Bidirectionally tolerating inconsistency: partial transformations

Perdita Stevens

University of Edinburgh

May 2014

Plan

- Brief intro to model-driven development (MDD) and its current deficiencies
- Brief intro to bidirectional transformations (bx)
- Why tolerate inconsistency? Formal framework.
- Examples
- Properties
- Examples
- Relating partial bx

Plan

- Brief intro to model-driven development (MDD) and its current deficiencies
- Brief intro to bidirectional transformations (bx)
- Why tolerate inconsistency? Formal framework.
- Examples
- Properties
- Examples
- Relating partial bx

Model-driven development

Defn: A model is an abstract, usually graphical, representation of some aspect of a system.

Defn: MDD is software development in which models are important.

Motivation: Manage information overload, by separating concerns and providing representations suitable for each set of decisions.

Current deficiencies of MDD

Despite many successes and ongoing popularity, MDD:

1. does not support agile development well enough
2. is not supported by good enough tools
3. especially, lacks tools that offer easy-to-use guarantees.

Poor support for bidirectional transformations (bx) is a big part of the problem.

The prize, if we solve these problems

MDD could shift the paradigm of software development.

The prize, if we solve these problems

MDD could shift the paradigm of software development.

The prize, if we solve these problems

MDD could shift the paradigm of software development.

Imagine a particular user's requirements represented as a model they can understand and change... and the software automatically updating.

Note that "modelling as higher-level programming" is not sufficient.

Plan

- Brief intro to model-driven development (MDD) and its current deficiencies
- Brief intro to bidirectional transformations (bx)
- Why tolerate inconsistency? Formal framework.
- Examples
- Properties
- Examples
- Relating partial bx

Why bx?

To manage information overload, different experts work with models that show only the things they need to know and decide.

E.g.

- architecture
- database schema
- security concerns
- requirements of one kind of user...

Expertise in *their* model: maybe even only in *the language of* their model – e.g., RDBMS vs UML, an ADL vs UML...

Decisions can have knock-on effects. Ideally, effected without endless meetings...

Concretely: bidirectional transformations

A bidirectional transformation (bx) has two, related, jobs:

1. check whether given models are consistent
2. if not, change one of them to restore consistency, on the assumption that the others are authoritative and must not be changed.

Ideally you don't want separate programs doing these jobs – they'd change together and duplicate lots of information.

Motivates bx languages.

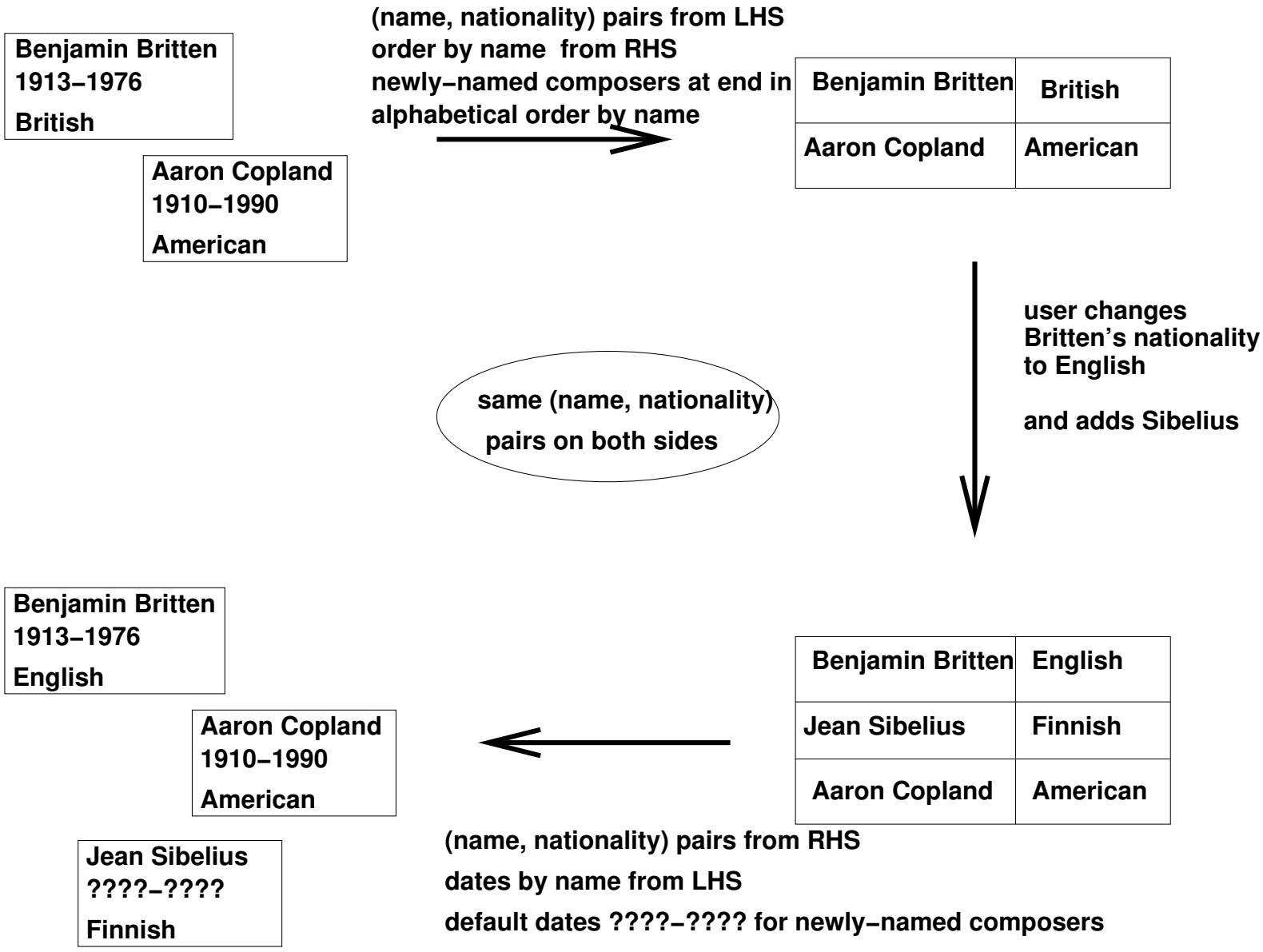
State-based bx

In this talk we are only dealing with state-based bx: that is, we assume the only information available to the bx is the current versions of the models it works on.

- no trace information relating subparts of models;
- no record of changes made;
- no bx-awareness in the user's modelling tool;
- (potentially) fully automatic; the bx programmer can encode the choice between consistent models.

Motivated by, e.g., wanting to allow users to continue to use their favourite tools – in practice very important. But other choices are possible.

Composers example



http://bx-community.wikidot.com/examples:home

Ordinary total bx

Sets of *models* M and N.

$R : M \leftrightarrow N$ comprising:

- Consistency relation $R \subseteq M \times N$
- Forward consistency restorer $\overrightarrow{R} : M \times N \to N$
- Backward consistency restorer $\overleftarrow{R} : M \times N \to M$

satisfying (at least):

- Correctness: $R(m, \overrightarrow{R}(m, n))$ and vv
- Hippocraticness: $R(m, n) \Rightarrow \overrightarrow{R}(m, n) = n$ and vv

Plan

- Brief intro to model-driven development (MDD) and its current deficiencies
- Brief intro to bidirectional transformations (bx)
- Why tolerate inconsistency? Formal framework
- Examples
- Properties
- Examples
- Relating partial bx

Tolerating inconsistency: in SE

Experience across SE shows that real development is messy: the artefacts involved (specifications, designs, code, tests...) do not stay tidy and improve in lockstep.

A bx that guarantees to restore perfect consistency will require preconditions: e.g., input models at least conform to their metamodels...

So we can't use the bx at all till that's true? Bit useless.

Want it to do *what it can* to save us effort anyway.

Tolerating inconsistency: specifically bx

In practice, bx engines do sometimes fail to restore consistency.

Worse: ModelMorf (for example) also sometimes "succeeds" in enforce mode, but produces a model that fails check (is not correct, in our terminology: and provides no alternative guarantee).

If you're relying on a bx engine to synchronise your large models, this could be annoying – or catastrophic.

We'd like clear, reliable guarantees from our bx.

Reasons for not restoring perfect consistency

Suppose we have models m, n that are not currently consistent according to bx R.

Why might a tool fail to find n' to replace n such that $R(m, n')$ holds?

Reasons for not restoring perfect consistency

Suppose we have models m, n that are not currently consistent according to bx R.

Why might a tool fail to find n' to replace n such that $R(m, n')$ holds?

- Maybe there isn't one

Reasons for not restoring perfect consistency

Suppose we have models m, n that are not currently consistent according to bx R.

Why might a tool fail to find n' to replace n such that $R(m, n')$ holds?

- Maybe there isn't one
- Maybe there are several, and the tool doesn't know which is best.

Reasons for not restoring perfect consistency

Suppose we have models m, n that are not currently consistent according to bx R.

Why might a tool fail to find n' to replace n such that $R(m, n')$ holds?

- Maybe there isn't one
- Maybe there are several, and the tool doesn't know which is best.
- Maybe this tool can't, or doesn't have permission to, make the necessary change.

Back up: Who sees partiality?

The user may see a bx that doesn't always restore perfect consistency (cf file synchroniser, VCS)

- we want to offer worthwhile guarantees to the user, nonetheless

Or partial bx may be for the model transformation developer: build up a total bx from partial bx (cf add/delete phases in QVT-R)

- we want to reason from properties of the phases to properties of the composed bx.

In either case, we need to reason about properties of partial bx.

Partial bx: A new framework

Sets of *models* M and N and a consistency partial order Λ.

$R : M \leftrightarrow N$ comprising:

- Consistency indicator $R : M \times N \to \Lambda$ that says how consistent a given pair of models is
- Forward consistency restorer $\overrightarrow{R} : M \times N \to N$
- Backward consistency restorer $\overleftarrow{R} : M \times N \to M$

Note: we haven't imposed correctness, so the consistency restorers don't necessarily restore consistency!

But in the worst case, each has an argument it can return, so they are still total functions.

Why "partial"?

Could just use earlier framework, making consistency restoration functions partial:

$\overrightarrow{R} : M \times N \dot{\to} N$

But if $(m, n) \notin \mathrm{dom}(\overrightarrow{R})$, we expect in practice that forward consistency restoration leaves n unchanged – which we can model as it returning n.

So using total functions:

$\overrightarrow{R} : M \times N \to N$

we can model "do nothing", "restore perfect consistency" and everything in between.

These bx are partial in the sense that they may only partially work!

What properties should a partial bx satisfy?

Maybe: Correctness and hippocraticness

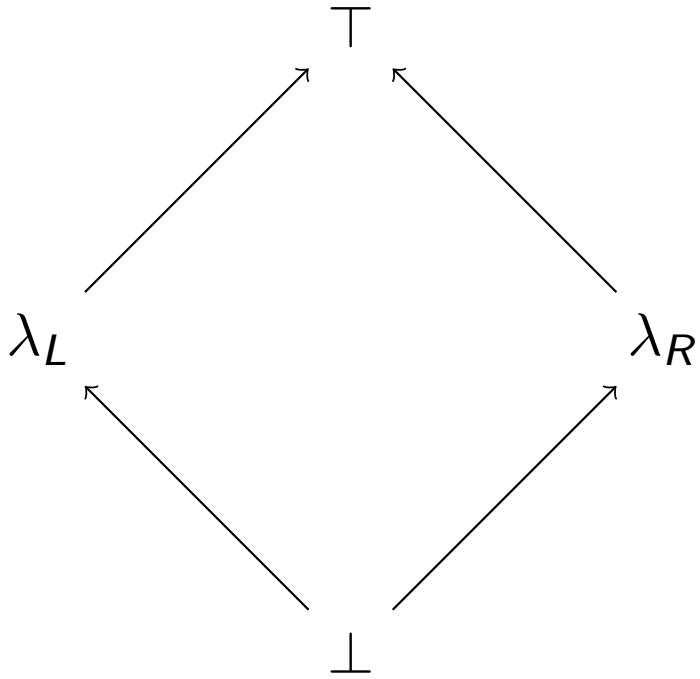
If our consistency structure Λ has a $\top$ meaning "perfectly consistent" we can still talk about correctness and hippocraticness – just replace "$R(m,n)$" by "$R(m,n) = \top$".

- Correctness: $R(m, \overrightarrow{R}(m,n)) = \top$ and vv
  – likely too strong
  – forces R to be consistency-total.
- Hippocraticness: $(R(m,n) = \top) \Rightarrow \overrightarrow{R}(m,n) = n$ and vv
  – likely too weak
  – says nothing about what happens if (m,n) are not perfectly consistent.

Plan

- Brief intro to model-driven development (MDD) and its current deficiencies
- Brief intro to bidirectional transformations (bx)
- Why tolerate inconsistency? Formal framework
- Examples
- Properties
- Examples
- Relating partial bx

A few consistency partial orders



Consistency as a conjunction

Suppose we say $R(m, n) =:$

- λ_L if every composer in m is matched in n;
- λ_R if every composer in n is matched in m;
- $\top$ if both, $\bot$ if neither.

Happier about elements being added than deleted? Make $\overrightarrow{R}$ add to n any missing composers – thus ensuring that consistency reaches at least λ_L – but not delete any composers who do not occur in m. (And vv.)

Making uncontroversial changes

When a new composer is added to n, and $\overleftarrow{R}$ is used to modify m accordingly, it must add a new composer to m, which involves choosing a value for the dates attribute.

User might prefer bx not to add new composers, but just to tidy up by deleting unnecessary ones.

New composer still has to be added manually, but some work has been automated, and the consistency indicator can provide helpful information about the current consistency state (λ_L in this case).

Consistency as diagnostic

We may wish to know just how inconsistent a pair of models is.

May be useful for bx to tell us this, even when its restoration functions are those of a total bx.

E.g. use the natural numbers as consistency levels to indicate the number of errors counted in some way. In our example, this could be the number of composers who occur on one side but not the other.

(Relationship of this to the original total bx? In a moment...)

A non-consistency-total example

Suppose we add condition that composer names in a table in $n \in N$ should be no more than 16 characters long (those in m unrestricted).

λ_L means both models comply with their constraints
λ_R means the same (name, nationality) pairs occur in both models
$\bot$ means neither holds
$\top$ means both hold.

Consider $m \in M$ that includes a Pyotr Ilyich Tchaikovsky (only).

There is no model $n \in N$ which is perfectly consistent with m.

A bx that is asked to modify n must choose: either it can include this name, which will allow it to achieve consistency level λ_R, or it can comply with its constraints, achieving λ_L, but it cannot do both.

Plan

- Brief intro to model-driven development (MDD) and its current deficiencies
- Brief intro to bidirectional transformations (bx)
- Why tolerate inconsistency? Formal framework
- Examples
- Properties
- Examples
- Relating partial bx

Properties of partial bx: stopping bad behaviour

"Don't make matters worse"

A bx R is *improving* if it always returns a model that is at least as consistent as its argument was. That is,
$R(m, \overrightarrow{R}(m,n)) \geq_\Lambda R(m,n)$, and dually.

Better:

"If you can't improve matters, do nothing"

A bx R is *as hippocratic as possible* (AHAP) if its consistency restoration functions return exactly their argument of appropriate type, unless returning something strictly more consistent. That is,

$$\overrightarrow{R}(m,n) = n \ \vee \ R(m, \overrightarrow{R}(m,n)) >_\Lambda R(m,n)$$

and dually.

Properties of partial bx: requiring good behaviour

"Do the best you can"

Given $m \in M$, the set of $\overrightarrow{R}$ *candidates with respect to* m is $\{n' \in N : R(m, n') \text{ is maximal}\}$. That is, n' is a candidate iff $R(m, n') = \lambda \in \Lambda$ such that there does not exist any $n'' \in N$ with $R(m, n'') >_\Lambda \lambda$.

A bx R is *as correct as possible* (ACAP) if it always returns a candidate.

Plan

- Brief intro to model-driven development (MDD) and its current deficiencies
- Brief intro to bidirectional transformations (bx)
- Why tolerate inconsistency? Formal framework
- Examples
- Properties
- Examples
- Relating partial bx

Consistency as a conjunction

Suppose we say $R(m, n) =:$

- λ_L if every composer in m is matched in n;
- λ_R if every composer in n is matched in m;
- $\top$ if both, $\bot$ if neither.

Happier about elements being added than deleted? Make $\overrightarrow{R}$ add to n any missing composers – thus ensuring that consistency reaches at least λ_L – but not delete any composers who do not occur in m. (And vv.)

Improving, AHAP, but not correct or ACAP.

Making uncontroversial changes

When a new composer is added to n, and $\overleftarrow{R}$ is used to modify m accordingly, it must add a new composer to m, which involves choosing a value for the dates attribute.

User might prefer bx not to add new composers, but just to tidy up by deleting unnecessary ones.

New composer still has to be added manually, but some work has been automated, and the consistency indicator can provide helpful information about the current consistency state (λ_L in this case).

Again, such a bx will be improving and AHAP, but not correct or ACAP.

Inconvenient truth

Just because n is a candidate wrt m, it does not follow that m is a candidate wrt n.

(If this is always the case, we call the bx balanced, and it has nice properties. If the consistency relation is total, the bx is balanced. Unfortunately, this is morally the only kind of balanced bx!)

A non-consistency-total example

Suppose we add condition that composer names in a table in $n \in N$ should be no more than 16 characters long (those in m unrestricted).

λ_L means both models comply with their constraints
λ_R means the same (name, nationality) pairs occur in both models
$\perp$ means neither holds
$\top$ means both hold.

Consider $m \in M$ that includes a Pyotr Ilyich Tchaikovsky (only).

There is no model $n \in N$ which is perfectly consistent with m.

A bx that is asked to modify n must choose: either it can include this name, which will allow it to achieve consistency level λ_R, or it can comply with its constraints, achieving λ_L, but it cannot do both.

Improving, AHAP, ACAP, not correct, not balanced.

Immediate consequences of definitions

1. If R is correct, then it is consistency-total and ACAP.
2. If R is AHAP, then it is hippocratic.
3. If R is correct, then it is hippocratic if and only if it is AHAP.
4. If R is consistency-total, then it is correct if and only if it is ACAP.
5. If R is consistency-total then R is balanced; the candidates and the dominant candidates (wrt m) are both just the set of perfectly-consistent elements (wrt m).
6. If R is either AHAP or ACAP, then it is improving.

Plan

- Brief intro to model-driven development (MDD) and its current deficiencies
- Brief intro to bidirectional transformations (bx)
- Why tolerate inconsistency? Formal framework
- Examples
- Properties
- Examples
- Relating partial bx

Homorphisms of partial bx

Let $R_1 : M_1 \leftrightarrow N_1$ be a partial bx on Λ_1.

Let $f_\Lambda : \Lambda_1 \to \Lambda_2$ be a total function preserving the partial order, that is, satisfying $x \leq y \Rightarrow f_\Lambda x \leq f_\Lambda y$.

Let $f_M : M_1 \to M_2$ and $f_N : N_1 \to N_2$ be surjective partial functions satisfying the following coherence condition: if $f_M m = f_M m'$ and $f_N n = f_N n'$ then

- $f_\Lambda R_1(m, n) = f_\Lambda R_1(m', n')$;
- f_N is defined on $\overrightarrow{R_1}(m, n)$, and dually;
- $f_N \overrightarrow{R_1}(m, n) = f_N \overrightarrow{R_1}(m', n')$, and dually.

Then the following gives a well-defined partial bx $f(R_1) = R_2 : M_2 \leftrightarrow N_2$ on Λ_2:

- $R_2(f_M m, f_N n) = f_\Lambda R_1(m, n)$
- $\overrightarrow{R_2}(f_M m, f_N n) = f_N \overrightarrow{R_1}(m, n)$ and dually.

Bx differing only in consistency structure

In particular, notice that if $M_1 = M_2$ and $N_1 = N_2$, with f_M and f_N being total identity functions, the coherence conditions become trivial. That is, given a bx $R : M \leftrightarrow N$ over Λ_1, and a partial order preserving function $f_\Lambda : \Lambda_1 \to \Lambda_2$, we can always build a bx $f(R) : M \leftrightarrow N$ over Λ_2.

E.g. map between ordinary total bx and a version that gives extra diagnostic information.

About the properties

Assuming coherence conditions satisfied:

- If R_1 is improving then so is $f(R_1)$.

About the properties

Assuming coherence conditions satisfied:

- If R_1 is improving then so is $f(R_1)$.
- If R_1 is ACAP and f_Λ satisfies $f_\Lambda x > f_\Lambda y \Rightarrow x > y$, then $f(R_1)$ is ACAP.
  (E.g. turning incomparability into dominance in the consistency structure may break ACAP.)

About the properties

Assuming coherence conditions satisfied:

- If R_1 is improving then so is $f(R_1)$.
- If R_1 is ACAP and f_Λ satisfies $f_\Lambda x > f_\Lambda y \Rightarrow x > y$, then $f(R_1)$ is ACAP.
  (E.g. turning incomparability into dominance in the consistency structure may break ACAP.)
- If R_1 is AHAP and f_Λ satisfies $x > y \Rightarrow f_\Lambda x > f_\Lambda y$, then $f(R_1)$ is AHAP.
  (Collapsing comparisons may break AHAP.)

Restriction to subspace pair

Let (M_2, N_2) be a subspace pair in (M_1, N_1) – that is, if users stay inside (M_2, N_2), bx won't move us outside.

Let f_M, f_N be the identity on M_2, N_2, undefined elsewhere.

Then as these are also injective where defined, the coherence conditions hold for any f_Λ preserving the partial order.

If we take f_Λ to be the identity, what we get is just the restriction of a bx to a subspace pair.

Restriction will be ACAP and AHAP if the original was.

Total bx from partial bx

Construct a total bx from any partial bx by throwing out all the elements on which consistency cannot be restored.

Let $R : M \leftrightarrow N$ be a partial bx which is ACAP and AHAP. Define $R_\top : M_\top \leftrightarrow N_\top$ by:

- $M_\top = \{m \in M : \exists n \in N. R(m, n) = \top\}$, and $N_\top$ dually;
- $R_\top(m, n)$ holds iff $R(m, n) = \top$;
- $\overrightarrow{R_\top}(m, n) = \overrightarrow{R}(m, n)$ and dually.

Then $R_\top$ is a correct and hippocratic total bx.

Total bx from partial bx

Construct a total bx from any partial bx by throwing out all the elements on which consistency cannot be restored.

Let $R : M \leftrightarrow N$ be a partial bx which is ACAP and AHAP. Define $R_\top : M_\top \leftrightarrow N_\top$ by:

- $M_\top = \{m \in M : \exists n \in N. R(m, n) = \top\}$, and $N_\top$ dually;
- $R_\top(m, n)$ holds iff $R(m, n) = \top$;
- $\overrightarrow{R_\top}(m, n) = \overrightarrow{R}(m, n)$ and dually.

Then $R_\top$ is a correct and hippocratic total bx.

(duh)

Total bx from partial bx

Construct a total bx from any partial bx by throwing out all the elements on which consistency cannot be restored.

Let $R : M \leftrightarrow N$ be a partial bx which is ACAP and AHAP. Define $R_\top : M_\top \leftrightarrow N_\top$ by:

- $M_\top = \{m \in M : \exists n \in N . R(m, n) = \top\}$, and $N_\top$ dually;
- $R_\top(m, n)$ holds iff $R(m, n) = \top$;
- $\overrightarrow{R_\top}(m, n) = \overrightarrow{R}(m, n)$ and dually.

Then $R_\top$ is a correct and hippocratic total bx.

(duh)

In development, though, may be very useful: want your bx to cope gracefully with early stage "bad" models that can't be made completely consistent, and then want "the same" bx to give stronger guarantees once you reach "good" models.

Conclusions

There are good reasons for being able to talk about bx in a formal way, even if they may fail to restore consistency.

We offered a basic framework in which to do this, some properties with which to talk about partial bx, and a way to relate bx.

Secret motivation: use this to understand witness structures and Least Change.

Questions?

↙ Coders

Images: Wikipedia; `http://www.germes-online.com`

Extra slides

The trouble with balanced

Morally, consistency relation total is the only circumstance when a bx can be balanced. For,

- given m, let λ_m be the maximum consistency level achievable together with m, i.e. max over n of $R(m, n)$. [Let's suppose all is finite, I doubt it matters.] Similarly λ_n.
- If n is a candidate wrt m, we must have $R(m, n) = \lambda_m$.
- But if m is a candidate wrt n, we must have $R(m, n) = \lambda_n$!
- So if R is balanced, then for all m, n, $\lambda_m = \lambda_n$
- Um, we may as well call this common λ $\top$, mayn't we? It's the highest consistency level ever achievable by any pair, so if the po has something not below this, who cares anyway.

Development discipline

The discipline for using a partial bx manually must be considered carefully.

E.g. suppose a "non-deleting" $\overrightarrow{R}$ does not restore perfect consistency (because there is a composer in n but not m, say).

Now if a "non-deleting" $\overleftarrow{R}$ is immediately applied – without the extra composer being manually deleted – this composer will be re-added to m.

Could actually model the disciplined use as a bx comprising an automatic non-deleting phase followed by a manual reconciliation phase...