# Change Management in Declarative Languages

Mihnea Iancu and Florian Rabe

Jacobs University Bremen

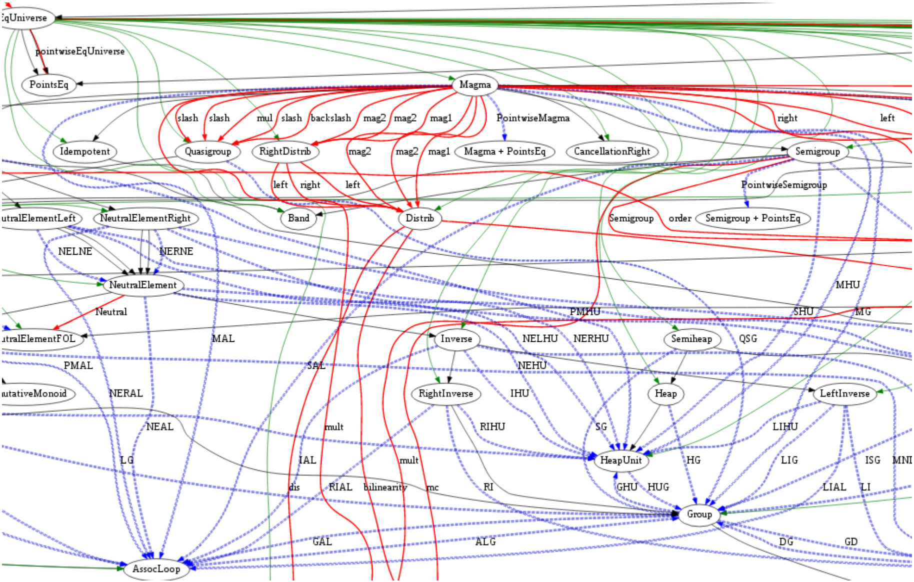
July 13, 2012

# Motivation

- mathematical knowledge grows relentlessly
- mathematics is intrinsically inter-connected
- formal mathematical libraries already too large to oversee
- need for adequate change management solutions

# Motivation: LATIN Library

- LATIN : an atlas of logic formalizations
  - inter-connected network of ∼1000 modules
  - based on the MMT/LF logical framework
  - highly modular (Little Theories approach)
- difficult to keep an overview (modularity helps but is not enough)
  - which declarations does the symbol $s$ depend on
  - which declarations depend on the symbol $s$


EqUniverse
pointwiseEqUniverse
PointsEq
Magma
slash
slash
mul
slash
backslash
mag2
mag2
mag1
PointwiseMagma
right
left
Idempotent
Quasigroup
RightDistrib
mag2
mag2
mag1
Magma + PointsEq
CancellationRight
Semigroup
left
right
left
PointwiseSemigroup
NeutralElementLeft
NeutralElementRight
Band
Distrib
Semigroup
order
Semigroup + PointsEq
NELNE
NERNE
NeutralElement
MHU
Neutral
PMHU
SHU
MG
NeutralElementFOL
MAL
Inverse
NELHU
NERHU
Semiheap
QSG
PMAL
SAL
NEHU
CommutativeMonoid
NERAL
RightInverse
IHU
Heap
LeftInverse
NEAL
mult
RIHU
SG
LIHU
LG
IAL
mult
HeapUnit
HG
LIG
ISG
MNI
dis
RIAL
bilinearity
mc
RI
GHU
HUG
LIAL
LI
Group
GAL
ALG
DG
GD
AssocLoop

# MMT

- a Module System for Mathematical Theories
- generic declarative language

  theories, morphisms, declarations, expressions

  module system
- OMDoc/OpenMath-based XML syntax with Scala-based API
- foundationally independent
  - no commitment to a particular logic or logical framework

    both represented as MMT theories
  - concise and natural representation of a variety of systems

    e.g. Twelf, Mizar, TPTP, OWL

# MMT-based MKM services

Foundation independence → MMT services carry over to languages represented in MMT

- presentation — MKM 2008
- interactive browsing — MKM 2009
- database — MKM 2010
- archival, project management — MKM 2011
- querying — Tuesday, MKM 2012
- editing (work in progress) — Wednesday, UITP 2012

- management of change (MoC) — now, AISC 2012

# Outline

## Management of Change

- MoC is not a new topic; usually involves
  - detect changes

    see if/how something changed
  - compute affected items

    maintain some notion of dependency
  - handle/identify conflicts

    in SE typically re-compile e.g. Eclipse

# Outline

## Management of Change

- MoC is not a new topic; usually involves
  - detect changes

    see if/how something changed
  - compute affected items

    maintain some notion of dependency
  - handle/identify conflicts

    in SE typically re-compile e.g. Eclipse

## Outline

- semantic differencing
- fine-grained dependencies
- impact propagation
- some form of a validity guarantee

# MMT Example

## MMT Notions

| | |
|---|---|
| theories | contain constant declarations |
| constants | have components (type and definiens) |
| components | represented as MMT/OPENMATH terms |
| URIs | for each theory/constant/component |

$Rev_1$

$$
\begin{aligned}
&PL = \{\\
&\quad bool : \mathtt{type}\\
&\quad \Rightarrow : bool \to bool \to bool\\
&\quad \wedge : bool \to bool \to bool\\
&\quad \Leftrightarrow : bool \to bool \to bool\\
&\qquad = \lambda x.\lambda y.(x \Rightarrow y) \wedge (y \Rightarrow x)\\
&\}
\end{aligned}
$$

$Rev_2$

$$
\begin{aligned}
&PL = \{\\
&\quad form : \mathtt{type}\\
&\quad \neg : form \to form\\
&\quad \wedge : bool \to bool \to bool\\
&\quad \Leftrightarrow : bool \to bool \to bool\\
&\qquad = \lambda x.\lambda y.(x \Rightarrow y) \wedge (y \Rightarrow x)\\
&\}
\end{aligned}
$$

# MMT Example

## MMT Notions

| | |
|---|---|
| theories | contain constant declarations |
| constants | have components (type and definiens) |
| components | represented as MMT/OPENMATH terms |
| URIs | for each theory/constant/component |

*Rev*$_1$

$$PL = \{$$
$$bool : \mathtt{type}$$
$$\Rightarrow : bool \to bool \to bool$$
$$\wedge : bool \to bool \to bool$$
$$\Leftrightarrow : bool \to bool \to bool$$
$$= \lambda x.\lambda y.(x \Rightarrow y) \wedge (y \Rightarrow x)$$
$$\}$$

*Rev*$_2$

$$PL = \{$$
$$form : \mathtt{type}$$
$$\neg : form \to form$$
$$\wedge : bool \to bool \to bool$$
$$\Leftrightarrow : bool \to bool \to bool$$
$$= \lambda x.\lambda y.(x \Rightarrow y) \wedge (y \Rightarrow x)$$
$$\}$$

# Semantic Differencing

- we extend MMT with a language of changes
  - add ($\mathcal{A}$) and delete ($\mathcal{D}$) constants
  - update ($\mathcal{U}$) components
  - rename ($\mathcal{R}$) constants

| | | | |
|---|---|---|---|
| Diff | $\Delta$ | $::=$ | $\cdot \mid \Delta, \delta$ |
| Change | $\delta$ | $::=$ | $\mathcal{A}(T, c : \omega = \omega') \mid \mathcal{D}(T, c : \omega = \omega') \mid$ |
| | | | $\mathcal{U}(T, c, o, \omega, \omega') \mid \mathcal{R}(T, c, c')$ |
| Component | $o$ | $::=$ | $\mathtt{tp} \mid \mathtt{def}$ |

# Example Revisited

$Rev_1$

$$
\begin{aligned}
&PL = \{\\
&\quad bool : \texttt{type}\\
&\quad \Rightarrow : bool \to bool \to bool\\
&\quad \wedge : bool \to bool \to bool\\
&\quad \Leftrightarrow : bool \to bool \to bool\\
&\qquad = \lambda x.\lambda y.(x \Rightarrow y) \wedge (y \Rightarrow x)\\
&\}
\end{aligned}
$$

$Rev_2$

$$
\begin{aligned}
&PL = \{\\
&\quad form : \texttt{type}\\
&\quad \neg : form \to form\\
&\quad \wedge : bool \to bool \to bool\\
&\quad \Leftrightarrow : bool \to bool \to bool\\
&\qquad = \lambda x.\lambda y.(x \Rightarrow y) \wedge (y \Rightarrow x)\\
&\}
\end{aligned}
$$

# Example Revisited

*Rev*$_1$

$$
\begin{aligned}
& PL = \{ \\
& \quad bool : \mathtt{type} \\
& \quad \Rightarrow : bool \rightarrow bool \rightarrow bool \\
& \quad \wedge : bool \rightarrow bool \rightarrow bool \\
& \quad \Leftrightarrow : bool \rightarrow bool \rightarrow bool \\
& \qquad = \lambda x.\lambda y.(x \Rightarrow y) \wedge (y \Rightarrow x) \\
& \}
\end{aligned}
$$

*Rev*$_2$

$$
\begin{aligned}
& PL = \{ \\
& \quad form : \mathtt{type} \\
& \quad \neg : form \rightarrow form \\
& \quad \wedge : bool \rightarrow bool \rightarrow bool \\
& \quad \Leftrightarrow : bool \rightarrow bool \rightarrow bool \\
& \qquad = \lambda x.\lambda y.(x \Rightarrow y) \wedge (y \Rightarrow x) \\
& \}
\end{aligned}
$$

- $\Delta_1 = \mathcal{D}(PL, bool : \mathtt{type}), \mathcal{A}(PL, form : \mathtt{type}), \mathcal{A}(PL, \neg : form \rightarrow form)$

# Example Revisited

$Rev_1$

$$
\begin{aligned}
&PL = \{\\
&\quad bool : \mathtt{type}\\
&\quad \Rightarrow : bool \to bool \to bool\\
&\quad \wedge : bool \to bool \to bool\\
&\quad \Leftrightarrow : bool \to bool \to bool\\
&\qquad = \lambda x.\lambda y.(x \Rightarrow y) \wedge (y \Rightarrow x)\\
&\}
\end{aligned}
$$

$Rev_2$

$$
\begin{aligned}
&PL = \{\\
&\quad form : \mathtt{type}\\
&\quad \neg : form \to form\\
&\quad \wedge : bool \to bool \to bool\\
&\quad \Leftrightarrow : bool \to bool \to bool\\
&\qquad = \lambda x.\lambda y.(x \Rightarrow y) \wedge (y \Rightarrow x)\\
&\}
\end{aligned}
$$

- $\Delta_1 = \mathcal{D}(PL, bool : \mathtt{type}), \mathcal{A}(PL, form : \mathtt{type}), \mathcal{A}(PL, \neg : form \to form)$

# Example Revisited

$Rev_1$

$$
\begin{aligned}
&PL = \{\\
&\quad bool : \texttt{type}\\
&\quad \Rightarrow: bool \to bool \to bool\\
&\quad \wedge : bool \to bool \to bool\\
&\quad \Leftrightarrow: bool \to bool \to bool\\
&\qquad = \lambda x.\lambda y.(x \Rightarrow y) \wedge (y \Rightarrow x)\\
&\}
\end{aligned}
$$

$Rev_2$

$$
\begin{aligned}
&PL = \{\\
&\quad form : \texttt{type}\\
&\quad \neg : form \to form\\
&\quad \wedge : bool \to bool \to bool\\
&\quad \Leftrightarrow: bool \to bool \to bool\\
&\qquad = \lambda x.\lambda y.(x \Rightarrow y) \wedge (y \Rightarrow x)\\
&\}
\end{aligned}
$$

- $\Delta_1 = \mathcal{D}(PL, bool : \texttt{type}), \mathcal{A}(PL, form : \texttt{type}),$
  $\mathcal{A}(PL, \neg : form \to form)$
- $\Delta_2 = \mathcal{D}(PL, bool : \texttt{type}), \mathcal{A}(PL, form : \texttt{type}),$
  $\mathcal{D}(PL, \Rightarrow: bool \to bool \to bool), \mathcal{A}(PL, \neg : form \to form)$

# Example Revisited

$Rev_1$

$$
\begin{aligned}
&PL = \{\\
&\quad bool : \mathtt{type}\\
&\quad \Rightarrow : bool \to bool \to bool\\
&\quad \wedge : bool \to bool \to bool\\
&\quad \Leftrightarrow : bool \to bool \to bool\\
&\qquad = \lambda x.\lambda y.(x \Rightarrow y) \wedge (y \Rightarrow x)\\
&\}
\end{aligned}
$$

$Rev_2$

$$
\begin{aligned}
&PL = \{\\
&\quad form : \mathtt{type}\\
&\quad \neg : form \to form\\
&\quad \wedge : bool \to bool \to bool\\
&\quad \Leftrightarrow : bool \to bool \to bool\\
&\qquad = \lambda x.\lambda y.(x \Rightarrow y) \wedge (y \Rightarrow x)\\
&\}
\end{aligned}
$$

- $\Delta_1 = \mathcal{D}(PL, bool : \mathtt{type}), \mathcal{A}(PL, form : \mathtt{type}),$ $\mathcal{A}(PL, \neg : form \to form)$
- $\Delta_2 = \mathcal{D}(PL, bool : \mathtt{type}), \mathcal{A}(PL, form : \mathtt{type}),$ $\mathcal{D}(PL, \Rightarrow : bool \to bool \to bool), \mathcal{A}(PL, \neg : form \to form)$

# Example Revisited

$Rev_1$

$$
\begin{aligned}
&PL = \{\\
&\quad bool : \mathtt{type}\\
&\quad \Rightarrow : bool \to bool \to bool\\
&\quad \wedge : bool \to bool \to bool\\
&\quad \Leftrightarrow : bool \to bool \to bool\\
&\qquad = \lambda x.\lambda y.(x \Rightarrow y) \wedge (y \Rightarrow x)\\
&\}
\end{aligned}
$$

$Rev_2$

$$
\begin{aligned}
&PL = \{\\
&\quad form : \mathtt{type}\\
&\quad \neg : form \to form\\
&\quad \wedge : bool \to bool \to bool\\
&\quad \Leftrightarrow : bool \to bool \to bool\\
&\qquad = \lambda x.\lambda y.(x \Rightarrow y) \wedge (y \Rightarrow x)\\
&\}
\end{aligned}
$$

- $\Delta_1 = \mathcal{D}(PL, bool : \mathtt{type}), \mathcal{A}(PL, form : \mathtt{type}),$
  $\mathcal{A}(PL, \neg : form \to form)$
- $\Delta_2 = \mathcal{D}(PL, bool : \mathtt{type}), \mathcal{A}(PL, form : \mathtt{type}),$
  $\mathcal{D}(PL, \Rightarrow : bool \to bool \to bool), \mathcal{A}(PL, \neg : form \to form)$
- $\Delta_3 = \mathcal{R}(PL, bool, form),$
  $\mathcal{D}(PL, \Rightarrow : bool \to bool \to bool), \mathcal{A}(PL, \neg : form \to form)$

# Example Revisited

*Rev*$_1$

$$
\begin{aligned}
&PL = \{\\
&\quad bool : \mathtt{type}\\
&\quad \Rightarrow : bool \to bool \to bool\\
&\quad \wedge : bool \to bool \to bool\\
&\quad \Leftrightarrow : bool \to bool \to bool\\
&\qquad = \lambda x.\lambda y.(x \Rightarrow y) \wedge (y \Rightarrow x)\\
&\}
\end{aligned}
$$

*Rev*$_2$

$$
\begin{aligned}
&PL = \{\\
&\quad form : \mathtt{type}\\
&\quad \neg : form \to form\\
&\quad \wedge : bool \to bool \to bool\\
&\quad \Leftrightarrow : bool \to bool \to bool\\
&\qquad = \lambda x.\lambda y.(x \Rightarrow y) \wedge (y \Rightarrow x)\\
&\}
\end{aligned}
$$

- $\Delta_1 = \mathcal{D}(PL, bool : \mathtt{type}), \mathcal{A}(PL, form : \mathtt{type}), \mathcal{A}(PL, \neg : form \to form)$
- $\Delta_2 = \mathcal{D}(PL, bool : \mathtt{type}), \mathcal{A}(PL, form : \mathtt{type}), \mathcal{D}(PL, \Rightarrow : bool \to bool \to bool), \mathcal{A}(PL, \neg : form \to form)$
- $\Delta_3 = \mathcal{R}(PL, bool, form), \mathcal{D}(PL, \Rightarrow : bool \to bool \to bool), \mathcal{A}(PL, \neg : form \to form)$

# Example Revisited

$Rev_1$

$$
\begin{aligned}
& PL = \{ \\
& \quad bool : \mathtt{type} \\
& \quad \Rightarrow : bool \to bool \to bool \\
& \quad \wedge : bool \to bool \to bool \\
& \quad \Leftrightarrow : bool \to bool \to bool \\
& \qquad = \lambda x.\lambda y.(x \Rightarrow y) \wedge (y \Rightarrow x) \\
& \}
\end{aligned}
$$

$Rev_2$

$$
\begin{aligned}
& PL = \{ \\
& \quad form : \mathtt{type} \\
& \quad \neg : form \to form \\
& \quad \wedge : bool \to bool \to bool \\
& \quad \Leftrightarrow : bool \to bool \to bool \\
& \qquad = \lambda x.\lambda y.(x \Rightarrow y) \wedge (y \Rightarrow x) \\
& \}
\end{aligned}
$$

- $\Delta_1 = \mathcal{D}(PL, bool : \mathtt{type}), \mathcal{A}(PL, form : \mathtt{type}),$ $\mathcal{A}(PL, \neg : form \to form)$
- $\Delta_2 = \mathcal{D}(PL, bool : \mathtt{type}), \mathcal{A}(PL, form : \mathtt{type}),$ $\mathcal{D}(PL, \Rightarrow : bool \to bool \to bool), \mathcal{A}(PL, \neg : form \to form)$
- $\Delta_3 = \mathcal{R}(PL, bool, form),$ $\mathcal{D}(PL, \Rightarrow : bool \to bool \to bool), \mathcal{A}(PL, \neg : form \to form)$
- $\Delta_4 = \mathcal{D}(PL, bool : \mathtt{type}), \mathcal{D}(PL, x : \mathtt{type})$

# Example Revisited

*Rev*$_1$

$$
\begin{aligned}
& PL = \{ \\
& \quad bool : \mathtt{type} \\
& \quad \Rightarrow : bool \to bool \to bool \\
& \quad \wedge : bool \to bool \to bool \\
& \quad \Leftrightarrow : bool \to bool \to bool \\
& \qquad = \lambda x.\lambda y.(x \Rightarrow y) \wedge (y \Rightarrow x) \\
& \}
\end{aligned}
$$

*Rev*$_2$

$$
\begin{aligned}
& PL = \{ \\
& \quad form : \mathtt{type} \\
& \quad \neg : form \to form \\
& \quad \wedge : bool \to bool \to bool \\
& \quad \Leftrightarrow : bool \to bool \to bool \\
& \qquad = \lambda x.\lambda y.(x \Rightarrow y) \wedge (y \Rightarrow x) \\
& \}
\end{aligned}
$$

- $\Delta_1 = \mathcal{D}(PL, bool : \mathtt{type}), \mathcal{A}(PL, form : \mathtt{type}), \mathcal{A}(PL, \neg : form \to form)$
- $\Delta_2 = \mathcal{D}(PL, bool : \mathtt{type}), \mathcal{A}(PL, form : \mathtt{type}), \mathcal{D}(PL, \Rightarrow : bool \to bool \to bool), \mathcal{A}(PL, \neg : form \to form)$
- $\Delta_3 = \mathcal{R}(PL, bool, form), \mathcal{D}(PL, \Rightarrow : bool \to bool \to bool), \mathcal{A}(PL, \neg : form \to form)$
- $\Delta_4 = \mathcal{D}(PL, bool : \mathtt{type}), \mathcal{D}(PL, x : \mathtt{type})$

# Example Revisited

$Rev_1$

$$
\begin{aligned}
&PL = \{\\
&\quad bool : \mathtt{type}\\
&\quad \Rightarrow : bool \to bool \to bool\\
&\quad \wedge : bool \to bool \to bool\\
&\quad \Leftrightarrow : bool \to bool \to bool\\
&\qquad = \lambda x.\lambda y.(x \Rightarrow y) \wedge (y \Rightarrow x)\\
&\}
\end{aligned}
$$

$Rev_2$

$$
\begin{aligned}
&PL = \{\\
&\quad form : \mathtt{type}\\
&\quad \neg : form \to form\\
&\quad \wedge : bool \to bool \to bool\\
&\quad \Leftrightarrow : bool \to bool \to bool\\
&\qquad = \lambda x.\lambda y.(x \Rightarrow y) \wedge (y \Rightarrow x)\\
&\}
\end{aligned}
$$

- $\Delta_1 = \mathcal{D}(PL, bool : \mathtt{type}), \mathcal{A}(PL, form : \mathtt{type}),$
  $\mathcal{A}(PL, \neg : form \to form)$
- $\Delta_2 = \mathcal{D}(PL, bool : \mathtt{type}), \mathcal{A}(PL, form : \mathtt{type}),$
  $\mathcal{D}(PL, \Rightarrow : bool \to bool \to bool), \mathcal{A}(PL, \neg : form \to form)$
- $\Delta_3 = \mathcal{R}(PL, bool, form),$
  $\mathcal{D}(PL, \Rightarrow : bool \to bool \to bool), \mathcal{A}(PL, \neg : form \to form)$
- ~~$\Delta_4 = \mathcal{D}(PL, bool : \mathtt{type}), \mathcal{D}(PL, x : \mathtt{type})$~~

# Example Revisited

$Rev_1$

$$
\begin{aligned}
&PL = \{ \\
&\quad bool : \mathtt{type} \\
&\quad \Rightarrow : bool \to bool \to bool \\
&\quad \wedge : bool \to bool \to bool \\
&\quad \Leftrightarrow : bool \to bool \to bool \\
&\qquad = \lambda x.\lambda y.(x \Rightarrow y) \wedge (y \Rightarrow x) \\
&\}
\end{aligned}
$$

$Rev_2$

$$
\begin{aligned}
&PL = \{ \\
&\quad form : \mathtt{type} \\
&\quad \neg : form \to form \\
&\quad \wedge : bool \to bool \to bool \\
&\quad \Leftrightarrow : bool \to bool \to bool \\
&\qquad = \lambda x.\lambda y.(x \Rightarrow y) \wedge (y \Rightarrow x) \\
&\}
\end{aligned}
$$

- $\Delta_1 = \mathcal{D}(PL, bool : \mathtt{type}), \mathcal{A}(PL, form : \mathtt{type}), \mathcal{A}(PL, \neg : form \to form)$
- $\Delta_2 = \mathcal{D}(PL, bool : \mathtt{type}), \mathcal{A}(PL, form : \mathtt{type}), \mathcal{D}(PL, \Rightarrow : bool \to bool \to bool), \mathcal{A}(PL, \neg : form \to form)$
- $\Delta_3 = \mathcal{R}(PL, bool, form), \mathcal{D}(PL, \Rightarrow : bool \to bool \to bool), \mathcal{A}(PL, \neg : form \to form)$
- ~~$\Delta_4 = \mathcal{D}(PL, bool : \mathtt{type}), \mathcal{D}(PL, x : \mathtt{type})$~~

# Semantic Differencing: Formal Properties

- change detection ($\mathcal{G}' - \mathcal{G}$)

  identify differences between two theory graphs

- change application ($\mathcal{G} \ll \Delta$)

  apply changes to a theory graph

# Semantic Differencing: Formal Properties

- change detection ($\mathcal{G}' - \mathcal{G}$)

  identify differences between two theory graphs

- change application ($\mathcal{G} \ll \Delta$)

  apply changes to a theory graph

- $\mathcal{G}$-applicability

  $\Delta_1$ applicable to $\mathcal{G}$ iff it can be applied to $\mathcal{G}$

# Semantic Differencing: Formal Properties

- change detection $(\mathcal{G}' - \mathcal{G})$

  identify differences between two theory graphs

- change application $(\mathcal{G} \ll \Delta)$

  apply changes to a theory graph

- $\mathcal{G}$-applicability

  $\Delta_1$ applicable to $\mathcal{G}$ iff it can be applied to $\mathcal{G}$

- $\mathcal{G}$-equivalence $(\equiv_{\mathcal{G}})$

  $\Delta_1 \equiv_{\mathcal{G}} \Delta_2$ iff $\mathcal{G} \ll \Delta_1 = \mathcal{G} \ll \Delta_2$

# Semantic Differencing: Formal Properties

- change detection $(\mathcal{G}' - \mathcal{G})$

  identify differences between two theory graphs

- change application $(\mathcal{G} \ll \Delta)$

  apply changes to a theory graph

- $\mathcal{G}$-applicability

  $\Delta_1$ applicable to $\mathcal{G}$ iff it can be applied to $\mathcal{G}$

- $\mathcal{G}$-equivalence $(\equiv_{\mathcal{G}})$

  $\Delta_1 \equiv_{\mathcal{G}} \Delta_2$ iff $\mathcal{G} \ll \Delta_1 = \mathcal{G} \ll \Delta_2$

- normal diffs

  minimal representatives w.r.t. $\equiv_{\mathcal{G}}$

# Semantic Differencing: Formal Properties

- change detection $(\mathcal{G}' - \mathcal{G})$

  identify differences between two theory graphs

- change application $(\mathcal{G} \ll \Delta)$

  apply changes to a theory graph

- $\mathcal{G}$-applicability

  $\Delta_1$ applicable to $\mathcal{G}$ iff it <u>can</u> be applied to $\mathcal{G}$

- $\mathcal{G}$-equivalence $(\equiv_{\mathcal{G}})$

  $\Delta_1 \equiv_{\mathcal{G}} \Delta_2$ iff $\mathcal{G} \ll \Delta_1 = \mathcal{G} \ll \Delta_2$

- normal diffs

  <u>minimal</u> representatives w.r.t. $\equiv_{\mathcal{G}}$

- inversability of diffs

  $\mathcal{G} \ll \Delta \ll \Delta^{-1} = \mathcal{G}$

# Semantic Differencing: Implementation

## Change Detection ($\mathcal{G}' - \mathcal{G}$)

- view theory graphs as (nested) URI-indexed tables of declarations.
- new URIs $\rightarrow$ adds, old URIs $\rightarrow$ deletes, preserved URIs $\rightarrow$ (if changed) updates.
- refine the resulting diff by replacing add-delete pairs that represent a rename with the corresponding rename

# Semantic Differencing: Implementation

## Change Detection ($\mathcal{G}' - \mathcal{G}$)

- view theory graphs as (nested) URI-indexed tables of declarations.
- new URIs $\rightarrow$ adds, old URIs $\rightarrow$ deletes, preserved URIs $\rightarrow$ (if changed) updates.
- refine the resulting diff by replacing add-delete pairs that represent a rename with the corresponding rename

## Change Application ($\mathcal{G} \ll \Delta$)

- follow the intuitive semantics of each change
- apply (in order) the changes from $\Delta$ to $\mathcal{G}$ (if $\mathcal{G}$-applicable)

# Fine-grained dependencies

- in MMT, validation units are individual components (types and definiens)
- we distinguish two types of dependencies
  - syntactic dependencies
    - declaration level
    - foundation-independent
    - occurs-in relation
  - semantic dependencies
    - component level
    - foundation-dependent
    - trace lookups during foundational validation

# Fine-grained dependencies

- in MMT, validation units are individual components (types and definiens)
- we distinguish two types of dependencies
  - syntactic dependencies
    - declaration level
    - foundation-independent
    - occurs-in relation
  - semantic dependencies
    - component level
    - foundation-dependent
    - trace lookups during foundational validation
- dependencies are indexed by MMT and are available at any time

# Example Revisited - Again

$Rev_1$

$PL = \{$

$\quad bool : \texttt{type}$

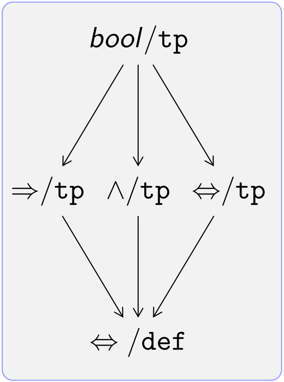
$\quad \Rightarrow : bool \rightarrow bool \rightarrow bool$

$\quad \wedge : bool \rightarrow bool \rightarrow bool$

$\quad \Leftrightarrow : bool \rightarrow bool \rightarrow bool$

$\qquad = \lambda x.\lambda y.(x \Rightarrow y) \wedge (y \Rightarrow x)$

$\}$



# Impact Propagation

- key idea : propagation as diff enrichment process
- impact propagation of a diff $\Delta$ is another diff $\overline{\Delta}$ that :
  - marks impacted components

    by surrounding with OpenMath error terms
  - automatically propagates renames

    updates in-term references

# Impact Propagation

- key idea : propagation as diff enrichment process
- impact propagation of a diff $\Delta$ is another diff $\overline{\Delta}$ that :
  - marks impacted components

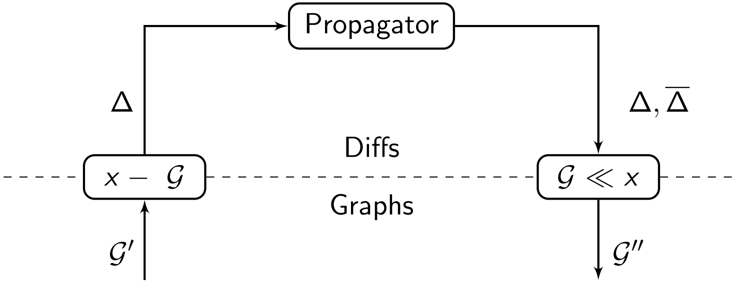
    by surrounding with OpenMath error terms
  - automatically propagates renames

    updates in-term references

**Theorem**

*After all error terms are replaced with valid terms in $\mathcal{G} \ll \Delta \ll \overline{\Delta}$, the resulting theory graph is valid.*

# Workflow Example (relative to a graph $\mathcal{G}$)



# Example Revisited - Yet Again

$$\overline{\Delta} = \mathcal{U}(PL, \Leftrightarrow, \mathtt{def}, \lambda x.\lambda y.(x \Rightarrow y) \wedge (y \Rightarrow x), \boxed{\lambda x.\lambda y.(x \Rightarrow y) \wedge (y \Rightarrow x)}),$$
$$\mathcal{U}(PL, \wedge, \mathtt{tp}, bool \to bool \to bool, form \to form \to form),$$
$$\mathcal{U}(PL, \Leftrightarrow, \mathtt{tp}, bool \to bool \to bool, form \to form \to form)$$

$$
\begin{aligned}
PL = \{ & \\
& form : \mathtt{type} \\
& \neg : form \to form \\
& \wedge : form \to form \to form \\
& \Leftrightarrow : form \to form \to form \\
& \quad = \boxed{\lambda x.\lambda y.(x \Rightarrow y) \wedge (y \Rightarrow x))} \\
\} &
\end{aligned}
$$

# Evaluation : LATIN

| Dependencies | Components (%) |
|---|---|
| 0 – 5 | 1373 (79) |
| 6 – 10 | 271 (15.6) |
| 11 – 15 | 81 (4.7) |
| 16 – 26 | 13 (0.7) |

| Impacts | Components (%) |
|---|---|
| 0 – 5 | 1504 (86.5) |
| 6 – 10 | 101 (5.8) |
| 11 – 25 | 76 (4.4) |
| 26 – 50 | 31 (1.8) |
| 50 – 449 | 26 (1.5) |

- generally low number of impacts

  due to modularity

- however, high variance of impacts

  creates need for detection tools

- on average, types have 3 times more impacts than definiens

  validates our fine-grained approach

# Observations

- at all steps renames ($\mathcal{R}$) require special treatment
  - good at the user level

    more change types means more change semantics
  - bad at the system level

    more change types means more complex formalization

# Observations

- at all steps renames ($\mathcal{R}$) require special treatment
  - good at the user level

    more change types means more change semantics
  - bad at the system level

    more change types means more complex formalization
- underlying problem: conflicting requirements for the change language

  simple vs expressive

# Observations

- at all steps renames ($\mathcal{R}$) require special treatment
  - good at the user level

    more change types means more change semantics
  - bad at the system level

    more change types means more complex formalization
- underlying problem: conflicting requirements for the change language

  simple vs expressive
- solution : extensibility
  - regular and minimal core language

    add, delete, update
  - enlarge expressivity through refinements

    rename

# Observations

- at all steps renames ($\mathcal{R}$) require special treatment
  - good at the user level

    more change types means more change semantics
  - bad at the system level

    more change types means more complex formalization
- underlying problem: conflicting requirements for the change language

  simple vs expressive
- solution : extensibility
  - regular and minimal core language

    add, delete, update
  - enlarge expressivity through refinements

    rename, merge

# Observations

- at all steps renames ($\mathcal{R}$) require special treatment
  - good at the user level

    more change types means more change semantics
  - bad at the system level

    more change types means more complex formalization
- underlying problem: conflicting requirements for the change language

  simple vs expressive
- solution : <u>extensibility</u>
  - regular and minimal core language

    add, delete, update
  - enlarge expressivity through refinements

    rename, merge, split

# Observations

- at all steps renames ($\mathcal{R}$) require special treatment
  - good at the user level

    more change types means more change semantics
  - bad at the system level

    more change types means more complex formalization
- underlying problem: conflicting requirements for the change language

  simple vs expressive
- solution : <u>extensibility</u>
  - regular and minimal core language

    add, delete, update
  - enlarge expressivity through refinements

    rename, merge, split, alpha-renaming

# Observations

- at all steps renames ($\mathcal{R}$) require special treatment
  - good at the user level

    more change types means more change semantics
  - bad at the system level

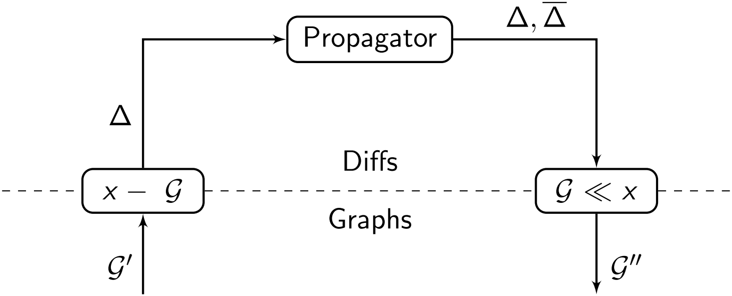
    more change types means more complex formalization
- underlying problem: conflicting requirements for the change language

  simple vs expressive
- solution : <u>extensibility</u>
  - regular and minimal core language

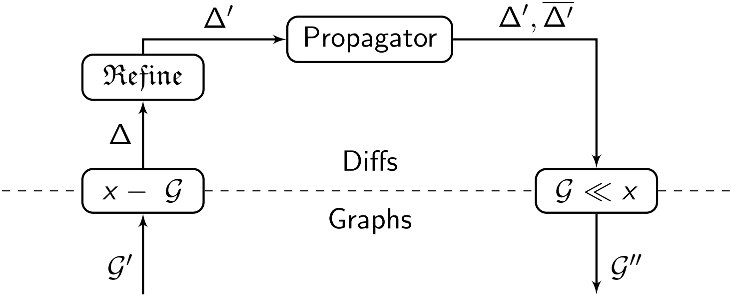
    add, delete, update
  - enlarge expressivity through refinements

    rename, merge, split, alpha-renaming, . . .

# Workflow Example (relative to a graph $\mathcal{G}$) – Again



# Workflow Example (relative to a graph $\mathcal{G}$) – Better



# Conclusion and Future Work

- MMT MoC : a change management solution for MMT
  - formal definition, theorems
  - supports transactions and roll-backs
  - uses fine-grained semantic dependencies
  - implemented in the MMT API
- future work (currently in progress)
  - refinement (add flexibility to the change language)

    towards an MMT theory of refactoring
  - integration with user interfaces