $\Phi(\Gamma)$, drawn as cells (fillers).
  - Supply definition of $\varphi_1 \leq \varphi_2$, drawn perhaps as $\varphi_1 \vdash \varphi_2$.
  - To each wiring diagram w, supply monotonic function $\Phi(w)\colon \Phi(\Gamma_1) \times \cdots \times \Phi(\Gamma_n) \to \Phi(\Gamma')$.
  - Ensure Φ preserves composition, identity, and dot-breaking.

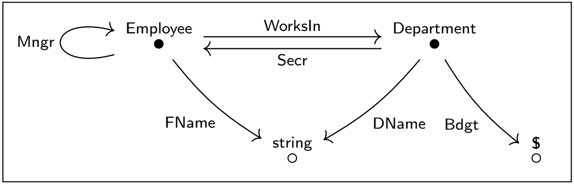
Backend: open source CQL

Backend: categorical query language, another approach to databases.

Backend: open source CQL

Backend: categorical query language, another approach to databases.

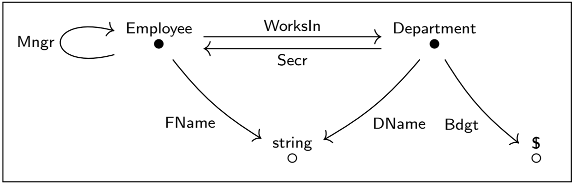
- A schema S is roughly a category and looks like this:



Backend: open source CQL

Backend: categorical query language, another approach to databases.

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

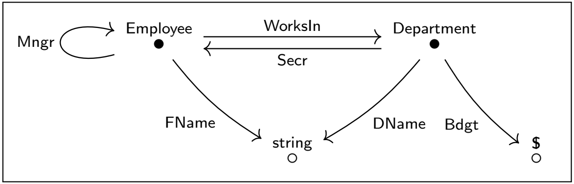


- An instance is roughly a functor $I\colon S \to \text{Set}$.

Backend: open source CQL

Backend: categorical query language, another approach to databases.

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

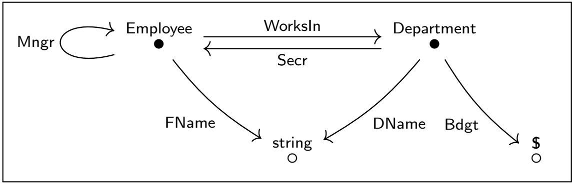


- An instance is roughly a functor $I \colon S \to \text{Set}$.
- Given schema S, consider each dot $s \in S$ as a shell and as a type.
  - Wires of shell s: its outgoing arrows $s \to t$ typed by target t.
  - If I is an S-instance, get filling cells as in detective case.

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 $I : S \to \mathsf{Set}$.
- Given schema S, consider each dot $s \in S$ as a shell and as a type.
  - Wires of shell s: its outgoing arrows $s \to t$ typed by target t.
  - If I is an S-instance, get filling cells as in detective case.
- Each regular sequent $\phi(\vec{x}) \vdash \psi(\vec{x})$ is called an embedded dependency.
- Chase these EDs to "repair data", forcing the axioms to hold.

Outline

Common interface

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

- In common: same GUI and common forms of interaction
  - Create new shells, attach shells, compose wiring diagrams
  - Zoom in and out of wiring diagrams.
  - Fill shells with cells
  - Click on cells (filled in shells) for interaction.
  - Do reasoning (entailment, substitution, dot-breaking)

Common interface

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

- In common: same GUI and common forms of interaction
  - Create new shells, attach shells, compose wiring diagrams
  - Zoom in and out of wiring diagrams.
  - Fill shells with cells
  - Click on cells (filled in shells) for interaction.
  - Do reasoning (entailment, substitution, dot-breaking)
- Differences between different games Φ:
  - Each Φ gives a world of cells that inhabit the shells.

Common interface

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

- In common: same GUI and common forms of interaction
    - Create new shells, attach shells, compose wiring diagrams
    - Zoom in and out of wiring diagrams.
    - Fill shells with cells
    - Click on cells (filled in shells) for interaction.
    - Do reasoning (entailment, substitution, dot-breaking)
- Differences between different games Φ:
    - Each Φ gives a world of cells that inhabit the shells.
- Some examples:
    - Where are my keys?
    - Smart witter
    - Partitions game
    - Boolean circuits
    - Solve equations
    - etc.

Game: Where are my keys?

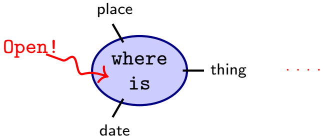
A simpler version of detective game: no reasoning $\vdash$, just query.

- 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.

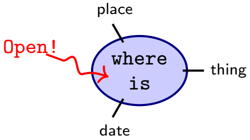


where is		
thing	place	date
keys	top drawer	2019/04/08
lease	file cabinet	2019/02/01
Mom's gift	basement	2019/03/30

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	2019/04/08
lease	file cabinet	2019/02/01
Mom's gift	basement	2019/03/30

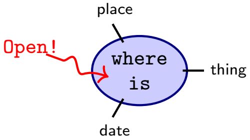
- Query using the graphical interface
  - Attach  to find places your keys were yesterday.

  - Similar interface 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.



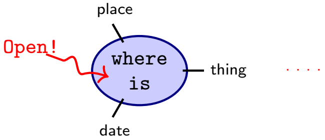
where is		
thing	place	date
keys	top drawer	2019/04/08
lease	file cabinet	2019/02/01
Mom's gift	basement	2019/03/30

- Query using the graphical interface
  - Attach thing –keys yesterday– date to find places your keys were yesterday.
  - Similar interface 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.



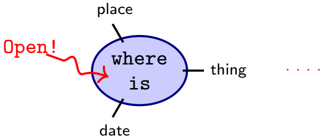
where is		
thing	place	date
keys	top drawer	2019/04/08
lease	file cabinet	2019/02/01
Mom's gift	basement	2019/03/30

- Query using the graphical interface
  - Attach thing –keys yesterday– date to find places your keys were yesterday.
  - Similar interface 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	2019/04/08
lease	file cabinet	2019/02/01
Mom's gift	basement	2019/03/30

- Query using the graphical interface
  - Attach thing –keys yesterday– date to find places your keys were yesterday.
  - Similar interface 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  .
  - Connect specialty –cardiologist to doctor's speciality port.
  - Connect its 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 just named, not inhabited.
- 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 just named, not inhabited.
- Idea: people blog concepts built from simple pieces.
  - Build "small world events" from basic parts (meet far from home)

Game: Smart witter

A smarter social network.

- This is another simplification of the detective game.
- Difference: cells are just named, not inhabited.
- Idea: people blog concepts built from simple pieces.
  - Build “small world events” from basic parts (meet far from home)
  - 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 just named, not inhabited.
- Idea: people blog concepts built from simple pieces.
  - Build "small world events" from basic parts (meet far from home)
  - Others can disassemble, reassemble, and nick-name concepts
  - Zoom into others' concepts to see what they really mean.
- What's trending? Concepts.
  - The machine finds concepts that people keep reusing.

Game: Smart witter

A smarter social network.

- This is another simplification of the detective game.
- Difference: cells are just named, not inhabited.
- Idea: people blog concepts built from simple pieces.
  - Build "small world events" from basic parts (meet far from home)
  - Others can disassemble, reassemble, and nick-name concepts
  - Zoom into others' concepts to see what they really mean.
- What's trending? Concepts.
  - The machine finds 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.

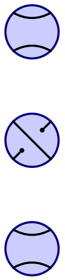
Game: Smart witter

A smarter social network.

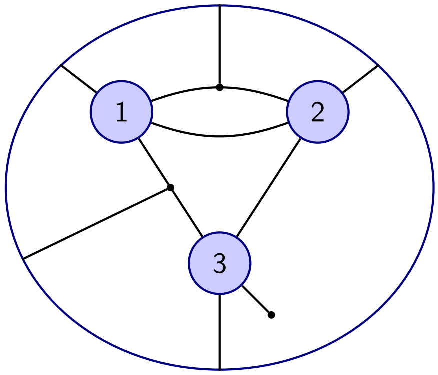
- This is another simplification of the detective game.
- Difference: cells are just named, not inhabited.
- Idea: people blog concepts built from simple pieces.
  - Build "small world events" from basic parts (meet far from home)
  - Others can disassemble, reassemble, and nick-name concepts
  - Zoom into others' concepts to see what they really mean.
- What's trending? Concepts.
  - The machine finds 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

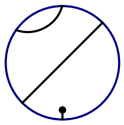
Put pieces:



into puzzle:

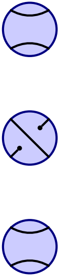


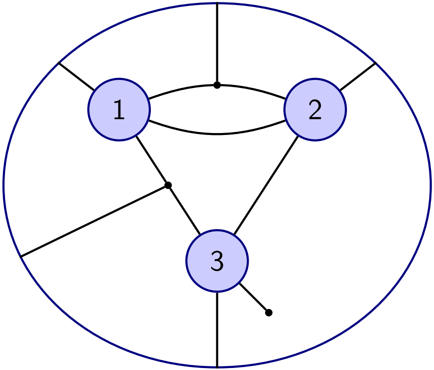
to obtain:

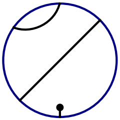


Game: Partitions puzzle

Put pieces: into puzzle: to obtain:

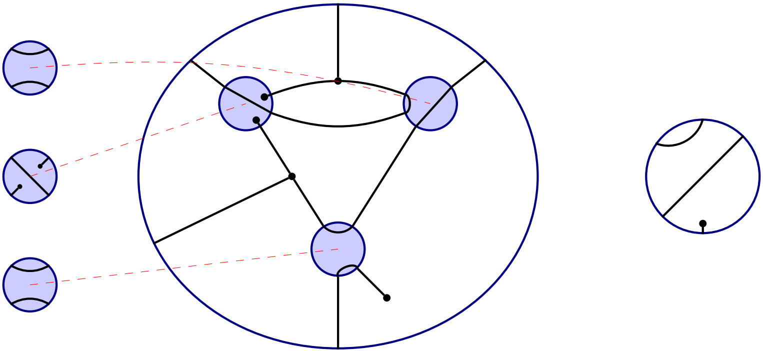






I'll give you a minute to solve it.

Game: partitions puzzle (solution)



Game: 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: 

Game: Solving equations

Consider an arbitrary system of equations having the following form:

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

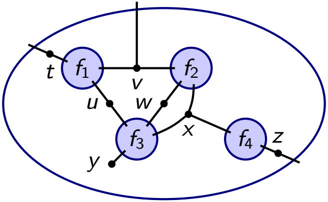
Bold variables are those we want to *expose*; others are latent or *unexposed*.

Game: Solving equations

Consider an arbitrary system of equations having the following form:

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

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 $g(x, y) = 0$.

- 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.

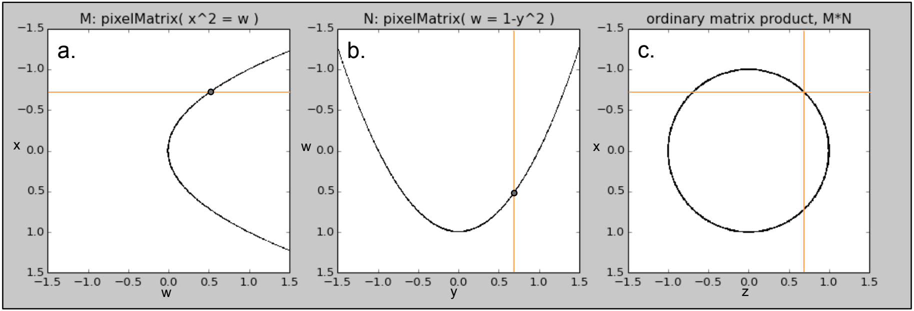
Systems of equations via pixel arrays

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

- 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

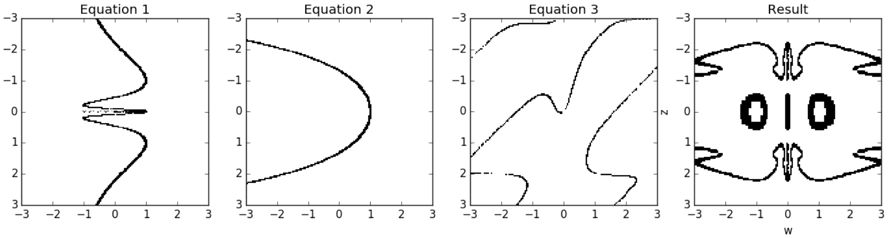
The following eq's are not differentiable, nor even defined everywhere.

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

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

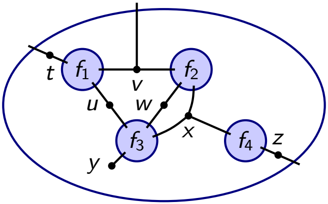
$$\tan(x+y)(x-2)^{-1}(x+3)^{-1}y^{-2} = 1 \qquad \text{(Equation 3)}$$

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



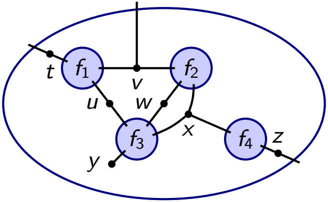
Game: systems of linear equations

PA is the backend for a game that plots sol'ns to arbitrary systems.



Game: systems of linear equations

PA is the backend for a game that plots sol'ns to arbitrary systems.



- Similar game: each cell f_i is a linear eq'n, e.g. $x_1 + 3x_2 - 2x_4 = 0$.
- Then the outer cell is a linear equation too.
- "Exponentially" smaller matrices to multiply.

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 monoids.
    - Have a multiplication cell 

 and a unit cell 

    - These satisfy various 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 monoids.
    - Have a multiplication cell 

 and a unit cell 

    - These satisfy various equations (more generally, regular axioms)
  - Like in Smart witter, machine could 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.

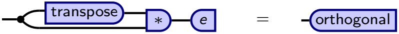
- Choose any algebraic theory, or more generally regular theory.
  - Let's say the theory of monoids.
    - Have a multiplication cell 

 and a unit cell 

    - These satisfy various equations (more generally, regular axioms)
  - Like in Smart witter, machine could 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 

 and 

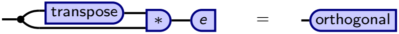
 and axiomatize them.



Game: theory of a group, ring, etc.

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

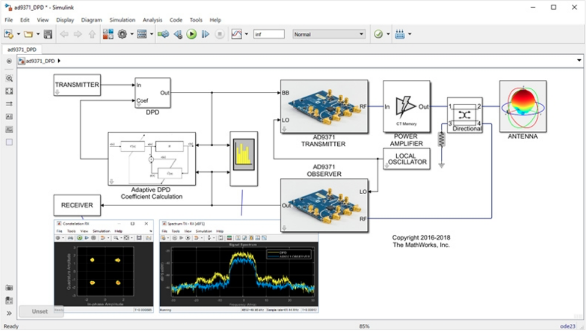
- Choose any algebraic theory, or more generally regular theory.
  - Let's say the theory of monoids.
    - Have a multiplication cell $*$ and a unit cell e
    - These satisfy various equations (more generally, regular axioms)
  - Like in Smart witter, machine could learn common moves.
    - Note that 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

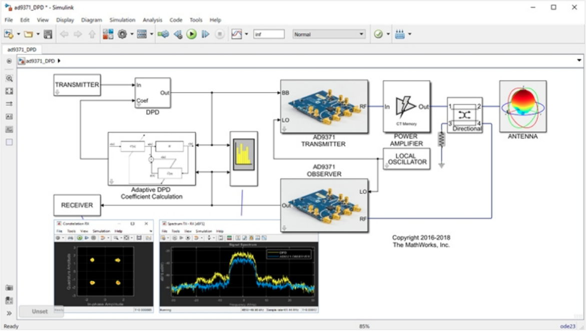
- Simulink (makers of Matlab): model and simulate dynamic systems.



- 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 an interface for Simulink.

Outline

Let's make this real!

We're ready to make this happen.

- The background math is complete
  - We understand the data structures involved.
  - Experience shows that coding it will uncover hidden assumptions.
  - Example: maybe player can "transform" typeset T and Φ, mid-game.
- We've seen some example games and roughly how they'd work.
- Creativity required to precise game specs and to create new games.

Let's make this real!

We're ready to make this happen.

- The background math is complete
  - We understand the data structures involved.
  - Experience shows that coding it will uncover hidden assumptions.
  - Example: maybe player can "transform" typeset T and Φ, mid-game.
- We've seen some example games and roughly how they'd work.
- Creativity required to precise game specs and to create new games.

How to proceed?

- I have some funding to get a programming effort started.
- Please contact me or Brendan if you're interested in contributing.

Let's make this real!

We're ready to make this happen.

- The background math is complete
  - We understand the data structures involved.
  - Experience shows that coding it will uncover hidden assumptions.
  - Example: maybe player can "transform" typeset T and Φ, mid-game.
- We've seen some example games and roughly how they'd work.
- Creativity required to precise game specs and to create new games.

How to proceed?

- I have some funding to get a programming effort started.
- Please contact me or Brendan if you're interested in contributing.

Thanks!