Refactoring of Theory Graphs in Knowledge Representation Systems

Bachelor Thesis in Computer Science

Navid Roux

navid.roux@fau.de

Advisors: Prof. Dr. Michael Kohlhase, PD Dr. Florian Rabe

Friedrich-Alexander-Universität Erlangen-Nürnberg

15th July 2019

Dear online visitor, want to dig into this topic? Have a look at the reading guide at the end: Jump to Reading Guide

Plan

Refactorings

"Improve internal structure without changing observable behavior."

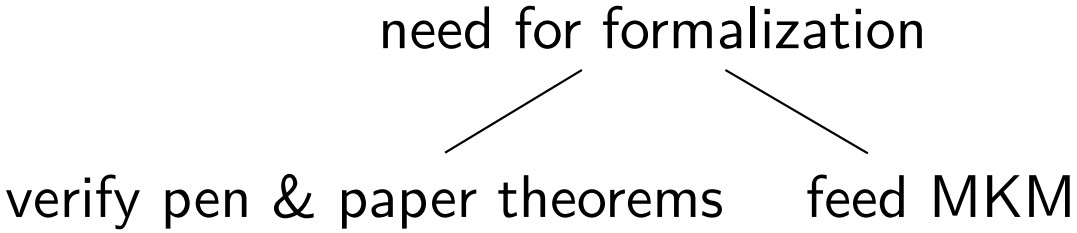
Refactorings

"Improve internal structure without changing observable behavior."

"Neither add nor remove features."

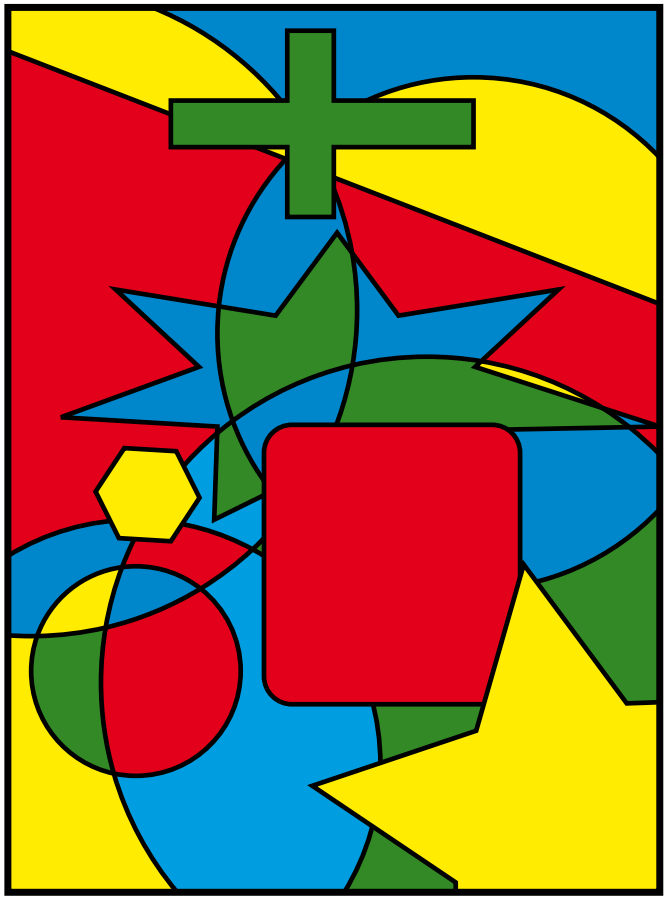
Refactorings on Theory Graphs

Objective: Help in creation & maintenance of formalizations
Reason:



Need For Formalization

Four Color Theorem



Source: Inductiveload on Wikimedia Commons, CC BY-SA 3.0

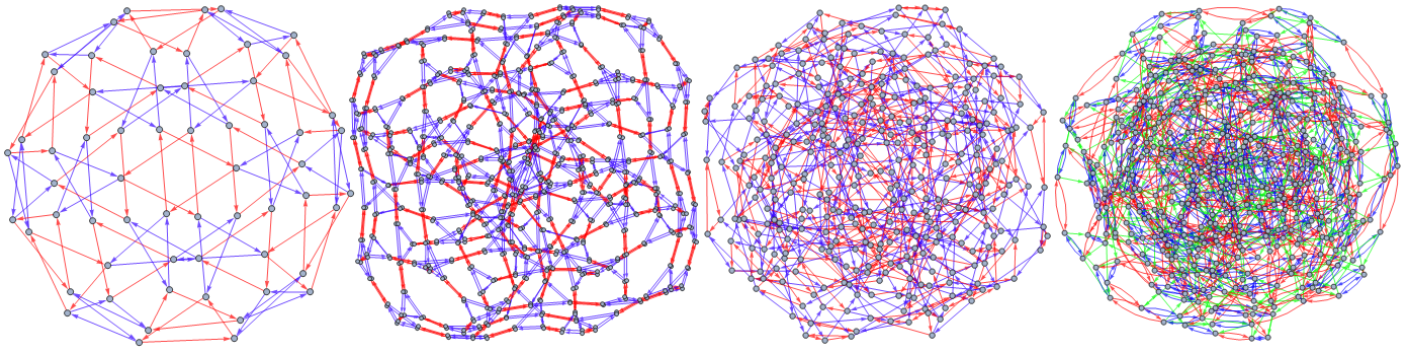
Need For Formalization

Kepler Conjecture

Source: Mike Licht on Flickr, CC BY 2.0

Need For Formalization

Feit-Thompson Theorem



Source: A. W. Walker on awwalker.com

Need For Formalization – MKM Perspective

- Operations on mathematical knowledge
  - cataloguing
  - retrieval
  - refactoring
  - change propagation
- Overcome One-Brain-Barrier & drive data analytics
  $\Rightarrow$ need for formalization

Need For Formalization – MKM Perspective

- Operations on mathematical knowledge
  - cataloguing
  - retrieval
  - refactoring
  - change propagation
- Overcome One-Brain-Barrier & drive data analytics
  $\Rightarrow$ need for formalization
- Generalization Refactorings improve induced knowledge space

Contribution

Focussing on refactorings to generalize, provide

- Framework for generalization refactorings
- Two principles
  - App-Gen — "application of ex. generalization"
  - Theory Splitting — omitted in presentation
- IntelliJ plugin GUI for App-Gen — on top of the MMT plugin

Plan

MMT

What is it?

MMT

What is it?

- Scalable module system for knowledge representation

  esp. formal knowledge

- Theoretical framework *and* implemented system

  "MMT" vs. "MMT system"

MMT

What is it?

- Scalable module system for knowledge representation

  esp. formal knowledge

- Theoretical framework *and* implemented system

  "MMT" vs. "MMT system"

Goals

MMT

What is it?

- Scalable module system for knowledge representation

  esp. formal knowledge
- Theoretical framework *and* implemented system

  "MMT" vs. "MMT system"

Goals

- Foundation independence

MMT

What is it?

- Scalable module system for knowledge representation
  esp. formal knowledge
- Theoretical framework *and* implemented system
  "MMT" vs. "MMT system"

Goals

- Foundation independence
- Minimalistic design
- Standardized representation format

MMT

What is it?

- Scalable module system for knowledge representation
  esp. formal knowledge
- Theoretical framework *and* implemented system
  "MMT" vs. "MMT system"

Goals

- Foundation independence
- Minimalistic design
- Standardized representation format

$\Rightarrow$ Organize knowledge into theories and morphisms

MMT Theories

```
theory Monoid =
  include ?LF
  include ?NatDed

  U: type
  e: U
  op: U → U → U | # #1 ∘ #2
  associative:
    ⊢ ∀ [a: U] ∀ [b: U] ∀ [c: U] (a ∘ b) ∘ c ≐ a ∘ (b ∘ c)

  neutral: U → prop
     | = [e'] ∀ [a: U] (a ∘ e' ≐ a) ∧ (e' ∘ a ≐ a)
  e_neutral: ⊢ neutral e

  e_unique: ⊢ ∀ [e': U] (neutral e') ⇒ e' ≐ e
     | = … |  /T Proof omitted
|
```

MMT Morphisms

Primer

Morphisms $S \rightsquigarrow T$ assign every S-declaration a T-expression
Consider

```
theory Monoid =
  U: type ▮
  e: U ▮
  op: U → U → U ▮
  associative: … ▮

  neutral: … | = … ▮
  e_neutral: … ▮

  e_unique: … | = … ▮
▮
```

```
theory Nat =
  ℕ: type ▮
  0: ℕ ▮
  s: ℕ → ℕ ▮

  /T Peano axioms … ▮

  plus: ℕ → ℕ → ℕ | = … ▮
```

One possibility: `U ↦ ℕ`, `e ↦ 0`, `op ↦ plus`, …

MMT Morphisms

Primer (cont.)

```
view σ : Monoid -> Nat =
  U = ℕ
  e = 0
  op = plus
  associative = …
  e_neutral = …
```

- Only need assignment to undefined declarations
- Gives rise to homomorphic extension $\overline{\sigma} : \texttt{Obj}(Monoid) \to \texttt{Obj}(Nat)$

MMT Morphisms

Truth Preservation

Assume well-typedness everywhere.

```
theory Monoid =
  // … 

  e_unique: ⊢ ∀ [e': U] (neutral e') ⟹ e' ≐ e
    | = … |  /T Proof omitted 
```

Then we get

$$\overline{\sigma}(\text{e\_unique})$$

```
  : ⊢ ∀ [n: ℕ] (neutral n) ⟹ n ≐ 0
  = …
```

MMT Morphisms

Truth Preservation

Assume well-typedness everywhere.

```
theory Monoid =
  // … 

  e_unique: ⊢ ∀ [e': U] (neutral e') ⟹ e' ≐ e
    | = … |  /T Proof omitted 
```

Then we get

$$\overline{\sigma}(\text{e\_unique})$$
```
  : ⊢ ∀ [n: ℕ] (neutral n) ⟹ n ≐ 0
  = …
```

Corollary (Truth Preservation)

Morphisms translate theorems to theorems.

MMT Morphisms

Meaning

They represent syntactically

MMT Morphisms

Meaning

They represent syntactically

- translation
- specialization
- refinement

from poorer into richer theories.

MMT Morphisms

Meaning

They represent syntactically

- translation
- specialization
- refinement

from poorer into richer theories.

Model-theoretically they represent model induction of more general models

every Nat-model induces a $Monoid$-model.

MMT Morphisms

Meaning

They represent syntactically

- translation
- specialization
- refinement

from poorer into richer theories.

Model-theoretically they represent model induction of more general models

every Nat-model induces a $Monoid$-model.

$$TopologicalSpace \rightsquigarrow MetricSpace \rightsquigarrow NormedVectorSpace \rightsquigarrow HilbertSpace$$

Plan

Generalizations

Definition

We call a theory G a **(theory-level) generalization** of a theory T if there is a morphism $g\colon G \rightsquigarrow T$.

Generalizations

Definition

We call a theory G a **(theory-level) generalization** of a theory T if there is a morphism $g\colon G \nrightarrow T$.
A **theory-level generalization principle** is a partial algorithm accepting T and outputting (G, g).

Behavior Preservation

Primer

When is $G \rightsquigarrow T$ behavior-preserving?

- Prevent information loss
- Prevent information addition

opposes generalization!

Behavior Preservation

Primer

When is $G \rightsquigarrow T$ behavior-preserving?

- Prevent information loss — opposes generalization!
- Prevent information addition

Intuition for inclusions

- Not behavior-preserving: removing undefined declarations
  e.g. $Monoid \hookrightarrow Group$
- Behavior-preserving: removing defined declarations
  e.g. all theorems of a theory

Behavior Preservation

Primer

When is $G \rightsquigarrow T$ behavior-preserving?

- Prevent information loss — opposes generalization!
- Prevent information addition

Intuition for inclusions

- Not behavior-preserving: removing undefined declarations
  e.g. $Monoid \hookrightarrow Group$
- Behavior-preserving: removing defined declarations
  e.g. all theorems of a theory

Intuition in general: preserving iff. the same theorems can be derived?

$\Rightarrow$ Impossible, but close!

Behavior Preservation

Definition

Definition

A generalization $g\colon G \rightsquigarrow T$ is called **behavior-preserving** if

$$T \subseteq \bar{g}(\texttt{Obj}(G))$$

where

- $\texttt{Obj}(\cdot)$ denotes all closed well-typed G-expressions
- $\subseteq$ as follows: for every $(c : E[= e]) \in T^\flat$ either $c \in \bar{g}(\texttt{Obj}(G))$ or $e \in \bar{g}(\texttt{Obj}(G))$.

Behavior Preservation

Fulfilled examples

$$T \subseteq \bar{g}(\texttt{Obj}(G))$$

- Removing undefined declarations *not* behavior-preserving
- Removing defined declarations behavior-preserving
- Following chain not behavior-preserving

$$TopologicalSpace \rightsquigarrow MetricSpace \rightsquigarrow NormedVectorSpace \rightsquigarrow HilbertSpace$$

$\Rightarrow$ Weaken to behavior preservation to subset of T

Behavior Preservation

Weakening to subset

Definition

A generalization $g\colon G \rightsquigarrow T$ is called **behavior-preserving wrt.** $T' \subseteq T$ if

$$T' \subseteq \bar{g}(\mathtt{Obj}(G))$$

- every morphism behavior-preserving wrt. its image

Behavior Preservation

Weakening to subset

Definition

A generalization $g\colon G \rightsquigarrow T$ is called **behavior-preserving wrt.** $T' \subseteq T$ if

$$T' \subseteq \bar{g}(\mathtt{Obj}(G))$$

- every morphism behavior-preserving wrt. its image
- $TopologicalSpace$ "behavior-preserving" wrt. continuity definitions in $MetricSpace$
- $MetricSpace$ "behavior-preserving" wrt. continuity definitions in $NormedVectorSpace$
- ...

Behavior Preservation

Weakening to subset

Definition

A generalization $g\colon G \rightsquigarrow T$ is called **behavior-preserving wrt.** $T' \subseteq T$ if

$$T' \subseteq \bar{g}(\mathtt{Obj}(G))$$

- every morphism behavior-preserving wrt. its image
- $TopologicalSpace$ "behavior-preserving" wrt. continuity definitions in $MetricSpace$
- $MetricSpace$ "behavior-preserving" wrt. continuity definitions in $NormedVectorSpace$
- ...

$\Rightarrow$ Weaken to allow $\alpha\beta\eta$ equational theory and more (future work)

Behavior Preservation

Mergesort – behavior-preserving to relevant subset

```
theory MergeOnNat =
  include ?Nat
  include ?PolymorphicLists

  merge: ℕʷ ⟶ ℕʷ ⟶ ℕʷ

  merge_ind_step: ⊦ …
    merge (x::xs) (y::ys) ≐
      if (x ≤ y)
        x :: (merge xs (y::ys))
      else
        y :: (merge (x::xs) ys)

  mergesort: ℕʷ ⟶ ℕʷ
  // …
```

Behavior Preservation

Mergesort – behavior-preserving to relevant subset

```
theory MergeOnToset =
  include ?Toset
  include ?PolymorphicLists

  merge: X* → X* → X*

  merge_ind_step: ⊢ …
    merge (x::xs) (y::ys) ≐
      if (x ≤x y)
        x :: (merge xs (y::ys))
      else
        y :: (merge (x::xs) ys)

  mergesort: X* → X*
  // …
```

```
theory MergeOnNat =
  include ?Nat
  include ?PolymorphicLists

  merge: ℕ* → ℕ* → ℕ*

  merge_ind_step: ⊢ …
    merge (x::xs) (y::ys) ≐
      if (x ≤ y)
        x :: (merge xs (y::ys))
      else
        y :: (merge (x::xs) ys)

  mergesort: ℕ* → ℕ*
  // …
```

Behavior Preservation

Mergesort – behavior-preserving to relevant subset

```
theory MergeOnToset =
  include ?Toset
  include ?PolymorphicLists

  merge: X^w → X^w → X^w

  merge_ind_step: ⊢ …
    merge (x::xs) (y::ys) ≐
      if (x ≤_X y)
        x :: (merge xs (y::ys))
      else
        y :: (merge (x::xs) ys)

  mergesort: X^w → X^w
  // …
```

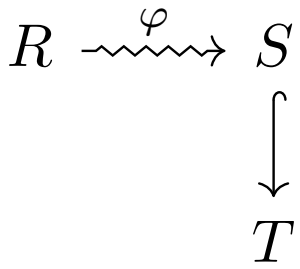
```
theory MergeOnNat =
  include ?Nat
  include ?PolymorphicLists

  merge: ℕ^w → ℕ^w → ℕ^w

  merge_ind_step: ⊢ …
    merge (x::xs) (y::ys) ≐
      if (x ≤ y)
        x :: (merge xs (y::ys))
      else
        y :: (merge (x::xs) ys)

  mergesort: ℕ^w → ℕ^w
  // …
```

Plan

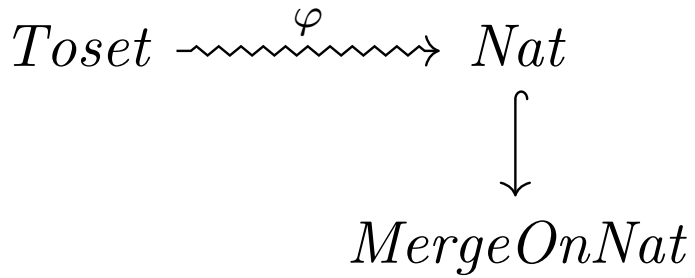
App-Gen

Overall task

Abstract task: Given



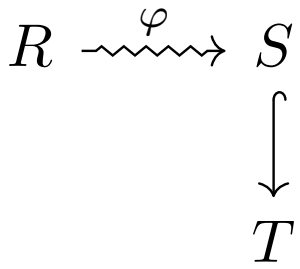
Concrete task: Given



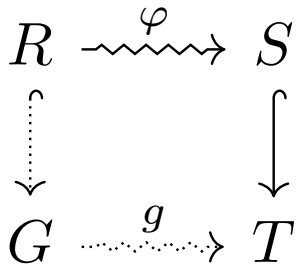
App-Gen

Overall task

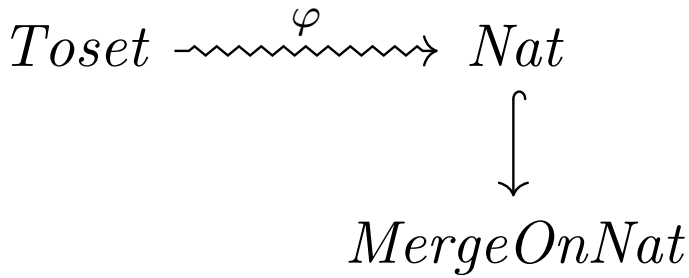
Abstract task: Given



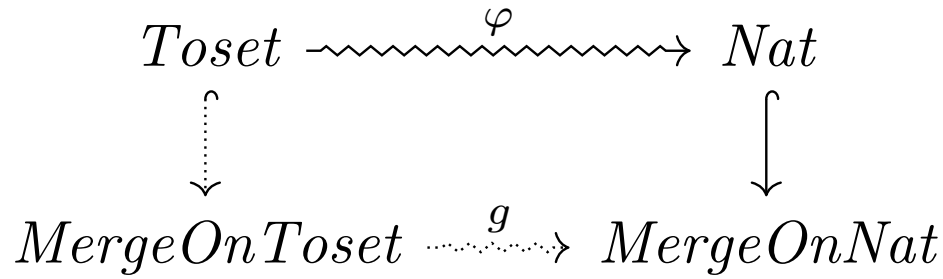
come up with



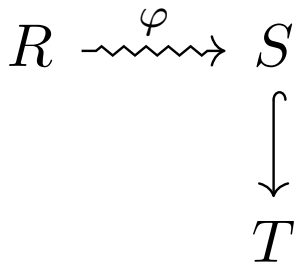
Concrete task: Given



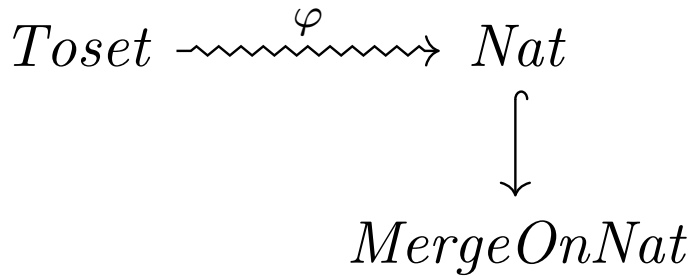
come up with



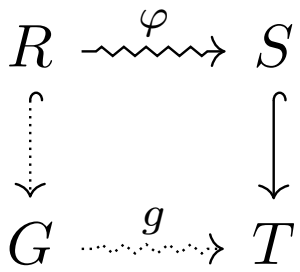
Abstract task: Given



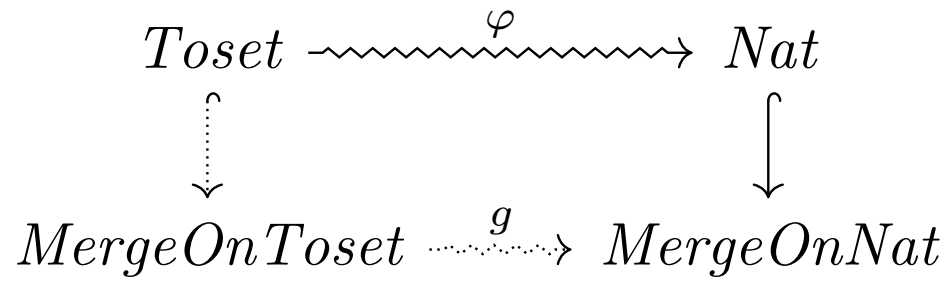
Concrete task: Given



come up with



come up with



"Generalize T along φ"

How to generalize?

- Replace `include ?Nat` by `include ?Toset`
- Replace $\mathbb{N}$ by X
- Replace $\leq$ by $\leq_X$

How to generalize?

- Replace `include ?Nat` by `include ?Toset`
- Replace $\mathbb{N}$ by X
- Replace $\leq$ by $\leq_X$

```
theory Toset =
  X: type
  ≤ₓ: X → X → prop

  ax_antisymmetry: ⊢ …
  ax_transivitivty: ⊢ …
  ax_connexity: ⊢ …
```

```
view φ : Toset -> Nat =
  X = ℕ
  ≤ₓ = ≤

  ax_antisymmetry = …
  ax_transitivity = …
  ax_connexity = …
```

$\Rightarrow$ Replacements given by morphism read backwards

How to generalize?

- Replace `include ?Nat` by `include ?Toset`
- Replace `ℕ` by `X`
- Replace `≤` by `≤ₓ`

$$\frac{\mathbb{N}}{X} \quad \frac{\leq}{\leq_x}$$

```
theory Toset =
  X: type
  ≤ₓ: X → X → prop

  ax_antisymmetry: ⊢ …
  ax_transivitivty: ⊢ …
  ax_connexity: ⊢ …
```

```
view φ : Toset -> Nat =
  X = ℕ
  ≤ₓ = ≤

  ax_antisymmetry = …
  ax_transitivity = …
  ax_connexity = …
```

⇒ Replacements given by morphism read backwards

App-Gen-0: A First Version

Definition (App-Gen-0)

Build G and $g\colon G \rightsquigarrow T$:

1. rewrite `include ?S` to `include ?R`
2. iteratively rewrite remaining declarations $\delta \in T$ using

$$\frac{c'}{c} \qquad\qquad \frac{d}{d'}$$

   for $(c := c') \in \varphi$ and rewritten d' of d.
3. adopt $g(d') := d$ for every rewritten d' of d

Mergesort Revisited

```
theory MergeOnToset =
  include ?Toset ▮
  include ?PolymorphicLists ▮

  merge: X^ω ⟶ X^ω ⟶ X^ω ▮

  merge_ind_step: ⊢ …
    merge (x::xs) (y::ys) ≐
      if (x ≤_X y)
        x :: (merge xs (y::ys))
      else
        y :: (merge (x::xs) ys)
  ▮

  mergesort: X^ω ⟶ X^ω ▮
  // … ▮
▮
```

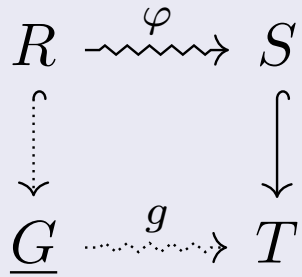
```
theory MergeOnNat =
  include ?Nat ▮
  include ?PolymorphicLists ▮

  merge: ℕ^ω ⟶ ℕ^ω ⟶ ℕ^ω ▮

  merge_ind_step: ⊢ …
    merge (x::xs) (y::ys) ≐
      if (x ≤ y)
        x :: (merge xs (y::ys))
      else
        y :: (merge (x::xs) ys)
  ▮

  mergesort: ℕ^ω ⟶ ℕ^ω ▮
  // … ▮
▮
```

App-Gen-0: Results I

Assume the generated theory G is well-typed, then

Lemma

The morphism g is well-typed and the square



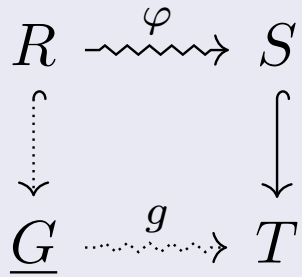
commutes.

App-Gen-0: Results I

Assume the generated theory G is well-typed, then

Lemma

The morphism g is well-typed and the square



commutes.

Corollary (Behavior Preservation)

$g\colon G \rightsquigarrow T$ is a behavior-preserving generalization wrt.

- $T \setminus S$
- $(T \setminus S) \cup S'$ *if φ was beh.-preserving wrt. $S' \subseteq S$.*

App-Gen-0: Limitations

By example

```
theory NormedVectorspaceThms =
  include ?NormedVectorspace

  cauchy: (ℕ ⟶ Y) ⟶ prop = [f] ∀[ε: ℝ] ∃[N: ℕ] ∀[n: ℕ] ∀[m: ℕ]
    (n ≥ N ∧ m ≥ N) ⇒ ((norm ((f n) - (f m))) < ε)
```

App-Gen-0: Limitations

By example

```
theory NormedVectorspaceThms =
  include ?NormedVectorspace

  cauchy: (ℕ ⟶ Y) ⟶ prop | = [f] ∀[ε: ℝ] ∃[N: ℕ] ∀[n: ℕ] ∀[m: ℕ]
    (n ≥ N ∧ m ≥ N) ⇒ ((norm ((f n) - (f m))) < ε)

  convergent_to: (ℕ ⟶ Y) ⟶ Y ⟶ prop | = [f, y] ∀[ε: ℝ] ∃[N: ℕ]
↪   ∀[n: ℕ]
    (n ≥ N) ⇒ ((norm ((f n) - y)) < ε)
```

App-Gen-0: Limitations

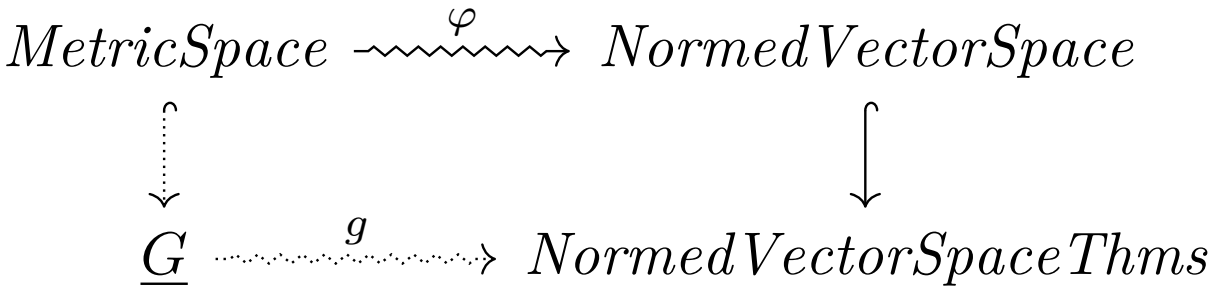
By example

```
theory NormedVectorspaceThms =
  include ?NormedVectorspace

  cauchy: (ℕ ⟶ Y) ⟶ prop | = [f] ∀[ε: ℝ] ∃[N: ℕ] ∀[n: ℕ] ∀[m: ℕ]
    (n ≥ N ∧ m ≥ N) ⇒ ((norm ((f n) - (f m))) < ε)

  convergent_to: (ℕ ⟶ Y) ⟶ Y ⟶ prop | = [f, y] ∀[ε: ℝ] ∃[N: ℕ]
↪   ∀[n: ℕ]
    (n ≥ N) ⇒ ((norm ((f n) - y)) < ε)

  /T Lipschitz continuity of an endofunction
  lipschitz: (Y ⟶ Y) ⟶ prop | = [f] ∀[y₁: Y] ∀[y₂: Y]
    norm ((f y₁) - (f y₂)) ≤ norm (y₁ - y₂)
```

Objective: Generalize to metric spaces

App-Gen-0: Limitations

By example (cont.)




```
theory MetricSpace =
  X : type
  d: X → X → ℝ
```

```
theory NormedVectorspace =
  Y: type
  norm: Y → ℝ
  minus: Y → Y → Y
    | # 1 - 2
```

```
view NormedAsMetricSpace : ?MetricSpace -> ?NormedVectorspace =
  X = Y
  d = [a,b] norm (a - b)
```

App-Gen-0: Limitations

By example (cont.)

Employed rewrite rules

$$\frac{Y}{X} \qquad \frac{\lambda a, b.\ \ \mathrm{norm}\ (a - b)}{\mathrm{d}}$$

insufficient to rewrite

```
cauchy: (ℕ ⟶ Y) ⟶ prop | = [f] ∀[ε: ℝ] ∃[N: ℕ] ∀[n: ℕ] ∀[m: ℕ]
  (n ≥ N ∧ m ≥ N) ⇒ ((norm ((f n) - (f m))) < ε) |
```

App-Gen-0: Limitations

By example (cont.)

Employed rewrite rules

$$\frac{Y}{X} \qquad \frac{\lambda a, b.\ \ \mathrm{norm}\ (a - b)}{\mathrm{d}}$$

insufficient to rewrite

```
cauchy: (ℕ ⟶ Y) ⟶ prop ❙ = [f] ∀[ε: ℝ] ∃[N: ℕ] ∀[n: ℕ] ∀[m: ℕ]
  (n ≥ N ∧ m ≥ N) ⇒ ((norm ((f n) - (f m))) < ε) ❙
```

Would like to have

$$\frac{Y}{X} \qquad \frac{\lambda a, b.\ \ \mathrm{norm}\ (a - b)}{\mathrm{d}} \qquad \frac{\mathrm{norm}\ (a - b)}{\mathrm{d}\ a\ b}$$

App-Gen-1: A Revision

Definition (App-Gen-1)

Build G and $g\colon G \rightsquigarrow T$:

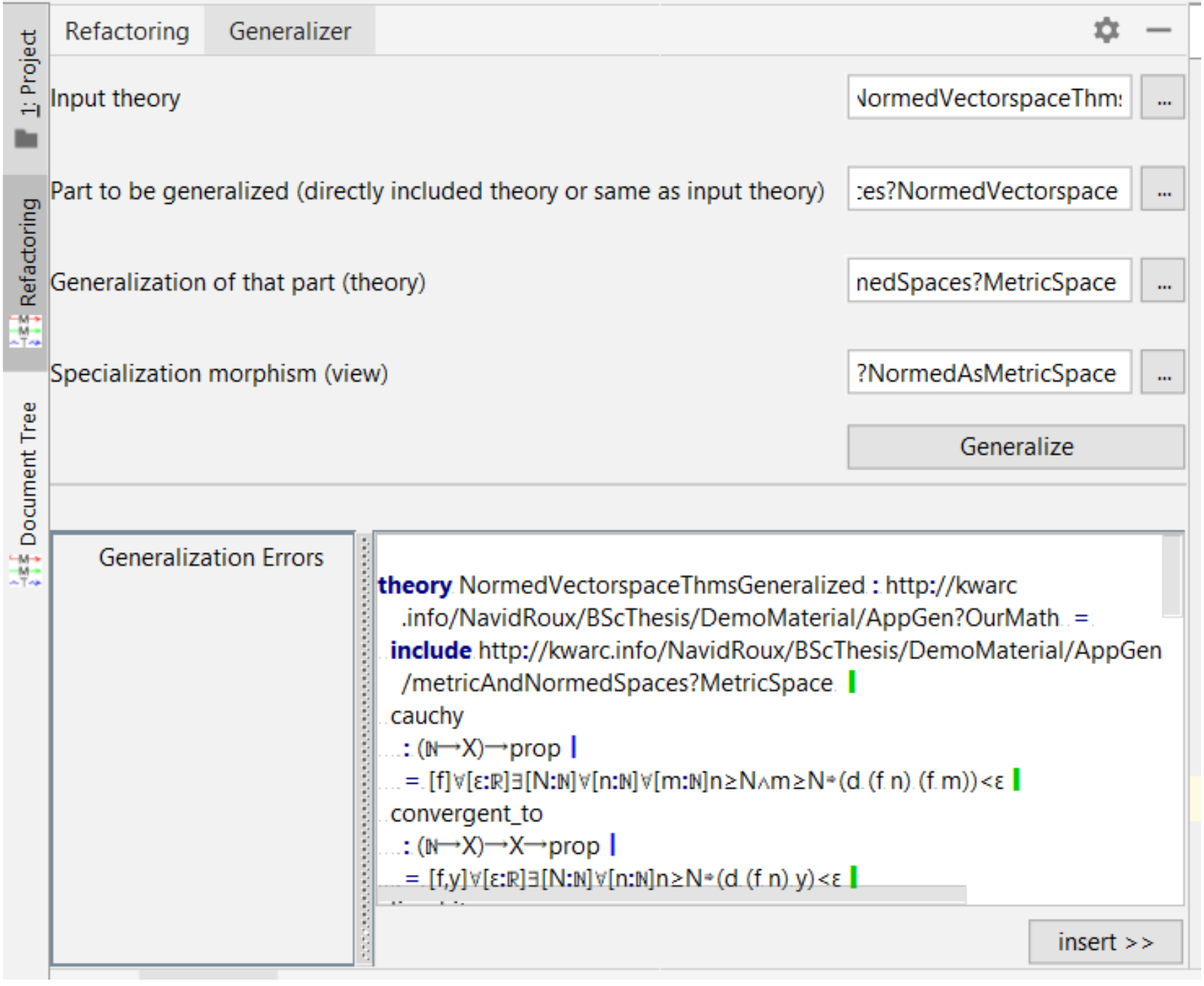
1. rewrite `include ?S` to `include ?R`
2. iteratively rewrite remaining declarations $\delta \in T$ using

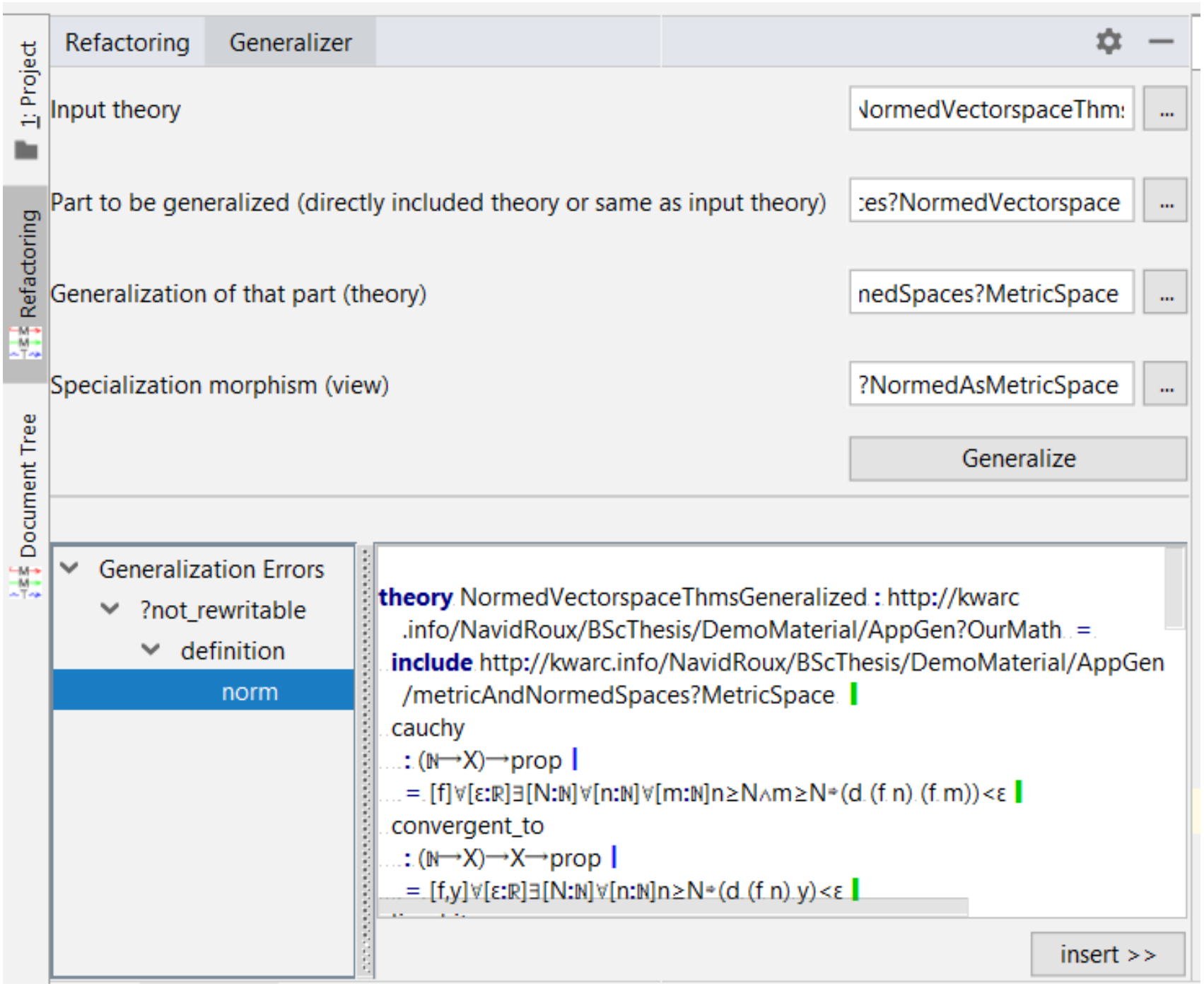
$$\frac{c'}{c} \qquad \frac{s\sigma}{c\ \sigma(x_1)\ldots\sigma(x_n)}\ \ \mathrm{dom}(\sigma) = \{x_1,\ldots,x_n\} \qquad \frac{d}{d'}$$

   for $(c := c'), (c := \lambda x_1,\ldots,x_n.\ \ s) \in \varphi$ and rewritten d' of d.
3. adopt $g(d') := d$ for every rewritten d' of d

Live Demo

Skip after backup slides


1: Project
Refactoring
Document Tree
Refactoring
Generalizer
Input theory
NormedVectorspaceThm:
...
Part to be generalized (directly included theory or same as input theory)
:es?NormedVectorspace
...
Generalization of that part (theory)
nedSpaces?MetricSpace
...
Specialization morphism (view)
?NormedAsMetricSpace
...
Generalize
Generalization Errors
theory NormedVectorspaceThmsGeneralized : http://kwarc
.info/NavidRoux/BScThesis/DemoMaterial/AppGen?OurMath =
include http://kwarc.info/NavidRoux/BScThesis/DemoMaterial/AppGen
/metricAndNormedSpaces?MetricSpace
cauchy
: (ℕ⟶X)⟶prop
= [f]∀[ε:ℝ]∃[N:ℕ]∀[n:ℕ]∀[m:ℕ]n≥N∧m≥N⇒(d (f n) (f m))<ε
convergent_to
: (ℕ⟶X)⟶X⟶prop
= [f,y]∀[ε:ℝ]∃[N:ℕ]∀[n:ℕ]n≥N⇒(d (f n) y)<ε
insert >>


1: Project
Refactoring
Document Tree
Refactoring
Generalizer
Input theory
lormedVectorspaceThm:
...
Part to be generalized (directly included theory or same as input theory)
:es?NormedVectorspace
...
Generalization of that part (theory)
nedSpaces?MetricSpace
...
Specialization morphism (view)
?NormedAsMetricSpace
...
Generalize
Generalization Errors
?not_rewritable
definition
norm
theory NormedVectorspaceThmsGeneralized : http://kwarc
.info/NavidRoux/BScThesis/DemoMaterial/AppGen?OurMath =
include http://kwarc.info/NavidRoux/BScThesis/DemoMaterial/AppGen
/metricAndNormedSpaces?MetricSpace
cauchy
: (ℕ⟶X)⟶prop
= [f]∀[ε:ℝ]∃[N:ℕ]∀[n:ℕ]∀[m:ℕ]n≥N∧m≥N⇒(d (f n) (f m))<ε
convergent_to
: (ℕ⟶X)⟶X⟶prop
= [f,y]∀[ε:ℝ]∃[N:ℕ]∀[n:ℕ]n≥N⇒(d (f n) y)<ε
insert >>

App-Gen-1: Outlook

- In general we need a strong enough equational theory
- Might include user-defined T-theorems

$\Rightarrow$ Complexity of higher-order rewriting and unification

Plan

Conclusion

Summary

- Refactorings useful on formalized mathematics

Conclusion

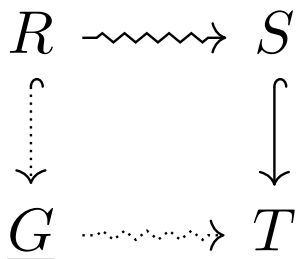
Summary

- Refactorings useful on formalized mathematics
- Framework for generalization refactorings
  - G generalization of $T :\Leftrightarrow$ there is a morphism $g : G \rightsquigarrow T$
  - Behavior-preserving wrt. $T :\Leftrightarrow T \subseteq g(G)$

Conclusion

Summary

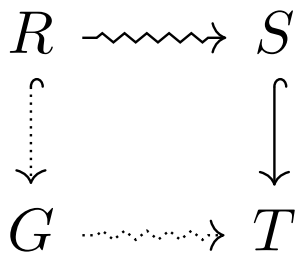
- Refactorings useful on formalized mathematics
- Framework for generalization refactorings
  - G generalization of $T :\Leftrightarrow$ there is a morphism $g : G \rightsquigarrow T$
  - Behavior-preserving wrt. $T :\Leftrightarrow T \subseteq g(G)$
- App-Gen captures abstraction along morphisms:



Conclusion

Summary

- Refactorings useful on formalized mathematics
- Framework for generalization refactorings
  - G generalization of $T :\Leftrightarrow$ there is a morphism $g : G \rightsquigarrow T$
  - Behavior-preserving wrt. $T :\Leftrightarrow T \subseteq g(G)$
- App-Gen captures abstraction along morphisms:

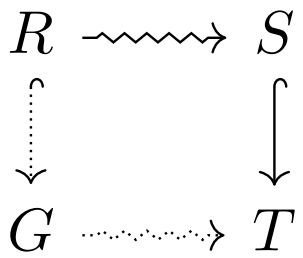


- Prototypical GUI for App-Gen

Conclusion

Summary

- Refactorings useful on formalized mathematics
- Framework for generalization refactorings
  - G generalization of $T :\Leftrightarrow$ there is a morphism $g : G \rightsquigarrow T$
  - Behavior-preserving wrt. $T :\Leftrightarrow T \subseteq g(G)$
- App-Gen captures abstraction along morphisms:



- Prototypical GUI for App-Gen

Future Work

- Evaluation of theory & applicability
- Measures to drive automated application
- Educational usecase

Appendix

Reading Guide I

An incomplete list of introductory references I would have personally recommended to my former me who desired to dive into this presentation's topics.

- My Thesis: [Rou19]
- Mathematical Knowledge Management: [Koh14; Car+19]
- MMT:
  - Overview: [MMT; Rab18]
  - Tutorial for Formalization: [OMT]
  - MMT Plugin for IntelliJ IDEA: [Intellij-MMT]
  - Categorical constructions, esp. colimits: [Rab17][1]

[1] This one is not really introductory, but I found some of the ideas very enlightening and deepening my understanding of formal systems. Even without understanding all details, it is a valuable read for exactly this reason.

Reading Guide II

- Refactoring
  - Pioneering works: [Opd92; Fow19][2]
  - Type Abstraction in Software Engineering: [BFS07; Tip07][3]
  - Behavior Preservation, short Survey on Refactoring: [Whi13, chs. 7.4 – 7.5]
- Last, but not least: talking to people!
  Especially my advisors, but as well other people at the research group.

As always, dig down the rabbit hole if you want to know something in great detail. In case you want to know more how my work relates to others, have a look at the related work section in my thesis [Rou19, ch. 2.3].

[2] Primarily to get a feel for what software engineers would call refactoring. You may particularly peek at how [Opd92] proves behavior preservation properties.

[3] These works do "subtype lifting" while APP-GEN does the more general "morphism lifting".

References I

Markus Bach, Florian Forster, and Friedrich Steimann. "Declared Type Generalization Checker: An Eclipse Plug-In for Systematic Programming with More General Types". In: *Fundamental Approaches to Software Engineering*. Ed. by Matthew B. Dwyer and Antónia Lopes. Vol. 4422. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 117–120. ISBN: 978-3-540-71288-6 978-3-540-71289-3. DOI: 10.1007/978-3-540-71289-3_10. URL: http://link.springer.com/10.1007/978-3-540-71289-3_10 (visited on 03/12/2019).

Jacques Carette et al. "Big Math and the One-Brain Barrier – A Position Paper and Architecture Proposal". submitted to Mathematical Intelligencer. 2019. URL: https://arxiv.org/abs/1904.10405.

References II

Martin Fowler. *Refactoring: improving the design of existing code*. Second edition. Addison-Wesley signature series. OCLC: on1064139838. Boston: Addison-Wesley, 2019. 418 pp. ISBN: 978-0-13-475759-9.

UniFormal/IntelliJ-MMT – An IntelliJ-Plugin for MMT. URL: https://github.com/UniFormal/IntelliJ-MMT (visited on 06/13/2019).

Michael Kohlhase. "Mathematical Knowledge Management: Transcending the One-Brain-Barrier with Theory Graphs". In: *EMS Newsletter* (June 2014), pp. 22–27. URL: https://kwarc.info/people/mkohlhase/papers/ems13.pdf.

UniFormal/MMT – The MMT Language and System. URL: https://github.com/UniFormal/MMT (visited on 10/24/2017).

References III

Michael Kohlhase and Dennis Müller. *OMDoc/MMT Tutorial for Mathematicians*. URL: https://gl.mathhub.info/Tutorials/Mathematicians/blob/master/tutorial/mmt-math-tutorial.pdf (visited on 10/07/2017).

William F. Opdyke. "Refactoring Object-oriented Frameworks". PhD thesis. Champaign, IL, USA: University of Illinois at Urbana-Champaign, 1992.

Florian Rabe. "How to Identify, Translate, and Combine Logics?" In: *Journal of Logic and Computation* 27.6 (2017), pp. 1753–1798.

Florian Rabe. "MMT: A Foundation-Independent Logical Framework". Online Documentation. 2018. URL: https://kwarc.info/people/frabe/Research/rabe_mmtsys_18.pdf.

References IV

Navid Roux. *Refactoring of Theory Graphs in Knowledge Representation Systems*. B.Sc. Thesis. July 2019. URL: https://gl.kwarc.info/supervision/BSc-archive/blob/master/2019/Roux_Navid.pdf.

Frank Tip. "Refactoring Using Type Constraints". In: *Static Analysis*. Ed. by Hanne Riis Nielson and Gilberto Filé. Springer Berlin Heidelberg, 2007, pp. 1–17. ISBN: 978-3-540-74061-2.

Iain Johnston Whiteside. "Refactoring Proofs". PhD thesis. Edinburgh: The University of Edinburgh, 2013. URL: https://www.era.lib.ed.ac.uk/handle/1842/7970?show=full.