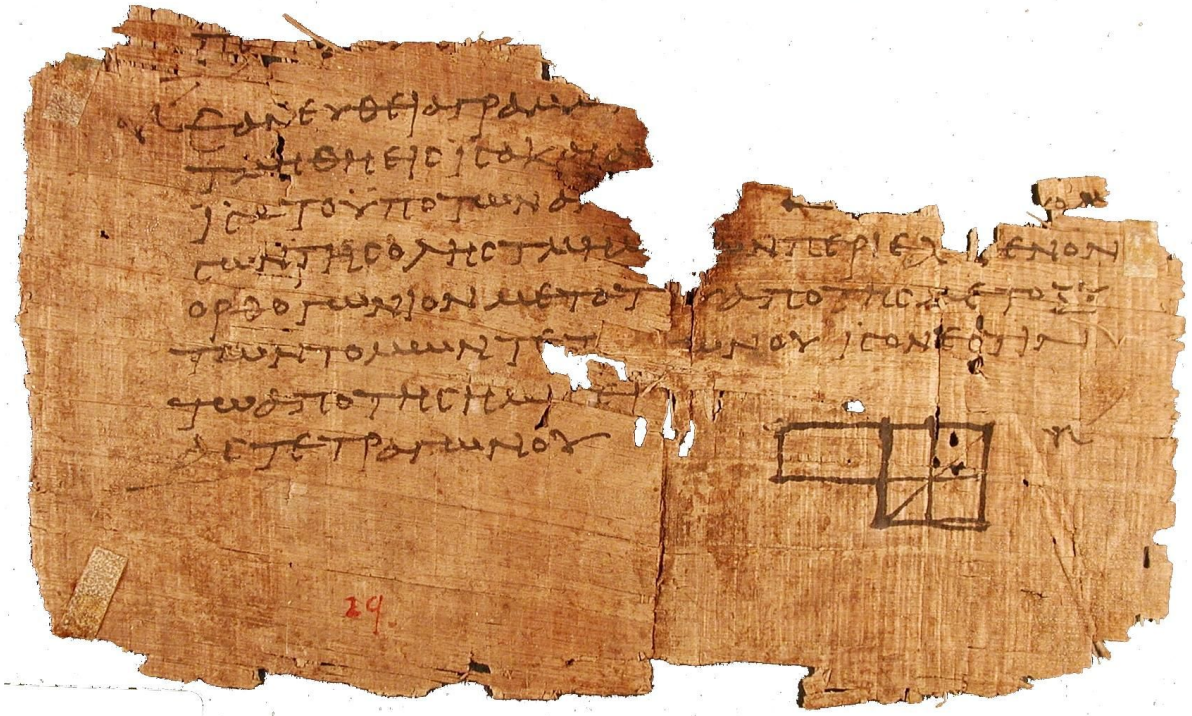
Diagrammatic notations for interactive theorem proving

Shardul Chiplunkar and Clément Pit-Claudel
Systems and Formalisms lab (SYSTEMF) @ EPFL

SVD 2024

Proofs through the ages

Proofs through the ages



Euclid's proof of

$$ab + \left(\frac{a-b}{2}\right)^2 = \left(\frac{a+b}{2}\right)^2$$

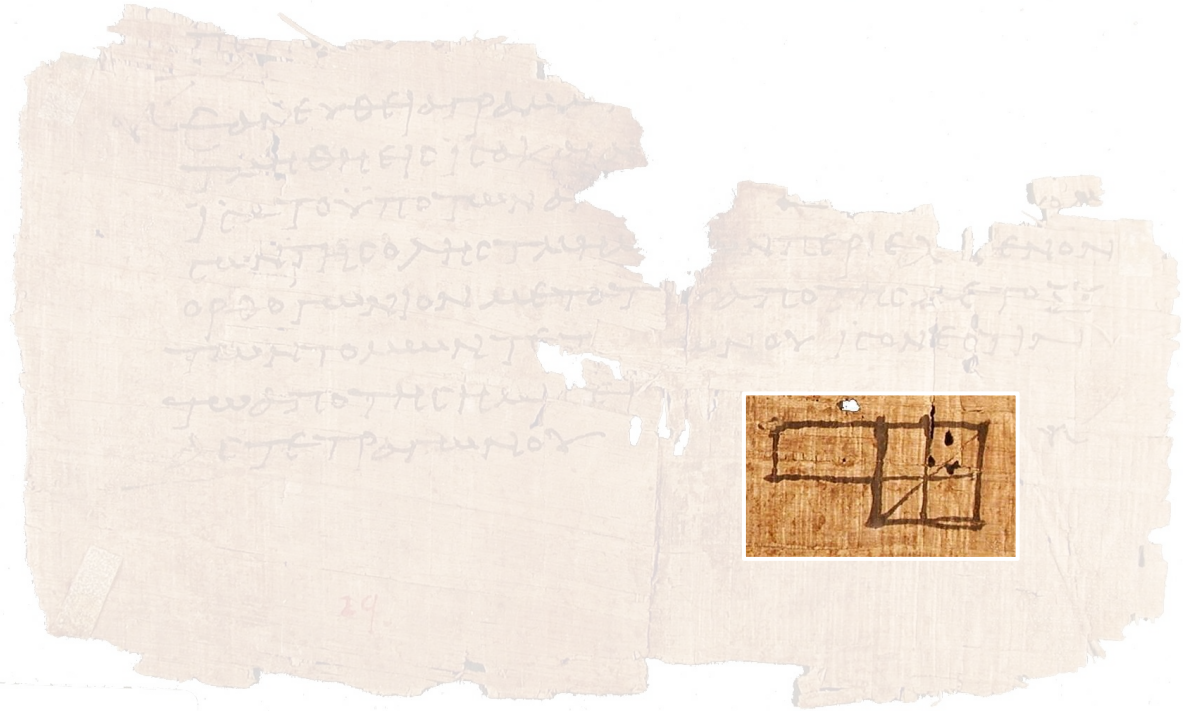
Elements, Euclid c. 300 BCE

scribbled on papyrus by unknown author, c. 100 CE

discovered by Grenfell and Hunt, 1896

photographed by Casselman, 1994

Proofs through the ages



Euclid's proof of

$$ab + \left(\frac{a-b}{2}\right)^2 = \left(\frac{a+b}{2}\right)^2$$

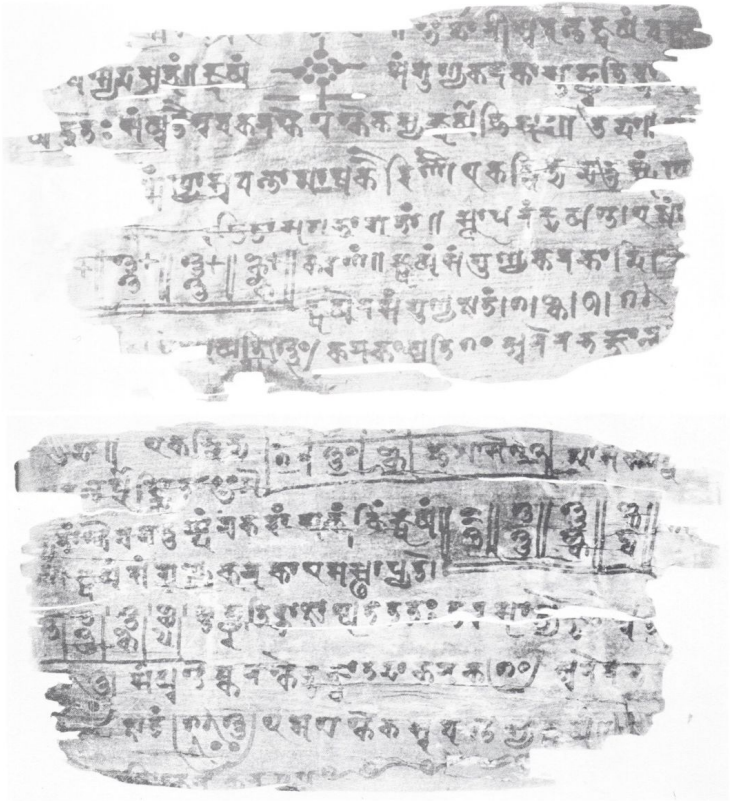
Elements, Euclid c. 300 BCE

scribbled on papyrus by unknown author, c. 100 CE

discovered by Grenfell and Hunt, 1896

photographed by Casselman, 1994

Proofs through the ages



solving cubic equations (?)

Bakhshali manuscript, c. 300–900 CE (disputed)
discovered by anonymous peasant, 1881
photographed by Kaye, 1927
from Plofker et al., 2017

Proofs through the ages

solving cubic equations (?)

Bakhshali manuscript, c. 300–900 CE (disputed)
discovered by anonymous peasant, 1881
photographed by Kaye, 1927
from Plofker et al., 2017

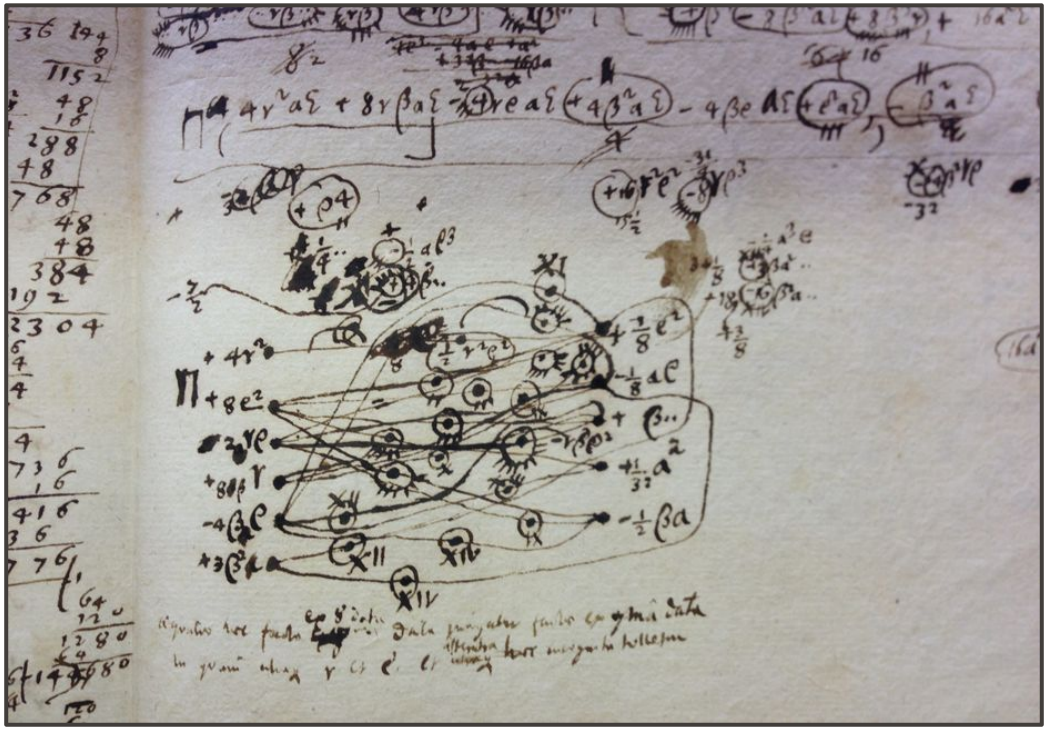
Proofs through the ages

solving cubic equations (?)

… and one of the oldest physical records of the numeral zero

Bakhshali manuscript, c. 300–900 CE (disputed)
discovered by anonymous peasant, 1881
photographed by Kaye, 1927
from Plofker et al., 2017

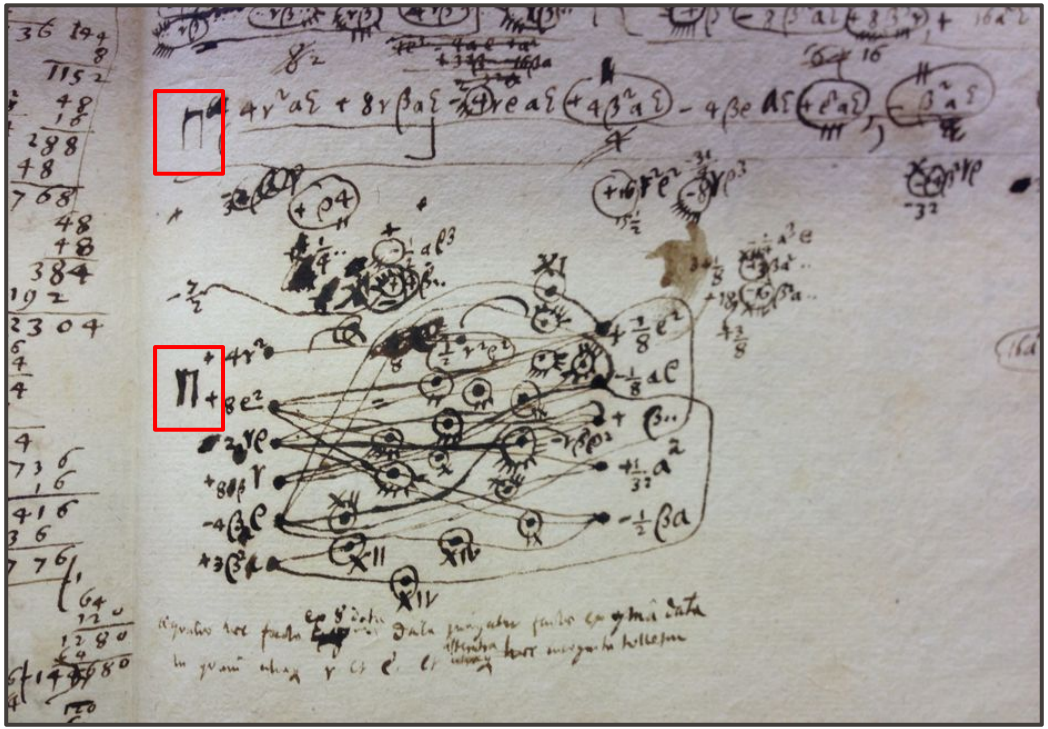
Proofs through the ages



Leibniz's diagrammatic calculus for… not sure exactly what

personal notes, Leibniz, c. 1675
photographed by Wolfram, 2013

Proofs through the ages



Leibniz's diagrammatic calculus for… not sure exactly what

personal notes, Leibniz, c. 1675
photographed by Wolfram, 2013

Proofs through the ages

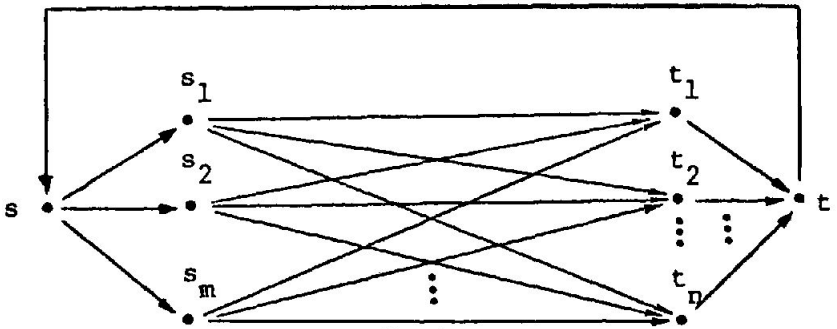


FIG. 1

THEOREM 8. *The flow f is extreme among maximum flows for the network of Figure 1, if and only if there exist* $u_0, u_1, \cdots, u_m$ *and* $v_0, v_1, \cdots, v_m$ *such that*

$$u_i - v_j + d_{ij} \geq 0,\ i = 1, 2, \cdots, m;\ j = 1, 2, \cdots, n, \tag{5a}$$

$$u_i - v_j + d_{ij} > 0 \Rightarrow f_{ij} = 0,\ i = 1, 2, \cdots, m;\ j = 1, 2, \cdots, n, \tag{5b}$$

$$u_0 > u_i \Rightarrow f_{0i} = 0, \tag{5c}$$

$$u_0 < u_i \Rightarrow f_{0i} = a_i, \tag{5d}$$

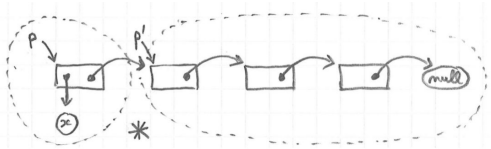
$$v_j > v_0 \Rightarrow f_{j0} = 0, \tag{5e}$$

$$v_j < v_0 \Rightarrow f_{j0} = b_j \tag{5f}$$

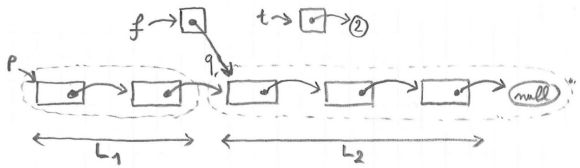
Edmonds-Karp algorithm
for max flow in a network

Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems, Edmonds and Karp, 1972

Proofs through the ages



$$p \rightsquigarrow \mathsf{Mlist}\, L \;\equiv\; \mathsf{match}\, L\, \mathsf{with}$$
$$\mid \mathsf{nil} \;\Rightarrow\; [p = \mathsf{null}]$$
$$\mid x :: L' \;\Rightarrow\; \exists p'.\quad p \rightsquigarrow \{\!|\mathsf{hd}{=}x;\ \mathsf{tl}{=}p'|\!\} \star p' \rightsquigarrow \mathsf{Mlist}\, L'$$



Loop invariant:

$$\exists L_1 L_2 q.\quad [L = L_1 +\!\!+ L_2] \star (t \mapsto \mathsf{length}\, L_1) \star (f \mapsto q) \star (p \rightsquigarrow \mathsf{MlistSeg}\, q\, L_1) \star (q \rightsquigarrow \mathsf{Mlist}\, L_2)$$

proof of `length` of a linked list with separation logic

Separation Logic, Charguéraud, 2016

Proofs through the ages

It’s 2023!

Proofs through the ages

It’s 2023!

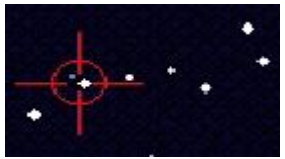
Proof assistants are becoming popular with mathematicians and computer scientists.

Proofs through the ages

It's 2023!

Proof assistants are becoming popular with mathematicians and computer scientists.

HOL
Interactive Theorem Prover



LEAN

Agda

Proofs through the ages

```
Lemma prop_14 : forall A B C D P Q R S T U,
    TS A B C D ->
    Per P Q R ->
    SumA A B C A B D S T U ->
    SumA P Q R P Q R S T U ->
    Bet C B D.
Proof.
  intros A B C D P Q R S T U HTS HP HSuma1 HSuma2.
  apply (bet_conga__bet S T U).
    apply (per2_suma__bet P Q R P Q R); assumption.
  apply (suma2__conga A B C A B D).
    assumption.
  apply suma_left_comm. apply ts__suma. apply HTS.
Qed.
```

```
1 goal (ID 248)

- Tn : Tarski_neutral_dimensionless
- TnEQD : Tarski_neutral_dimensionless_with_decidable_point_equality Tn
- A, B, C, D, P, Q, R, S, T, U : Tpoint
- HTS : TS A B C D
- HP : Per P Q R
- HSuma1 : SumA A B C A B D S T U
- HSuma2 : SumA P Q R P Q R S T U
  ============================
  Bet C B D
```

GeoCoq, Narboux et al.

Proofs through the ages

```
Lemma prop_14 : forall A B C D P Q R S T U,
    TS A B C D ->
    Per P Q R ->
    SumA A B C A B D S T U ->
    SumA P Q R P Q R S T U ->
    Bet C B D.
Proof.
  intros A B C D P Q R S T U HTS HP HSuma1 HSuma2.
  apply (bet_conga__bet S T U).
    apply (per2_suma__bet P Q R P Q R); assumption.
  apply (suma2__conga A B C A B D).
    assumption.
  apply suma_left_comm. apply ts__suma. apply HTS.
Qed.
```

```
2 goals (ID 251)

- Tn : Tarski_neutral_dimensionless
- TnEQD : Tarski_neutral_dimensionless_with_decidable_point_equality Tn
- A, B, C, D, P, Q, R, S, T, U : Tpoint
- HTS : TS A B C D
- HP : Per P Q R
- HSuma1 : SumA A B C A B D S T U
- HSuma2 : SumA P Q R P Q R S T U
  ============================
  Bet S T U

goal 2 (ID 252) is:
 CongA S T U C B D
```

GeoCoq, Narboux et al.

Proofs through the ages

```
Lemma prop_14 : forall A B C D P Q R S T U,
    TS A B C D ->
    Per P Q R ->
    SumA A B C A B D S T U ->
    SumA P Q R P Q R S T U ->
    Bet C B D.
Proof.
  intros A B C D P Q R S T U HTS HP HSuma1 HSuma2.
  apply (bet_conga__bet S T U).
    apply (per2_suma__bet P Q R P Q R); assumption.
  apply (suma2__conga A B C A B D).
    assumption.
  apply suma_left_comm. apply ts__suma. apply HTS.
Qed.
```

```
1 goal (ID 252)

- Tn : Tarski_neutral_dimensionless
- TnEQD : Tarski_neutral_dimensionless_with_decidable_point_equality Tn
- A, B, C, D, P, Q, R, S, T, U : Tpoint
- HTS : TS A B C D
- HP : Per P Q R
- HSuma1 : SumA A B C A B D S T U
- HSuma2 : SumA P Q R P Q R S T U
  ============================
  CongA S T U C B D
```

GeoCoq, Narboux et al.

Proofs through the ages

```
Lemma prop_14 : forall A B C D P Q R S T U,
    TS A B C D ->
    Per P Q R ->
    SumA A B C A B D S T U ->
    SumA P Q R P Q R S T U ->
    Bet C B D.
Proof.
  intros A B C D P Q R S T U HTS HP HSuma1 HSuma2.
  apply (bet_conga__bet S T U).
    apply (per2_suma__bet P Q R P Q R); assumption.
  apply (suma2__conga A B C A B D).
    assumption.
  apply suma_left_comm. apply ts__suma. apply HTS.
Qed.
```

```
2 goals (ID 260)

- Tn : Tarski_neutral_dimensionless
- TnEQD : Tarski_neutral_dimensionless_with_decidable_point_equality Tn
- A, B, C, D, P, Q, R, S, T, U : Tpoint
- HTS : TS A B C D
- HP : Per P Q R
- HSuma1 : SumA A B C A B D S T U
- HSuma2 : SumA P Q R P Q R S T U
  ============================
  SumA A B C A B D S T U

goal 2 (ID 261) is:
 SumA A B C A B D C B D
```

GeoCoq, Narboux et al.

Proofs through the ages

```
Lemma prop_14 : forall A B C D P Q R S T U,
    TS A B C D ->
    Per P Q R ->
    SumA A B C A B D S T U ->
    SumA P Q R P Q R S T U ->
    Bet C B D.
Proof.
  intros A B C D P Q R S T U HTS HP HSuma1 HSuma2.
  apply (bet_conga__bet S T U).
    apply (per2_suma__bet P Q R P Q R); assumption.
  apply (suma2__conga A B C A B D).
    assumption.
 apply suma_left_comm. apply ts__suma. apply HTS.
Qed.
```

```
1 goal (ID 261)

- Tn : Tarski_neutral_dimensionless
- TnEQD : Tarski_neutral_dimensionless_with_decidable_point_equality Tn
- A, B, C, D, P, Q, R, S, T, U : Tpoint
- HTS : TS A B C D
- HP : Per P Q R
- HSuma1 : SumA A B C A B D S T U
- HSuma2 : SumA P Q R P Q R S T U
  ============================
  SumA A B C A B D C B D
```

GeoCoq, Narboux et al.

Proofs through the ages

```
Lemma prop_14 : forall A B C D P Q R S T U,
    TS A B C D ->
    Per P Q R ->
    SumA A B C A B D S T U ->
    SumA P Q R P Q R S T U ->
    Bet C B D.
Proof.
  intros A B C D P Q R S T U HTS HP HSuma1 HSuma2.
  apply (bet_conga__bet S T U).
    apply (per2_suma__bet P Q R P Q R); assumption.
  apply (suma2__conga A B C A B D).
    assumption.
  apply suma_left_comm. apply ts__suma. apply HTS.
Qed.
```

```
1 goal (ID 264)

- Tn : Tarski_neutral_dimensionless
- TnEQD : Tarski_neutral_dimensionless_with_decidable_point_equality Tn
- A, B, C, D, P, Q, R, S, T, U : Tpoint
- HTS : TS A B C D
- HP : Per P Q R
- HSuma1 : SumA A B C A B D S T U
- HSuma2 : SumA P Q R P Q R S T U
  ============================
  SumA C B A A B D C B D
```

GeoCoq, Narboux et al.

Proofs through the ages

```
Lemma prop_14 : forall A B C D P Q R S T U,
    TS A B C D ->
    Per P Q R ->
    SumA A B C A B D S T U ->
    SumA P Q R P Q R S T U ->
    Bet C B D.
Proof.
  intros A B C D P Q R S T U HTS HP HSuma1 HSuma2.
  apply (bet_conga__bet S T U).
    apply (per2_suma__bet P Q R P Q R); assumption.
  apply (suma2__conga A B C A B D).
    assumption.
  apply suma_left_comm. apply ts__suma. apply HTS.
Qed.
```

```
1 goal (ID 267)

- Tn : Tarski_neutral_dimensionless
- TnEQD : Tarski_neutral_dimensionless_with_decidable_point_equality Tn
- A, B, C, D, P, Q, R, S, T, U : Tpoint
- HTS : TS A B C D
- HP : Per P Q R
- HSuma1 : SumA A B C A B D S T U
- HSuma2 : SumA P Q R P Q R S T U
  ============================
  TS A B C D
```

GeoCoq, Narboux et al.

Proofs through the ages

```
Lemma concat_spec : forall A `{Enc A}
                           (p1 p2:loc) (L1 L2:list A),
  SPEC (concat p1 p2)
    PRE (p1 ~> Stack L1 \* p2 ~> Stack L2)
    POST (fun (u:unit) => p1 ~> Stack (L1 ++ L2)
                           \* p2 ~> Stack (@nil A)).
Proof using.
  xcf. xunfold Stack. xapp. xapp. xapp.
  rewrite <- (Stack_eq p2). xapp. xsimpl.
Qed.
```

```
1 goal (ID 127)

  ============================
  forall (A : Type) (H : Enc A) p1 p2 (L1 L2 : list A),
  Triple
    (Trm_apps concat
       (cons {| dyn_type := loc; dyn_enc := Enc_loc; dyn_value := p1 |}
          (cons {| dyn_type := loc; dyn_enc := Enc_loc; dyn_value := p2 |}
             nil))) (hstar (repr (Stack L1) p1) (repr (Stack L2) p2))
    (fun _ : unit =>
     hstar (repr (Stack (app L1 L2)) p1) (repr (Stack nil) p2))
```

Proofs through the ages

```
Lemma concat_spec : forall A `{Enc A}
                           (p1 p2:loc) (L1 L2:list A),
  SPEC (concat p1 p2)
    PRE (p1 ~> Stack L1 \* p2 ~> Stack L2)
    POST (fun (u:unit) => p1 ~> Stack (L1 ++ L2)
                           \* p2 ~> Stack (@nil A)).
Proof using.
  xcf. xunfold Stack. xapp. xapp. xapp.
  rewrite <- (Stack_eq p2). xapp. xsimpl.
Qed.
```

```
1 focused goal
(shelved: 2) (ID 154)

- A : Type
- H : Enc A
- p1, p2 : loc
- L1, L2 : list A
  ============================
  himpl (hstar (repr (Stack L1) p1) (repr (Stack L2) p2))
    (Wptag
       (Wpgen_let_trm
          (Wptag
             (Wpgen_app (list ?A_) infix_emark__
                (cons {| dyn_type := ref_ (list ?A_); dyn_enc := Enc_loc;
dyn_value := p2 |} nil)))
          (fun x1__ : list ?A_ =>
           Wptag
             (Wpgen_let_trm
                (Wptag
                   (Wpgen_app (list ?A_) infix_emark__
                      (cons {| dyn_type := ref_ (list ?A_); dyn_enc := Enc
_loc; dyn_value := p1 |} nil)))
                (fun x0__ : list ?A_ =>
                 Wptag
                   (Wpgen_seq
                      (Wptag
                         (Wpgen_app unit infix_colon_eq__
                            (cons {| dyn_type := ref_ (list ?A_); dyn_enc
:= Enc_loc; dyn_value := p1 |}
                               (cons {| dyn_type := list ?A_; dyn_enc := E
nc_list; dyn_value := app x0__ x1__ |} nil))))
                      (Wptag
                         (Wpgen_app unit clear
                            (cons {| dyn_type := ref_ (list ?A_); dyn_enc
:= Enc_loc; dyn_value := p2 |} nil)))))))
          Enc_unit (fun _ : unit => hstar (hstar (repr (Stack (app L1 L2)) p1
) (repr (Stack nil) p2)) hgc))
```

Proofs through the ages

```
Lemma concat_spec : forall A `{Enc A}
                           (p1 p2:loc) (L1 L2:list A),
  SPEC (concat p1 p2)
    PRE (p1 ~> Stack L1 \* p2 ~> Stack L2)
    POST (fun (u:unit) => p1 ~> Stack (L1 ++ L2)
                           \* p2 ~> Stack (@nil A)).
Proof using.
  xcf. xunfold Stack. xapp. xapp. xapp.
  rewrite <- (Stack_eq p2). xapp. xsimpl.
Qed.
```

```
1 focused goal
(shelved: 2) (ID 168)

- A : Type
- H : Enc A
- p1, p2 : loc
- L1, L2 : list A
  ============================
  himpl (hstar (repr (Ref L1) p1) (repr (Ref L2) p2))
    (Wptag
       (Wpgen_let_trm
          (Wptag
             (Wpgen_app (list ?A_) infix_emark__
                (cons {| dyn_type := ref_ (list ?A_); dyn_enc := Enc_loc;
*dyn_value := p2 |} nil)))
          (fun x1__ : list ?A_ =>
           Wptag
             (Wpgen_let_trm
                (Wptag
                   (Wpgen_app (list ?A_) infix_emark__
                      (cons {| dyn_type := ref_ (list ?A_); dyn_enc := Enc
*_loc; dyn_value := p1 |} nil)))
                (fun x0__ : list ?A_ =>
                 Wptag
                   (Wpgen_seq
                      (Wptag
                         (Wpgen_app unit infix_colon_eq__
                            (cons {| dyn_type := ref_ (list ?A_); dyn_enc
*:= Enc_loc; dyn_value := p1 |}
                               (cons {| dyn_type := list ?A_; dyn_enc := E
*nc_list; dyn_value := app x0__ x1__ |} nil))))
                      (Wptag
                         (Wpgen_app unit clear
                            (cons {| dyn_type := ref_ (list ?A_); dyn_enc
*:= Enc_loc; dyn_value := p2 |} nil))))))))
       Enc_unit (fun _ : unit => hstar (hstar (repr (Ref (app L1 L2)) p1)
*(repr (Ref nil) p2)) hgc))
```

Proofs through the ages

```
Lemma concat_spec : forall A `{Enc A}
                          (p1 p2:loc) (L1 L2:list A),
  SPEC (concat p1 p2)
    PRE (p1 ~> Stack L1 \* p2 ~> Stack L2)
    POST (fun (u:unit) => p1 ~> Stack (L1 ++ L2)
                           \* p2 ~> Stack (@nil A)).
Proof using.
  xcf. xunfold Stack. xapp. xapp. xapp.
  rewrite <- (Stack_eq p2). xapp. xsimpl.
Qed.
```

```
1 goal (ID 2077)

- A : Type
- H : Enc A
- p1, p2 : loc
- L1, L2 : list A
  ============================
  PRE (p1 ~~> (L1 ++ L2) \* p2 ~~> L2)
  CODE (Wpgen_app unit clear ((Dyn p2) :: nil))
  POST (fun _ : unit => (p1 ~~> (L1 ++ L2) \* p2 ~~> nil) \* \GC)
```

cfml2, Charguéraud

Proofs through the ages

```
Lemma concat_spec : forall A `{Enc A}
                          (p1 p2:loc) (L1 L2:list A),
  SPEC (concat p1 p2)
    PRE (p1 ~> Stack L1 \* p2 ~> Stack L2)
    POST (fun (u:unit) => p1 ~> Stack (L1 ++ L2)
                           \* p2 ~> Stack (@nil A)).
Proof using.
  xcf. xunfold Stack. xapp. xapp. xapp.
  rewrite <- (Stack_eq p2). xapp. xsimpl.
Qed.
```

```
1 goal (ID 2078)

- A : Type
- H : Enc A
- p1, p2 : loc
- L1, L2 : list A
  ============================
  PRE (p1 ~~> (L1 ++ L2) \* p2 ~> Stack L2)
  CODE (Wpgen_app unit clear ((Dyn p2) :: nil))
  POST (fun _ : unit => (p1 ~~> (L1 ++ L2) \* p2 ~~> nil) \* \GC)
```

cfml2, Charguéraud

Proofs through the ages

```
Lemma concat_spec : forall A `{Enc A}
                          (p1 p2:loc) (L1 L2:list A),
  SPEC (concat p1 p2)
    PRE (p1 ~> Stack L1 \* p2 ~> Stack L2)
    POST (fun (u:unit) => p1 ~> Stack (L1 ++ L2)
                           \* p2 ~> Stack (@nil A)).
Proof using.
  xcf. xunfold Stack. xapp. xapp. xapp.
  rewrite <- (Stack_eq p2). xapp. xsimpl.
Qed.
```

```
1 goal (ID 3173)

- A : Type
- H : Enc A
- p1, p2 : loc
- L1, L2 : list A
  ============================
  p2 ~> Stack nil \* p1 ~~> (L1 ++ L2) ==> (p1 ~~> (L1 ++ L2) \* p2 ~~> nil) \* \GC
```

cfml2, Charguéraud

Proofs through the ages

```
Lemma concat_spec : forall A `{Enc A}
                          (p1 p2:loc) (L1 L2:list A),
  SPEC (concat p1 p2)
    PRE (p1 ~> Stack L1 \* p2 ~> Stack L2)
    POST (fun (u:unit) => p1 ~> Stack (L1 ++ L2)
                          \* p2 ~> Stack (@nil A)).
Proof using.
  xcf. xunfold Stack. xapp. xapp. xapp.
  rewrite <- (Stack_eq p2). xapp. xsimpl.
Qed.
```

```
1 goal (ID 3173)

- A : Type
- H : Enc A
- p1, p2 : loc
- L1, L2 : list A
  ============================
  p2 ~> Stack nil \* p1 ~~> (L1 ++ L2) ==> (p1 ~~> (L1 ++ L2) \* p2 ~~> nil) \* \GC
```

$$(f_2 \mapsto \langle d_1, c_2\rangle \quad \star\, p_1 \mapsto \langle f_1, b_2\rangle \quad \star\, p_2 \mapsto \langle f_2, b_2\rangle$$
$$\star\, c_2 \mapsto \langle\langle b_2, L_2'\rangle\rangle \quad \star\, b_2 \mapsto \langle d_2, \mathrm{null}\rangle \quad \star\, f_1 \mapsto \langle\langle b_1, L_1\rangle\rangle)$$

$$\exists L_1 L_2 q. \quad [L = L_1 +\!\!+ L_2] \star (t \mapsto \mathsf{length}\, L_1) \star (f \mapsto q)$$
$$\star\, (p \rightsquigarrow \mathsf{MlistSeg}\, q\, L_1) \star (q \rightsquigarrow \mathsf{Mlist}\, L_2)$$

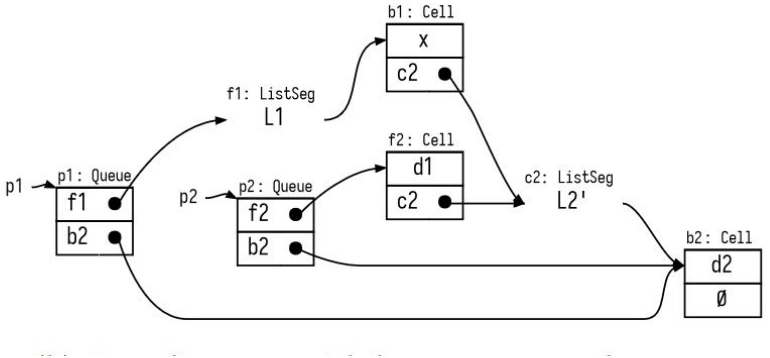
Proofs through the ages

```
Lemma concat_spec : forall A `{Enc A}
                      (p1 p2:loc) (L1 L2:list A),
  SPEC (concat p1 p2)
    PRE (p1 ~> Stack L1 \* p2 ~> Stack L2)
    POST (fun (u:unit) => p1 ~> Stack (L1 ++ L2)
                          \* p2 ~> Stack (@nil A)).
Proof using.
  xcf. xunfold Stack. xapp. xapp. xapp.
  rewrite <- (Stack_eq p2). xapp. xsimpl.
Qed.
```

```
1 goal (ID 3173)

- A : Type
- H : Enc A
- p1, p2 : loc
- L1, L2 : list A
  ============================
  p2 ~> Stack nil \* p1 ~~> (L1 ++ L2) ==> (p1 ~~> (L1 ++ L2) \* p2 ~~> nil) \* \GC
```

$$(f_2 \mapsto \langle d_1, c_2\rangle \quad \star\, p_1 \mapsto \langle f_1, b_2\rangle \quad \star\, p_2 \mapsto \langle f_2, b_2\rangle$$
$$\star\, c_2 \mapsto \langle\langle b_2, L_2'\rangle\rangle \quad \star\, b_2 \mapsto \langle d_2, \mathrm{null}\rangle \quad \star\, f_1 \mapsto \langle\langle b_1, L_1\rangle\rangle)$$

$$\exists L_1 L_2 q. \quad [L = L_1 +\!\!+ L_2] \star (t \mapsto \mathsf{length}\, L_1) \star (f \mapsto q)$$
$$\star\, (p \rightsquigarrow \mathsf{MlistSeg}\, q\, L_1) \star (q \rightsquigarrow \mathsf{Mlist}\, L_2)$$



Plan for this talk

- We need **diagrammatic notations** in computer-assisted proofs, but they are mostly missing.

Plan for this talk

- We need **diagrammatic notations** in computer-assisted proofs, but they are mostly missing. Why?

Plan for this talk

- We need **diagrammatic notations** in computer-assisted proofs, but they are mostly missing. Why?
- Let's learn from a success story: **text notations**.

Plan for this talk

- We need **diagrammatic notations** in computer-assisted proofs, but they are mostly missing. Why?
- Let's learn from a success story: **text notations**.
  → Identify **five principles**.

Plan for this talk

- We need **diagrammatic notations** in computer-assisted proofs, but they are mostly missing. Why?
- Let's learn from a success story: **text notations**.
  → Identify **five principles**.
- What might these look like for diagrams?

```
(sep (MCell f2 d1 c2) (sep (MCell p1
  f1 b2) (sep (MCell p2 f2 b2) (sep
    (MCell b1 x c2) (sep (MListSeg c2 b2
      L2') (sep (MCell b2 d2 null)
        (MListSeg f1 b1 L1)))))))
```

```
    (sep (MCell f2 d1 c2) (sep (MCell p1
      f1 b2) (sep (MCell p2 f2 b2) (sep
        (MCell b1 x c2) (sep (MListSeg c2 b2
          L2') (sep (MCell b2 d2 null)
            (MListSeg f1 b1 L1)))))))

Notation "p1 '\*' p2" := sep p1 p2.
```

```
(sep (MCell f2 d1 c2) (sep (MCell p1
  f1 b2) (sep (MCell p2 f2 b2) (sep
    (MCell b1 x c2) (sep (MListSeg c2 b2
      L2') (sep (MCell b2 d2 null)
        (MListSeg f1 b1 L1)))))))
```

```
Notation "p1 '\*' p2" := sep p1 p2.
```

```
((MCell f2 d1 c2) \* ((MCell p1 f1 b2)
  \* ((MCell p2 f2 b2) \* ((MCell b1 x c2)
  \* ((MListSeg c2 b2 L2') \* ((MCell b2 d2 null)
  \* (MListSeg f1 b1 L1)))))))
```

```
   (sep (MCell f2 d1 c2) (sep (MCell p1
     f1 b2) (sep (MCell p2 f2 b2) (sep
       (MCell b1 x c2) (sep (MListSeg c2 b2
         L2') (sep (MCell b2 d2 null)
           (MListSeg f1 b1 L1)))))))

Notation "p1 '\*' p2" := sep p1 p2.

   ((MCell f2 d1 c2) \* ((MCell p1 f1 b2)
     \* ((MCell p2 f2 b2) \* ((MCell b1 x c2)
     \* ((MListSeg c2 b2 L2') \* ((MCell b2 d2 null)
     \* (MListSeg f1 b1 L1)))))))

Notation "p '~>'  '<' hd  ','  tl  '>'"  := MCell p hd tl.
Notation "p '~>' '<<' pre ',' rest '>>'" := MListSeg p pre rest.
```

```
(sep (MCell f2 d1 c2) (sep (MCell p1
  f1 b2) (sep (MCell p2 f2 b2) (sep
    (MCell b1 x c2) (sep (MListSeg c2 b2
      L2') (sep (MCell b2 d2 null)
        (MListSeg f1 b1 L1)))))))

Notation "p1 '\*' p2" := sep p1 p2.

((MCell f2 d1 c2) \* ((MCell p1 f1 b2)
  \* ((MCell p2 f2 b2) \* ((MCell b1 x c2)
  \* ((MListSeg c2 b2 L2') \* ((MCell b2 d2 null)
  \* (MListSeg f1 b1 L1)))))))

Notation "p '~>'  '<' hd  ','  tl  '>'"  := MCell p hd tl.
Notation "p '~>' '<<' pre ',' rest '>>'" := MListSeg p pre rest.

((f2 ~> <d1, c2>) \* ((p1 ~> <f1, b2>)
  \* ((p2 ~> <f2, b2>) \* ((b1 ~> <x, c2>)
  \* ((c2 ~> <<b2, L2'>>) \* ((b2 ~> <d2, null>)
  \* (f1 ~> <<b1, L1>>)))))))
```

```
  (sep (MCell f2 d1 c2) (sep (MCell p1
    f1 b2) (sep (MCell p2 f2 b2) (sep
      (MCell b1 x c2) (sep (MListSeg c2 b2
        L2') (sep (MCell b2 d2 null)
          (MListSeg f1 b1 L1)))))))

Notation "p1 '\*' p2" := sep p1 p2.

  ((M
```

Low cost, high reward

```
Notation "p '~>'  '<' hd  ','  tl  '>'"  := MCell p hd tl.
Notation "p '~>' '<<' pre ',' rest '>>'" := MListSeg p pre rest.

  ((f2 ~> <d1, c2>) \* ((p1 ~> <f1, b2>)
    \* ((p2 ~> <f2, b2>) \* ((b1 ~> <x, c2>)
    \* ((c2 ~> <<b2, L2'>>) \* ((b2 ~> <d2, null>)
    \* (f1 ~> <<b1, L1>>)))))))
```

```
Notation "p1 '\*' p2" := sep p1 p2.
Notation "p '~>'  '<' hd  ','  tl  '>'"  := MCell p hd tl.
Notation "p '~>' '<<' pre ',' rest '>>'" := MListSeg p pre rest.
```

```
(sep (MCell f2 d1 c2) (sep (MCell p1
  f1 b2) (sep (MCell p2 f2 b2) (sep
    (MCell b1 x c2) (sep (MListSeg c2 b2
      L2') (sep (MCell b2 d2 null)
        (MListSeg f1 b1 L1)))))))
```

```
((f2 ~> <d1, c2>) \* ((p1 ~> <f1, b2>)
  \* ((p2 ~> <f2, b2>) \* ((b1 ~> <x, c2>)
  \* ((c2 ~> <<b2, L2'>>) \* ((b2 ~> <d2, null>)
  \* (f1 ~> <<b1, L1>>)))))))
```

```
Notation "p1 '\*' p2" := sep p1 p2.
Notation "p '~>'  '<' hd  ','  tl  '>'"  := MCell p hd tl.
Notation "p '~>' '<<' pre ',' rest '>>'" := MListSeg p pre rest.
```

```
(sep (MCell p1 h1 t1) (MCell p2 h2 t2))
```

```
((p1 ~> <h1, t1>) \* (p2 ~> <h2, t2>))
```

```
Notation "p1 '\*' p2" := sep p1 p2.
Notation "p '~>'  '<' hd  ','  tl  '>'"  := MCell p hd tl.
Notation "p '~>' '<<' pre ',' rest '>>'" := MListSeg p pre rest.
```

```
(sep (MCell p1 h1 t1) (MCell p2 h2 Lprimes))
```

```
((p1 ~> <h1, t1>) \* (p2 ~> <h2, [2, 3, 5, 7]>))
```

```
Notation "p1 '\*' p2" := sep p1 p2.
Notation "p '~>'  '<' hd  ','  tl  '>'"  := MCell p hd tl.
Notation "p '~>' '<<' pre ',' rest '>>'" := MListSeg p pre rest.
```

```
(sep (MCell p1 h1 t1) (MCell p2 h2 Lprimes))
```

Structure-driven composition

```
((p1 ~> <h1, t1>) \* (p2 ~> <h2, [2, 3, 5, 7]>))
```

```
Notation "p1 '\*' p2" := sep p1 p2.

Notation "p '~>'  '<' hd  ','  tl  '>'"  := MCell p hd tl.

Notation "p '~>' '<<' pre ',' rest '>>'" := MListSeg p pre rest.
```

```
((f2 ~> <d1, c2>) \* ((p1 ~> <f1, b2>)
  \* ((p2 ~> <f2, b2>) \* ((b1 ~> <x, c2>)
  \* ((c2 ~> <<b2, L2'>>) \* ((b2 ~> <d2, null>)
  \* (f1 ~> <<b1, L1>>)))))))
```

```
Notation "p1 '\*' p2" := sep p1 p2
  (right associativity).

Notation "p '~>'  '<' hd  ','  tl  '>'"  := MCell p hd tl.

Notation "p '~>' '<<' pre ',' rest '>>'" := MListSeg p pre rest.
```

```
((f2 ~> <d1, c2>) \* ((p1 ~> <f1, b2>)
  \* ((p2 ~> <f2, b2>) \* ((b1 ~> <x, c2>)
  \* ((c2 ~> <<b2, L2'>>) \* ((b2 ~> <d2, null>)
  \* (f1 ~> <<b1, L1>>)))))))
```

```
Notation "p1 '\*' p2" := sep p1 p2
  (right associativity).

Notation "p '~>'  '<' hd  ','  tl  '>'"  := MCell p hd tl.

Notation "p '~>' '<<' pre ',' rest '>>'" := MListSeg p pre rest.
```

```
(f2 ~> <d1, c2>) \* (p1 ~> <f1, b2>)
  \* (p2 ~> <f2, b2>) \* (b1 ~> <x, c2>)
  \* (c2 ~> <<b2, L2'>>) \* (b2 ~> <d2, null>)
  \* (f1 ~> <<b1, L1>>)
```

```
Notation "p1 '\*' p2" := sep p1 p2
  (right associativity, at level 75).

Notation "p '~>'  '<' hd  ','  tl  '>'"  := MCell p hd tl
  (at level 70).

Notation "p '~>' '<<' pre ',' rest '>>'" := MListSeg p pre rest
  (at level 70).
```

```
(f2 ~> <d1, c2>) \* (p1 ~> <f1, b2>)
  \* (p2 ~> <f2, b2>) \* (b1 ~> <x, c2>)
  \* (c2 ~> <<b2, L2'>>) \* (b2 ~> <d2, null>)
  \* (f1 ~> <<b1, L1>>)
```

```
Notation "p1 '\*' p2" := sep p1 p2
  (right associativity, at level 75).

Notation "p '~>'  '<' hd  ','  tl  '>'"  := MCell p hd tl
  (at level 70).

Notation "p '~>' '<<' pre ',' rest '>>'" := MListSeg p pre rest
  (at level 70).
```

```
f2 ~> <d1, c2> \* p1 ~> <f1, b2>
  \* p2 ~> <f2, b2> \* b1 ~> <x, c2>
  \* c2 ~> <<b2, L2'>> \* b2 ~> <d2, null>
  \* f1 ~> <<b1, L1>>
```

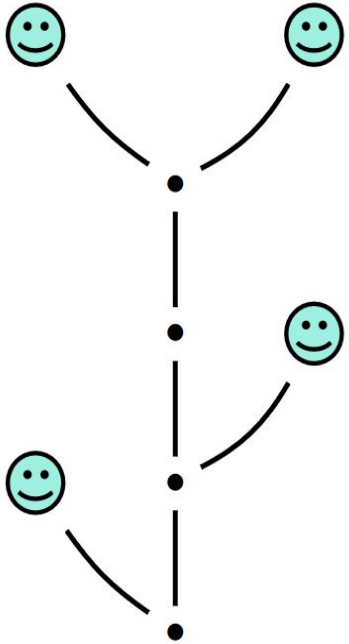
```
Infix "\*" := sep
  (right associativity, at level 75).

Notation "p '~>'  '<' hd  ','  tl  '>'"  := MCell p hd tl
  (at level 70).

Notation "p '~>' '<<' pre ',' rest '>>'" := MListSeg p pre rest
  (at level 70).
```

```
f2 ~> <d1, c2> \* p1 ~> <f1, b2>
  \* p2 ~> <f2, b2> \* b1 ~> <x, c2>
  \* c2 ~> <<b2, L2'>> \* b2 ~> <d2, null>
  \* f1 ~> <<b1, L1>>
```

```
Infix "\*" := sep
  (right associativity, at level 75).

Notation "p '~>'  '<' hd  ','  tl  '>'"  := MCell p hd tl
  (at level 70).

Notation "p '~>' '<<' pre ',' rest '>>'" := MListSeg p pre rest
  (at level
```

Good-looking idioms

```
f2 ~> <d1, c2> \* p1 ~> <f1, b2>
  \* p2 ~> <f2, b2> \* b1 ~> <x, c2>
  \* c2 ~> <<b2, L2'>> \* b2 ~> <d2, null>
  \* f1 ~> <<b1, L1>>
```



a hydra

```
(hyd2
  head
  (hyd2
    (hyd1
      (hyd2 head head))
    head))
```

```
(hyd2
  head
  (hyd2
    (hyd1
      (hyd2 head head))
    head))
```

```
(hyd2
  head
  (hyd2
    (hyd1
      (hyd2 head head))
    head))
```

```
(hyd2
  head
  (hyd2
    (hyd1
      (hyd2 head head))
    head))
```

```
(hyd2
  head
  (hyd2
    (hyd1
      (hyd2 head     ))
    head))
```

```
(hyd2
  head
  (hyd2
    (hyd5
      (hyd1 head)
      (hyd1 head)
      (hyd1 head)
      (hyd1 head)
      (hyd1 head))
    head))
```

```
(hyd2
  head
  (hyd2
    (hyd1
      (hyd2 head head))
    head))
```

```
(hyd2
  head
  (hyd2
    (hyd5
      (hyd1 head)
      (hyd1 head)
      (hyd1 head)
      (hyd1 head)
      (hyd1 head))
    head))
```

```
(hyd2
  head
  (hyd2
      (hyd1 head)
      (hyd1 head)
      (hyd1 head))
    head))
```

Continuity

```
(hyd2
  head
  (hyd2
    (hyd1
      (hyd2 head head))
    head))
```

```
(hyd2
  head
  (hyd2
    (hyd1
      (hyd2 head head))
    head))
```

```
(hyd2
  head
  (hyd2
    head
    (hyd1
      (hyd2 head head))))
```

```
(hyd2
  head
  (hyd2
    (hyd1
      (hyd2 head head))
    head))
```

```
level 0: #0
level 1: 0 -> #1, 0 -> #2
level 2: 2 -> #3, 2 -> #4
level 3: 3 -> #5
level 4: 5 -> #6, 5 -> #7
```

```
(hyd2
  head
  (hyd2
    (hyd1
      (hyd2 head head))
    head))
```

```
level 0: #0
level 1: 0 -> #1, 0 -> #2
level 2: 2 -> #3, 2 -> #4
level 3: 3 -> #5
level 4: 5 -> #6, 5 -> #7
```

```
(hyd2
  head
  (hyd2
    (hyd5
      (hyd1 head)
      (hyd1 head)
      (hyd1 head)
      (hyd1 head)
      (hyd1 head))
    head))
```

```
level 0: #0
level 1: 0 -> #1, 0 -> #2
level 2: 2 -> #3, 2 -> #4
level 3: 3 -> #5, 3 -> #8,
         3 -> #10, 3 -> #12,
         3 -> #14
level 4: 5 -> #6, 8 -> #9,
         10 -> #11,
         12 -> #13,
         14 -> #15
```

```
(hyd2
  head
  (hyd2
    (hyd5
      (hyd1 head)
      (hyd1 head)
      (hyd1 head)
      (hyd1 head)
      (hyd1 head))
    head))
```

```
(hyd2
  head
  (hyd2
    (hyd5
      (hyd1 h
      (hyd1 h
      (hyd1 head)
      (hyd1 head)
      (hyd1 head))
    head))
```

Canonicity

Ideal diagrammatic notation systems

Learn from the success of **text notations**:

Ideal diagrammatic notation systems

Learn from the success of **text notations**:

- Low cost, high reward
- Structure-driven composition
- Continuity
- Canonicity
- Good-looking idioms

Ideal diagrammatic notation systems

Learn from the success of **text notations**:

- Low cost, high reward
- Structure-driven composition
- Continuity
- Canonicity
- Good-looking idioms

What might this look like for **diagrammatic notations**?

We are *not* trying to…

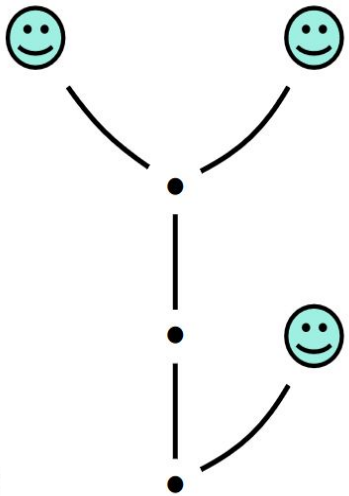
- Describe ‘good’ diagrams
  - Rather, describe good diagramming systems, and illustrate their properties with some concrete diagrams
- Prove our system better than other tools
  - (We don’t have one)
- Prove diagrams better than text notations
  - Rather, computer-assisted proofs should have both!

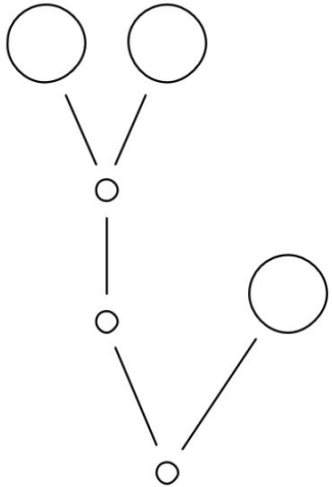
Ideal diagrammatic notation systems

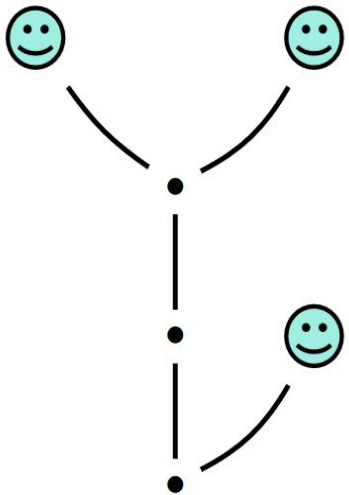
Learn from the success of **text notations**:

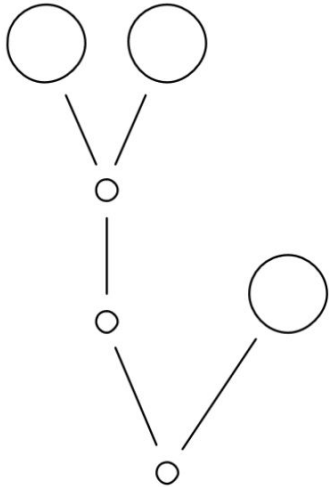
- Low cost, high reward
- Structure-driven composition
- Continuity
- Canonicity
- Good-looking idioms

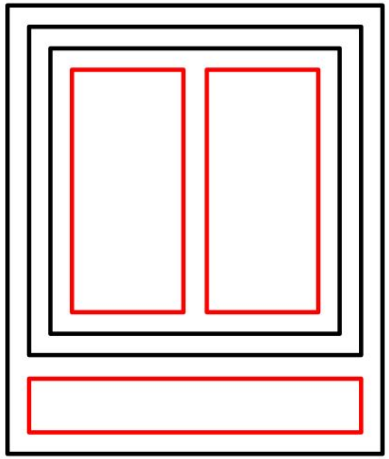
What might this look like for **diagrammatic notations**?











Structure-driven composition

Structure-driven composition

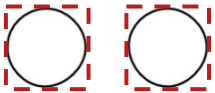
```
(node hnil)
```

Structure-driven composition

```
(node hnil)
```

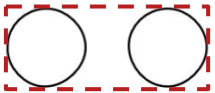
```
diagram (node hnil) := circle
```

Structure-driven composition



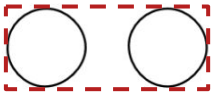
```
(node hnil)
(node hnil)
```

Structure-driven composition



```
(hcons (node hnil)
(hcons (node hnil)
hnil))
```

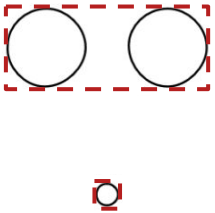
Structure-driven composition



```
(hcons (node hnil)
(hcons (node hnil)
hnil))
```

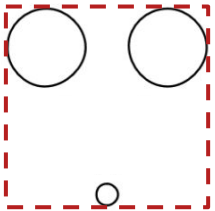
```
diagram (hcons x xs) :=
  happend (diagram x) (diagram xs)
```

Structure-driven composition



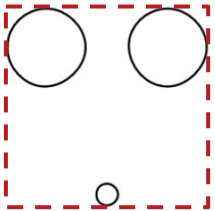
```
node
(hcons (node hnil)
(hcons (node hnil)
hnil))
```

Structure-driven composition



```
      (node
(hcons (node hnil)
(hcons (node hnil)
hnil)))
```

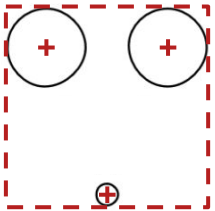
Structure-driven composition



```
      (node
(hcons (node hnil)
(hcons (node hnil)
hnil)))
```

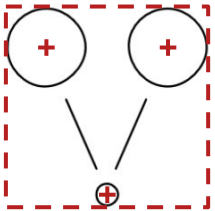
```
diagram (node xs) :=
  vappend circle (diagram xs)
```

Structure-driven composition



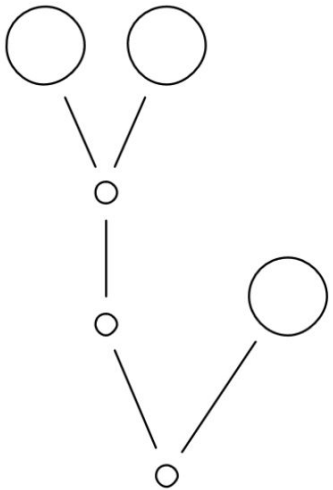
```
(node
(hcons (node hnil)
(hcons (node hnil)
hnil)))
```

Structure-driven composition



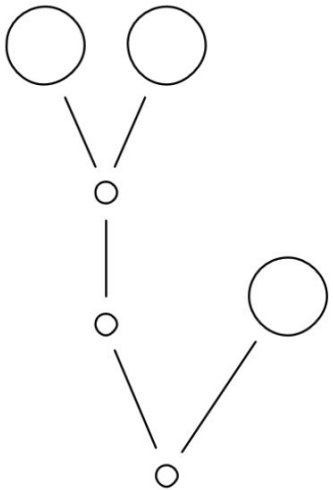
```
      (node
(hcons (node hnil)
(hcons (node hnil)
hnil)))
```

Structure-driven composition



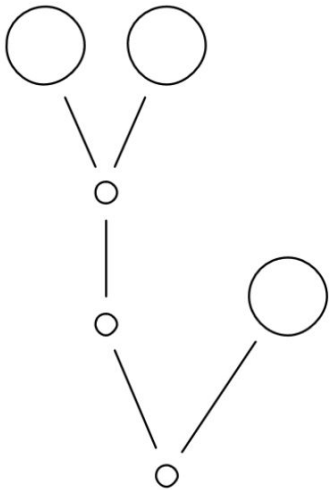
```
diagram (node hnil) := circle

diagram (hcons x xs) :=
  happend (diagram x) (diagram xs)

diagram (node xs) :=
  vappend circle (diagram xs)
```

High reward. Low cost?



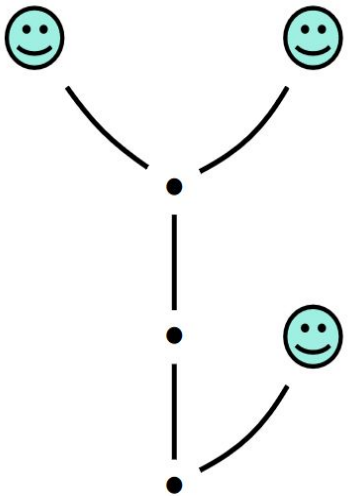
```
diagram (node hnil) := circle

diagram (hcons x xs) :=
  happend (diagram x) (diagram xs)

diagram (node xs) :=
  vappend circle (diagram xs)
```

High reward. Low cost?



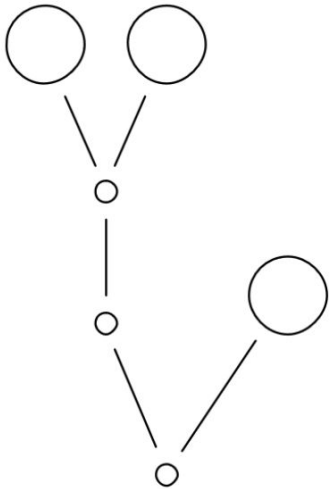
```
diagram (node hnil) := circle

diagram (hcons x xs) :=
  happend (diagram x) (diagram xs)

diagram (node xs) :=
  vappend circle (diagram xs)


Infix "\*" := sep.
```

High reward. Low cost?

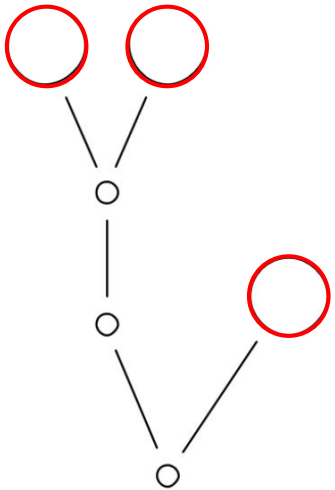


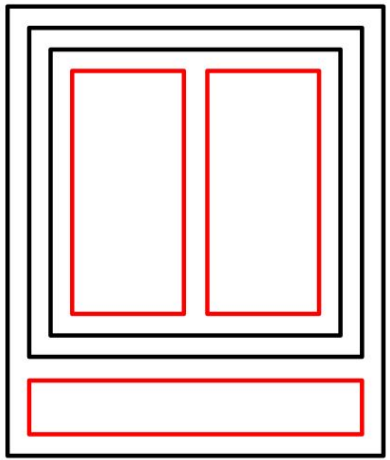
```
\node (N1) at (2,2) {$\bullet$};
\node (N2) at (2,4) {$\bullet$};
\node (N3) at (2,6) {$\bullet$};
\node (H1) at (0,8)
  {$\Smiley[1.5][vertfluo]$};
\node (H2) at (4,8)
  {$\Smiley[1.5][vertfluo]$};
\node (H5) at (4,4)
  {$\Smiley[1.5][vertfluo]$};
\draw (N1) -- (N2);
\draw (N2) -- (N3);
\draw (N3) to [bend left=10]  (H1);
\draw (N3) to [bend right=16] (H2);
\draw (N1) to [bend right=16] (H5);
```

Good-looking idioms

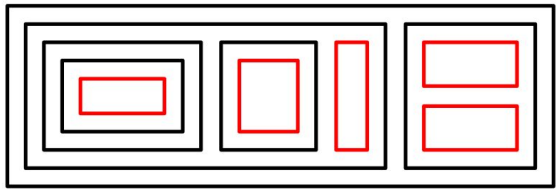


Good-looking idioms

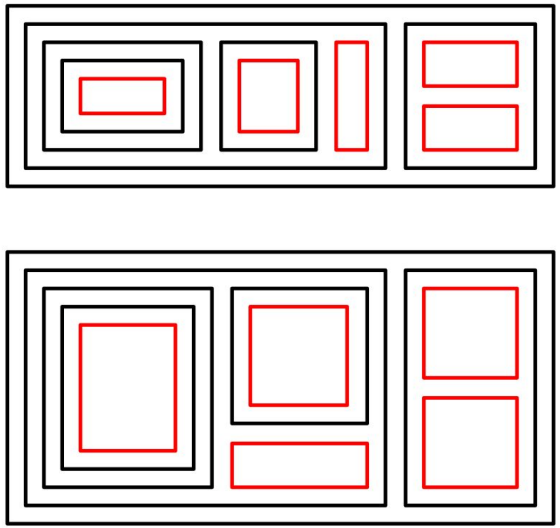




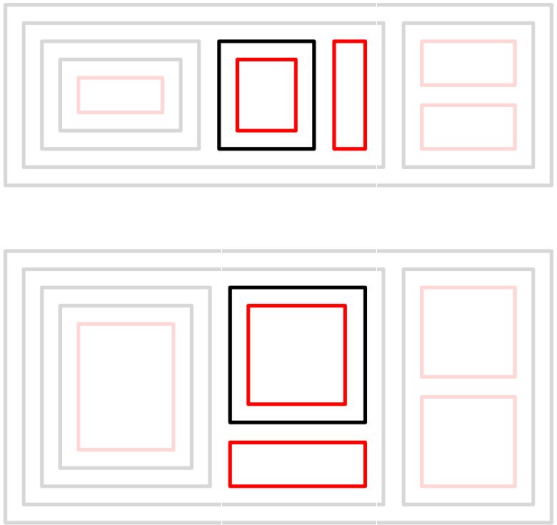
Good-looking idioms



Good-looking idioms

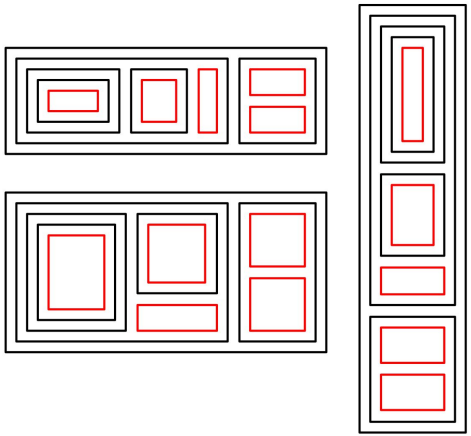


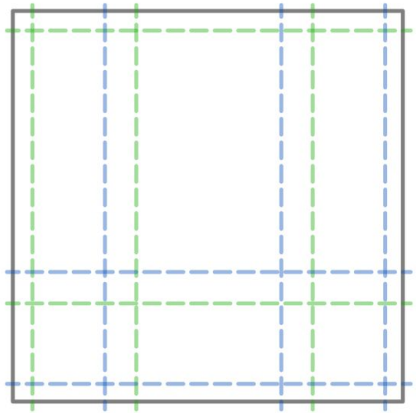
Good-looking idioms



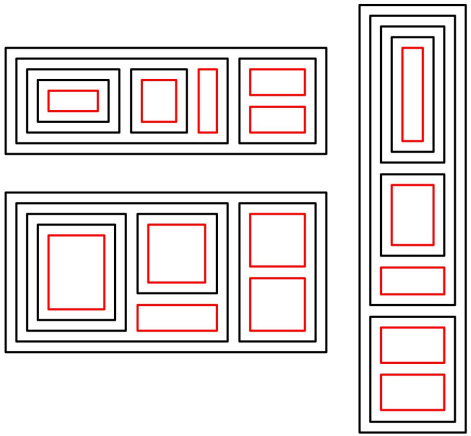
Good-looking idioms

Good-looking idioms

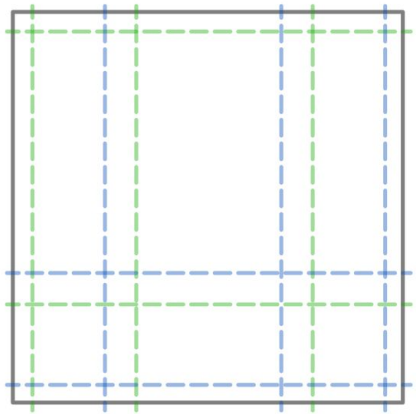




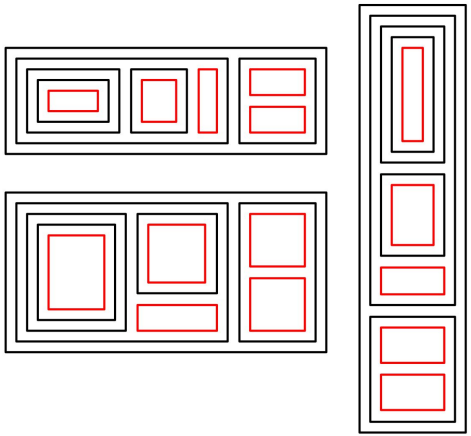
Good-looking idioms



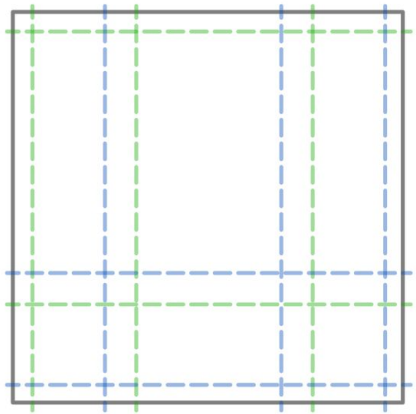
```
(define gl
  (new grid-layout [...]

                                  ))
```



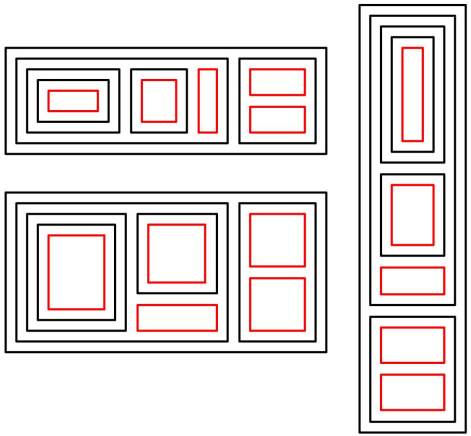
Good-looking idioms



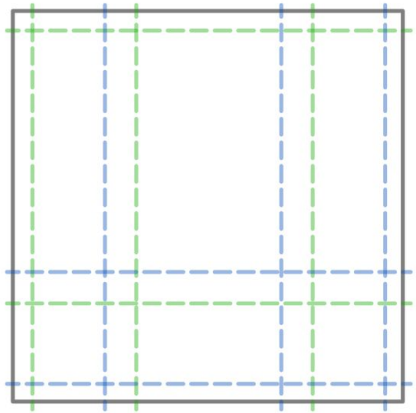
```
(define gl
  (new grid-layout
                ; columns  rows
    [grid-defn '((1 2 1)  (3 1))]
                       ))
```



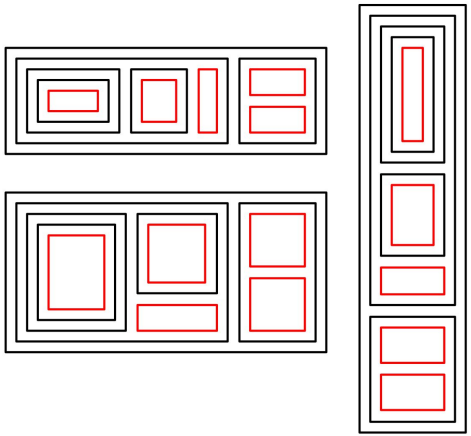
Good-looking idioms



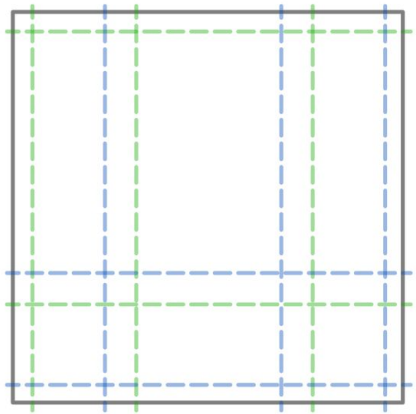
```
(define gl
  (new grid-layout
                ; columns  rows
    [grid-defn '((1 2 1)  (3 1))]
    [gap 16] [padding 10]))
```



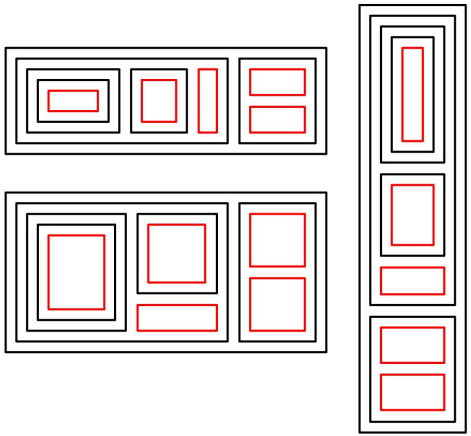
Good-looking idioms



```
(define gl
  (new grid-layout
                ; columns  rows
    [grid-defn '((1 2 1)  (3 1))]
    [gap 16] [padding 10]))


(send gl draw! 200 200)
```

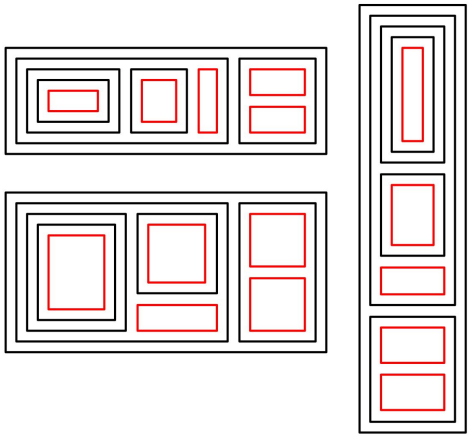


Good-looking idioms

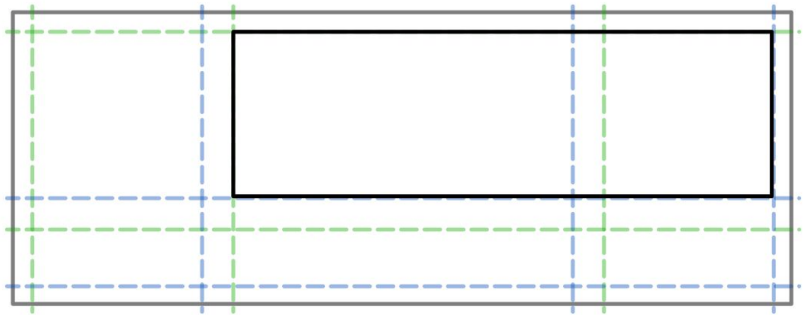


```
(define gl
  (new grid-layout
                ; columns  rows
    [grid-defn '((1 2 1)  (3 1))]
    [gap 16] [padding 10]))


(send gl draw! 400 150)
```

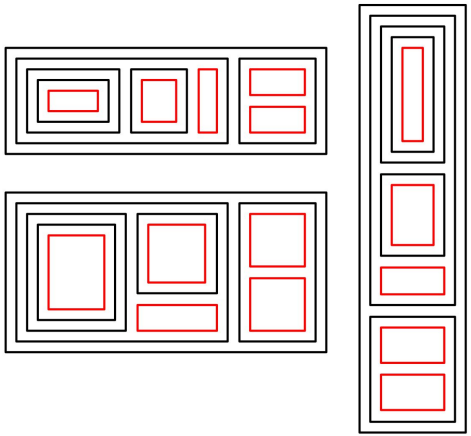
Good-looking idioms



```
(define gl
  (new grid-layout
                ; columns  rows
    [grid-defn '((1 2 1)  (3 1))]
    [gap 16] [padding 10]
    [item-defns `((0 1 1 2 ,box))]))

(send gl draw! 400 150)
```

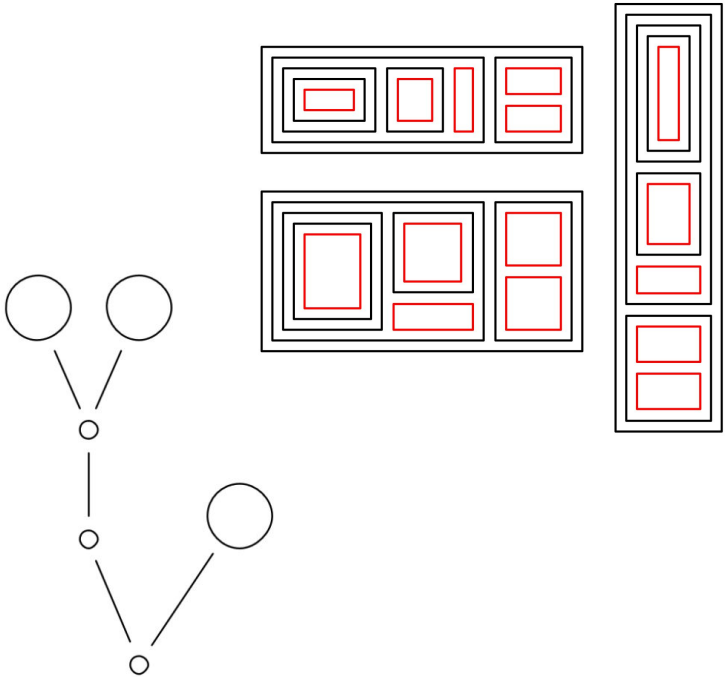


Good-looking idioms

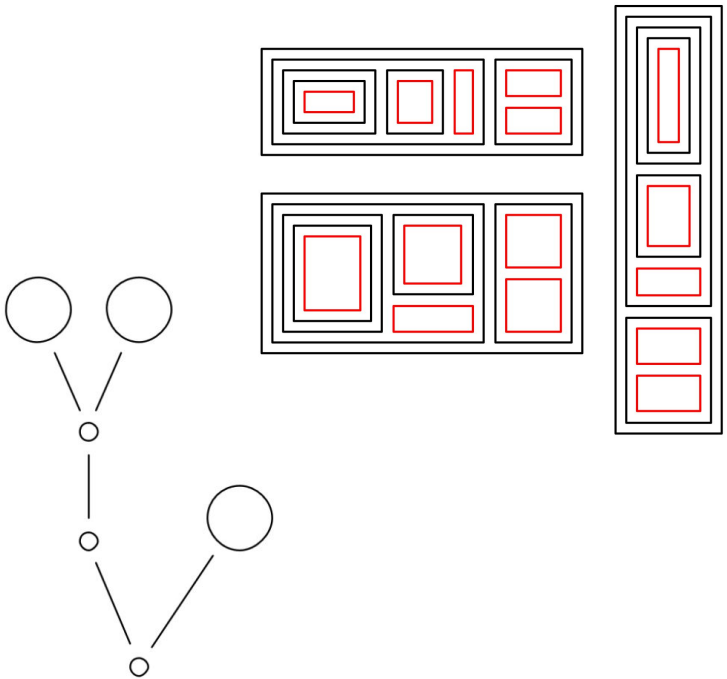


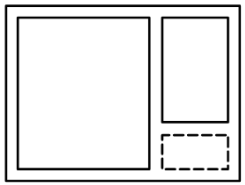
```
(define gl
  (new grid-layout
                ; columns  rows
    [grid-defn '((1 2 1)  (3 1))]
    [gap 16] [padding 10]
    [item-defns `((0 1 1 2 ,box))]))

(send (send gl transpose) draw! 400 150)
```

Good-looking idioms

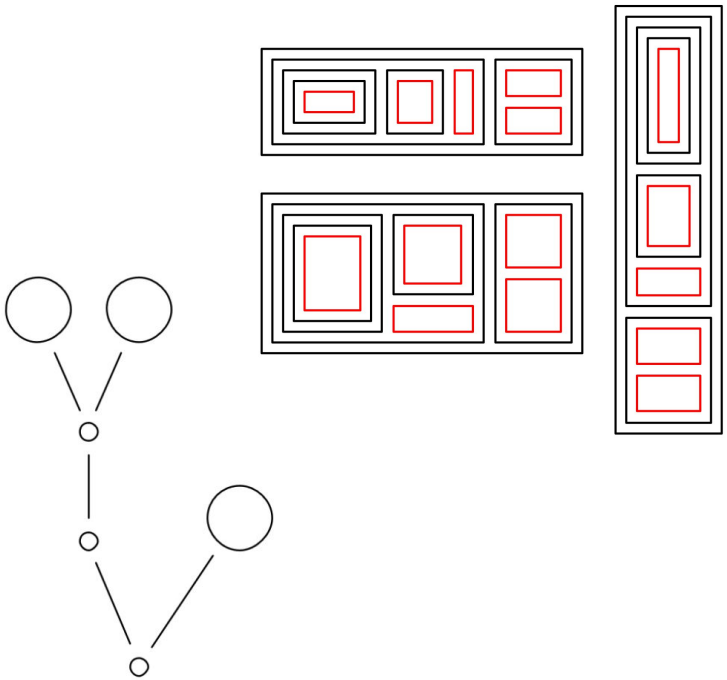


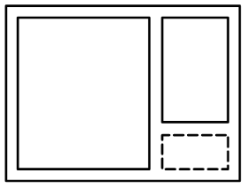
Good-looking idioms



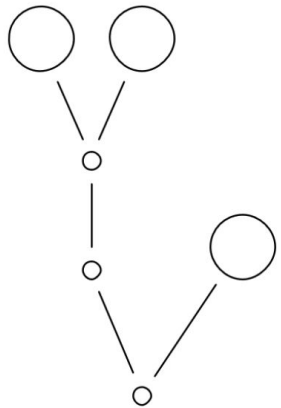
- Grid layouts
  - Transpositions
  - 

Good-looking idioms

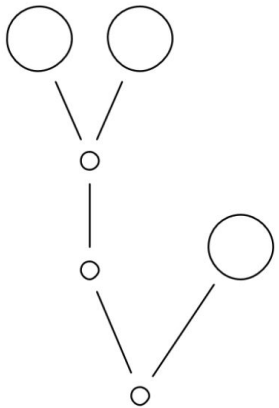


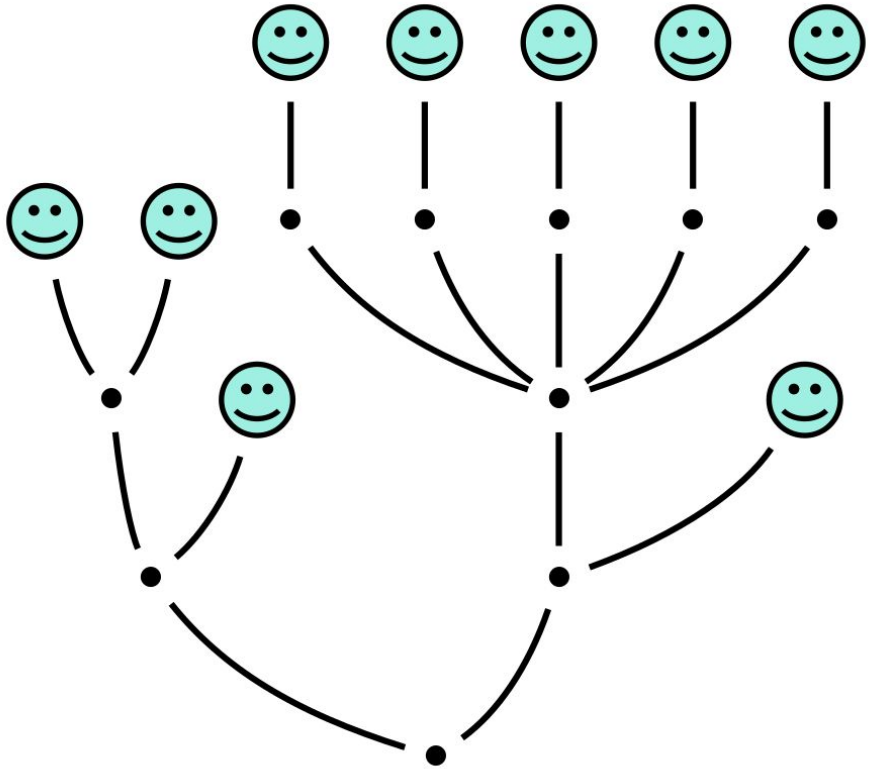
- Grid layouts
  - Transpositions
  - 
- Diagram composition
  - horizontal & vertical juxtaposition
  - bounding boxes
  - endpoints & connectors
  - nesting

Good-looking idioms

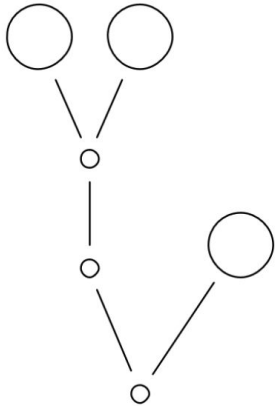


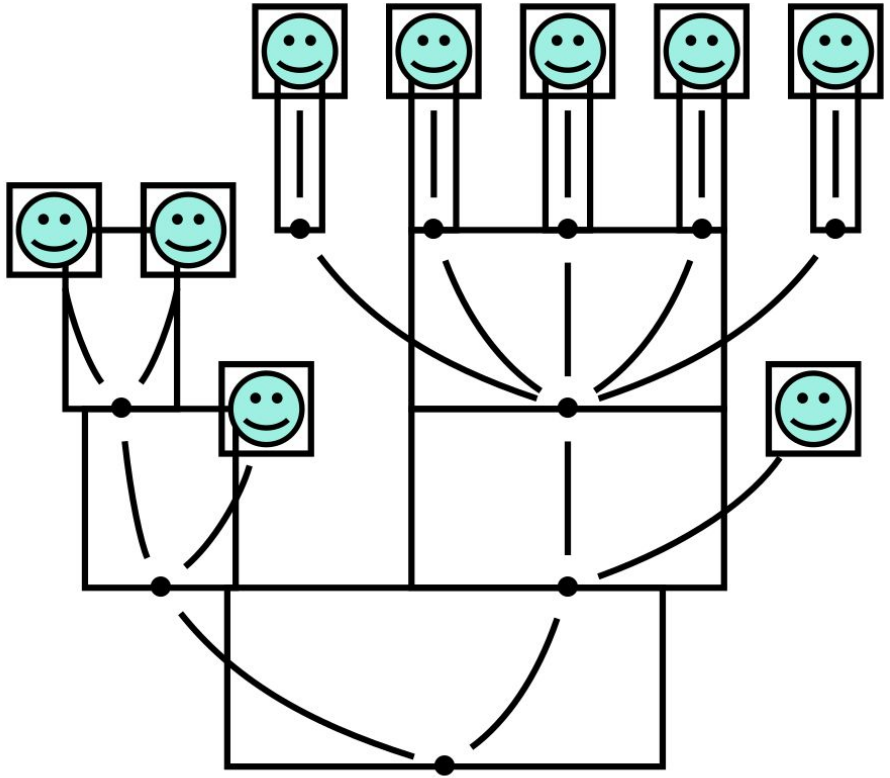
Good-looking idioms



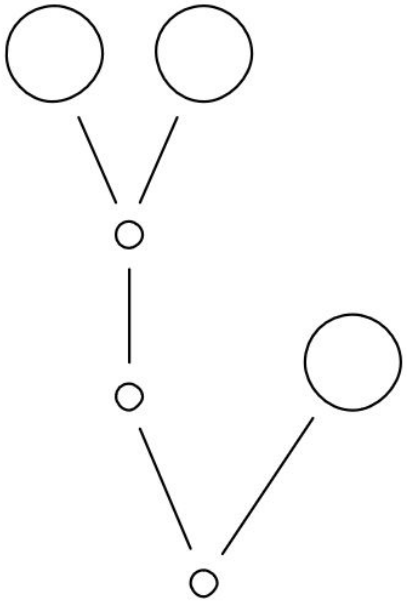


Good-looking idioms

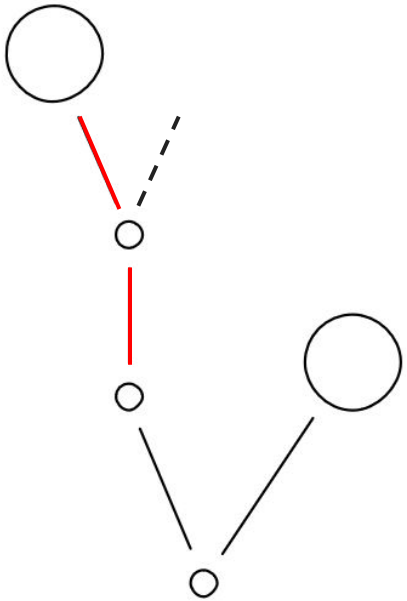




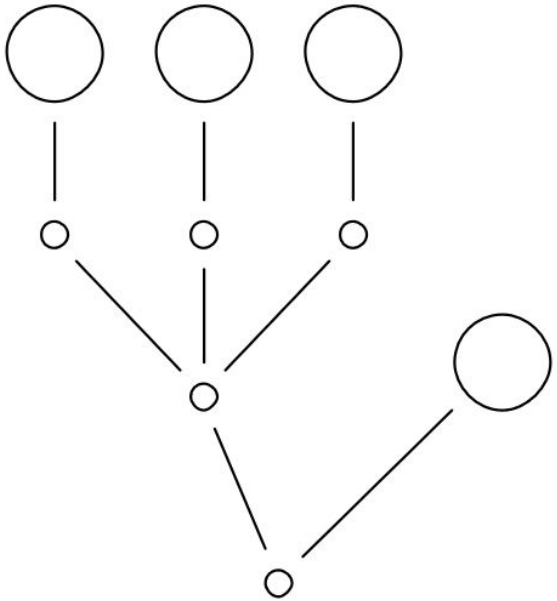
Continuity



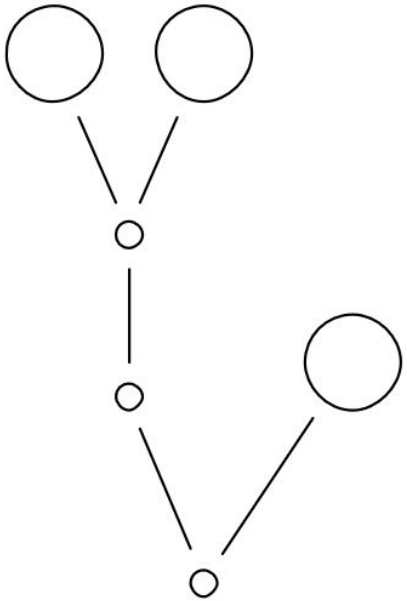
Continuity



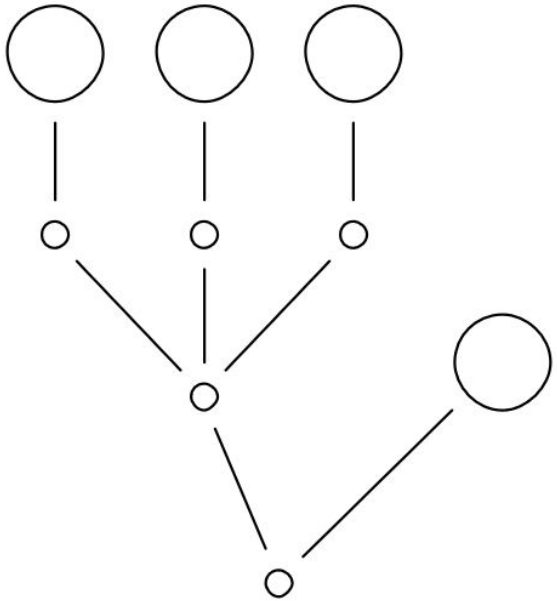
Continuity



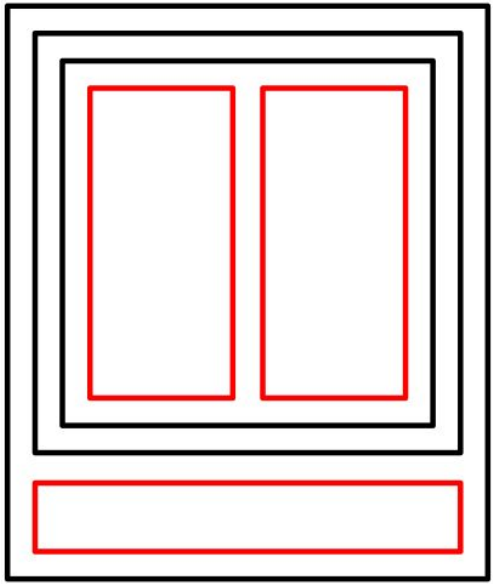
Continuity



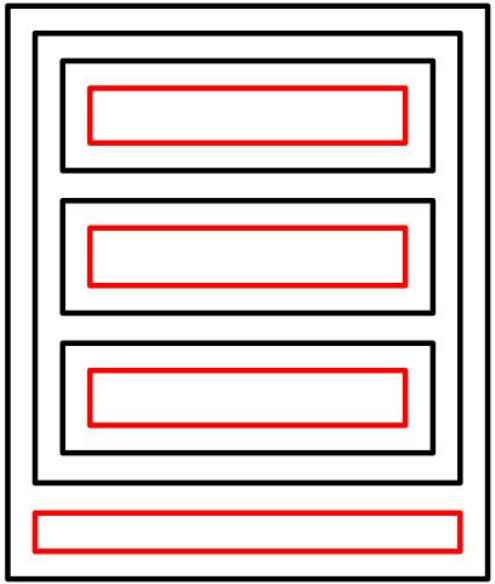
Continuity



Continuity

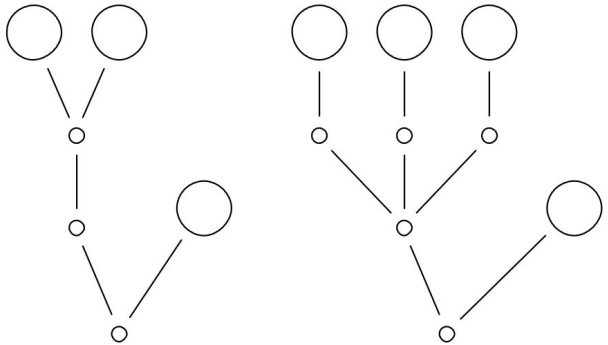


Continuity



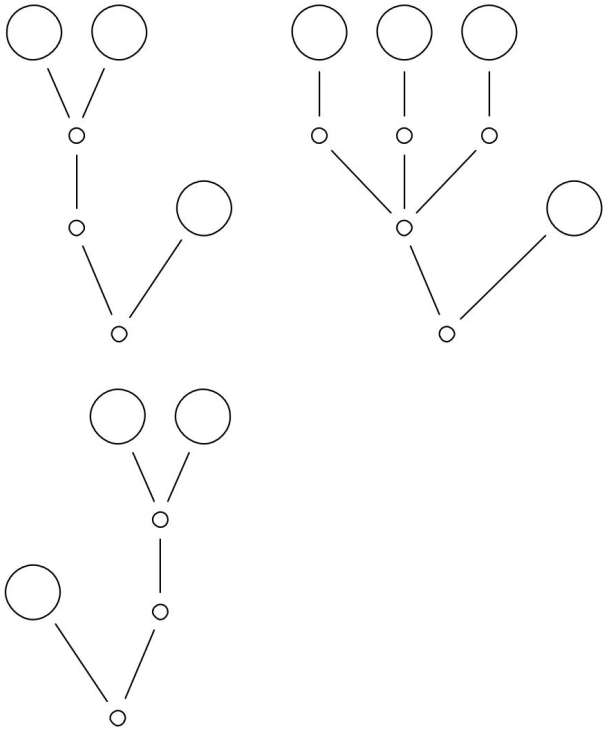
Canonicity

Canonicity



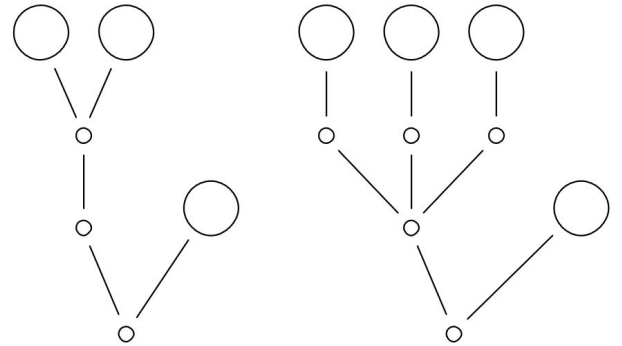
- One possible diagram per hydra
- Children are ordered, visually and syntactically

Canonicity

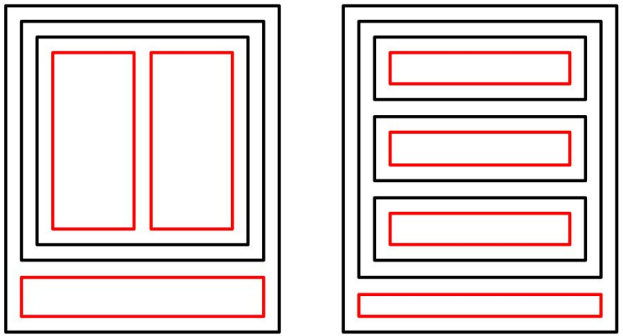


- One possible diagram per hydra
- Children are ordered, visually and syntactically

Canonicity



- One possible diagram per hydra
- Children are ordered, visually and syntactically



- Many possible diagrams per hydra
- Syntactic order ↔ visual order?

Conclusion

- We need diagrammatic notations in computer-assisted proofs, but they are mostly missing.

Conclusion

- We need diagrammatic notations in computer-assisted proofs, but they are mostly missing. Why?

Conclusion

- We need diagrammatic notations in computer-assisted proofs, but they are mostly missing. Why?
- Let’s learn from the success of **text notations**.

Conclusion

- We need diagrammatic notations in computer-assisted proofs, but they are mostly missing. Why?
- Let's learn from the success of **text notations**.
  - Low cost, high reward
  - Structure-driven composition
  - Continuity
  - Canonicity
  - Good-looking idioms

Conclusion

- We need diagrammatic notations in computer-assisted proofs, but they are mostly missing. Why?
- Let's learn from the success of **text notations**.
  - Low cost, high reward
  - Structure-driven composition
  - Continuity
  - Canonicity
  - Good-looking idioms
- An ideal **diagrammatic notation** system translates these properties to diagrams.

Our plan

- Study diagram language design.
  - for graphs (math & stats)
  - for individual diagrams vs. notations
- Study diagram workflows.
  - interactivity
  - theorem prover integration
- Study diagrammatic calculi.
- Build stuff.

Thank you!

Questions?

Diagrammatic notations for ITP

- We need diagrammatic notations in computer-assisted proofs, but they are mostly missing. Why?
- Let's learn from the success of **text notations**.
  - Low cost, high reward
  - Structure-driven composition
  - Continuity
  - Canonicity
  - Good-looking idioms
- An ideal **diagrammatic notation** system translates these properties to diagrams.

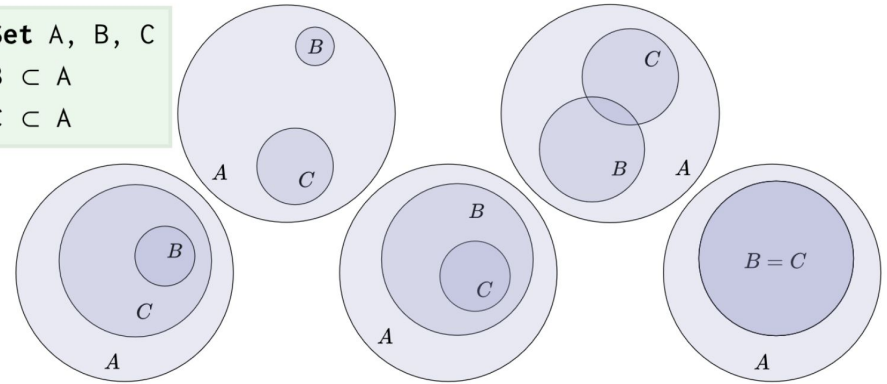
Penrose

- Moderate cost, high reward:
  Offline generation of **individual** diagrams
- Structure-driven composition:
  “Domain” + “stylesheet” = “substance” → diagram
- Continuity: non-goal
- Canonicity: non-goal
- Good-looking idioms

Penrose

```
forall Set x; Set y
where IsSubset(x, y) {
    ensure contains(y.shape, x.shape)
    ensure smallerThan(x.shape, y.shape)
    ensure outsideOf(y.text, x.shape)
    layer x.shape above y.shape
    layer y.text below x.shape
}
```

```
Set A, B, C
B ⊂ A
C ⊂ A
```



ProofWidgets (Lean 4)

- High cost, high reward:
  Fully general HTML+JS integration
- Structure-driven composition, continuity, canonicity:
  Largely up to the UI programmer
- Good-looking idioms:
  UI components, interactivity, extensibility

ProofWidgets (Lean 4)

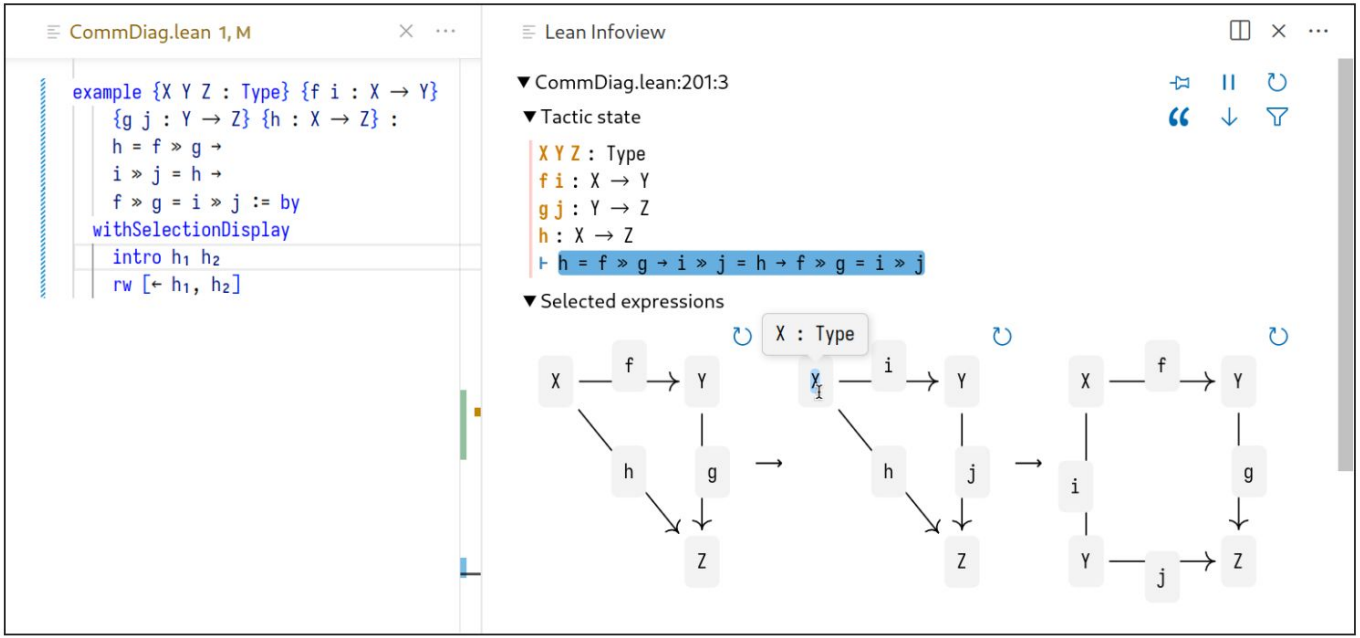


Figure 4 A target type in the language of category theory is selected. The statement is displayed as a sequence of commutative diagrams connected by implication arrows.

D3.js

- High cost, high reward:
  “D3 is for bespoke visualization. D3 makes things possible, not necessarily easy; […] D3 makes sense for media organizations […] where a single graphic may be seen by a million readers”
- Structure-driven composition:
  “Data-driven diagrams”
- Continuity, canonicity: ??
- Good-looking idioms:
  Are the soul of D3

Hydras & Co.

- Textbook written in literate Coq style with Alectryon

Likewise, the *height* (maximum distance between the foot and a head) is defined by mutual recursion:

```
Fixpoint height  (h:Hydra) : nat :=
  match h with
  | node l => lheight l
  end
with lheight l : nat :=
       match l with
       | hnil => 0
       | hcons h hs => Nat.max (S (height h)) (lheight hs)
       end.

  Compute height Hy.
```

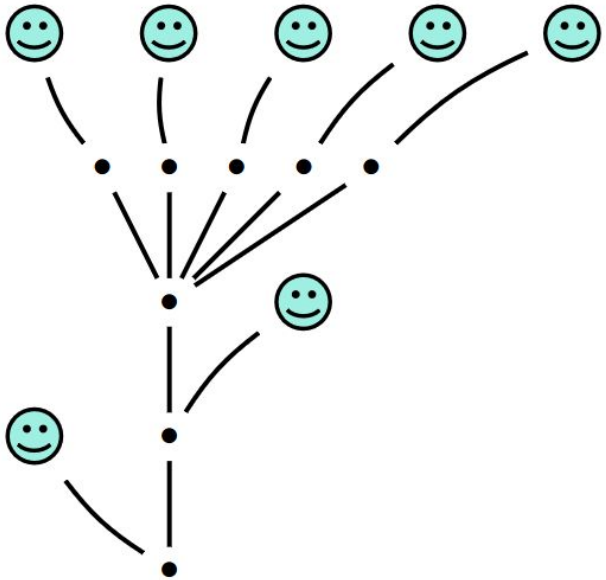
```
= 4
: nat
```

Hydras & Co.

Let us consider for instance the hydra `Hy''` of Fig 2.5 on page 24.

From Module Hydra.Hydra_Examples

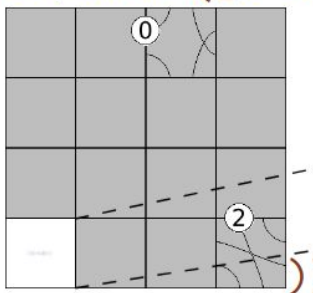
```
Example Hy'' :=
  hyd2 head
       (hyd2
          (hyd_mult (hyd1 head) 5)
          head).
```



Interactive visual syntax (Andersen et al., 2020)

- Turo game tiles as part of Racket syntax

```
; Tile -> [Listof Board]
(define (all-possible-configurations t)
  (for/list ([d DEGREES])
```



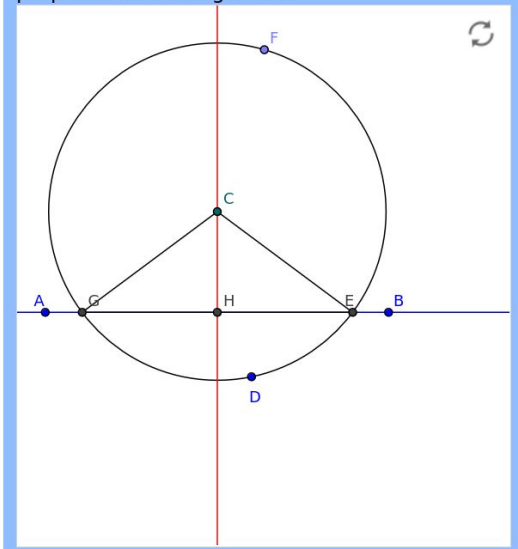

```
))
```

```
(send t rotate d)
```

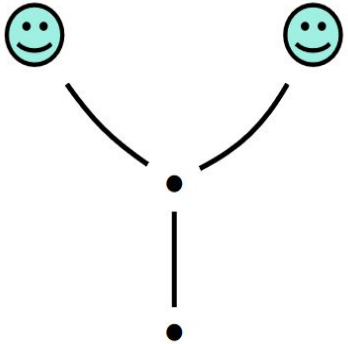
GeoCoq

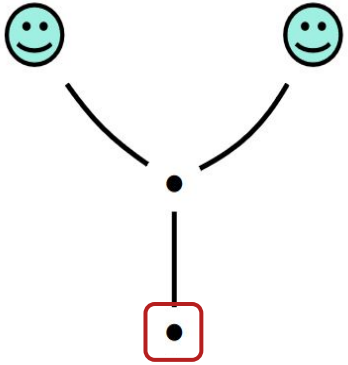
Proposition 12

To a given infinite straight line, from a given point which is not on it, to draw a perpendicular straight line.



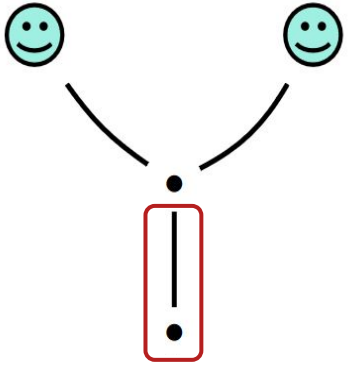

```
Lemma prop_12 : ∀ A B C, ¬ Col A B C →
  ∃ X : Tpoint, Col A B X ∧ Perp A B C X.
Proof.
  apply l8_18_existence.
Qed.
```



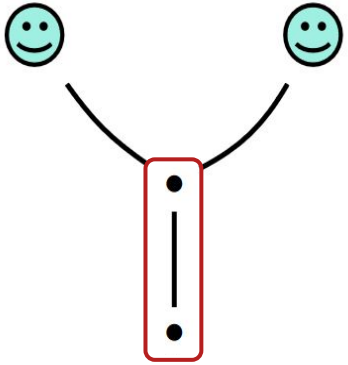


(node

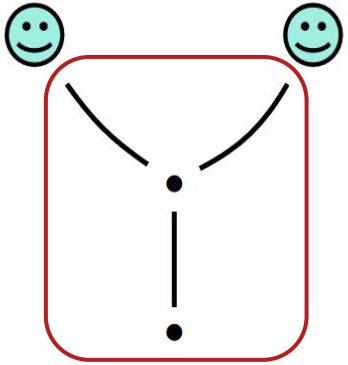
)



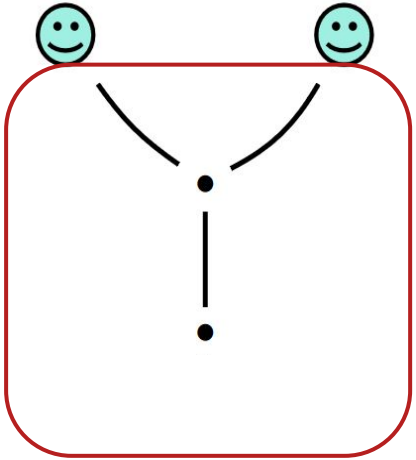
```
(node
  (hcons

   hnil))
```



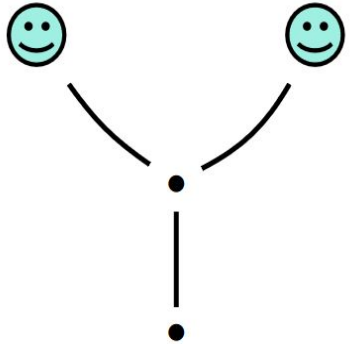
```
(node
  (hcons (node
            )
   hnil))
```



```
(node
  (hcons (node
     (hcons
     (hcons
      hnil)))
   hnil))
```

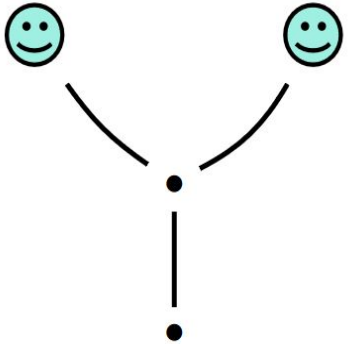


```
(node
  (hcons (node
     (hcons (node hnil)
     (hcons (node hnil)
      hnil)))
   hnil))
```



```
(node
  (hcons (node
     (hcons (node hnil)
     (hcons (node hnil)
      hnil)))
   hnil))
```

```
Notation head := (node hnil).
Notation hyd1 h := (node (hcons h hnil)).
Notation hyd2 h h' := (node (hcons h (hcons h' hnil))).
Notation hyd3 h h' h'' := (node (hcons h (hcons h' (hcons h'' hnil)))).
```



```
(hyd1 (hyd2 head head))
```

```
Notation head := (node hnil).
Notation hyd1 h := (node (hcons h hnil)).
Notation hyd2 h h' := (node (hcons h (hcons h' hnil))).
Notation hyd3 h h' h'' := (node (hcons h (hcons h' (hcons h'' hnil)))).
```