Retrenchments, Refinements, and the Mondex Electronic Purse

R. Banach,
School of Computer Science, University of Manchester, UK

Contents:

1. Model oriented refinement: Pros and Cons.

In model oriented refinement, we build models of the system, by specifying:

- the state (and I/O) space of a model
- the operations (or events) of a model: via eg.
  transition systems,
  programming notations,
  predicate transformers,
  etc.

1. Model oriented refinement: Pros and Cons.

In model oriented refinement, we build models of the system, by specifying:

- the state (and I/O) space of a model
- the operations (or events) of a model: via eg.
  transition systems,
  programming notations,
  predicate transformers,
  etc.

Models can then be related pairwise by REFINEMENT. This usually involves a notion of *substitutivity*, of some **concrete** system behaviours for some **abstract** system behaviours, intended to help move closer to an implementation, and leading to *sufficient conditions* for refinement.

To every concrete behaviour there is at least one abstract behaviour which is correct.

Refinement Success Stories

There are various detailed theoretical variations on model oriented refinement, and various specific languages and implementations. The implementations of refinement have had some outstanding successes in the construction of dependable industrial scale systems of high criticality.

Refinement Success Stories

There are various detailed theoretical variations on model oriented refinement, and various specific languages and implementations. The implementations of refinement have had some outstanding successes in the construction of dependable industrial scale systems of high criticality.

Some languages: Z, VDM, B, RAISE, ASM.

Some key projects:

- Mondex Purse, Multos OS (Z)
- CDIS etc. (VDM)
- MÉTÉOR (and many more French and other railway systems) (B)
- Prolog, C, Java (and many more) language definitions (ASM)

Refinement Success Stories

There are various detailed theoretical variations on model oriented refinement, and various specific languages and implementations. The implementations of refinement have had some outstanding successes in the construction of dependable industrial scale systems of high criticality.

Some languages: Z, VDM, B, RAISE, ASM.

Some key projects:

- Mondex Purse, Multos OS (Z)
- CDIS etc. (VDM)
- MÉTÉOR (and many more French and other railway systems) (B)
- Prolog, C, Java (and many more) language definitions (ASM)

What's up with model oriented refinement?

Refinement can work wonderfully well in certain circumstances. Its paradigm of building models of the system at various levels of abstraction fits very naturally with the desire of designers to manipulate 'solid' representations of the system.

However, there are aspects of the real world design activity that can get into tension with refinement in serious developments.

Often physical models are involved:

- The continuous / discrete transition in modelling (as understood by engineers) is invariably not doable within (strict) refinement, restricting the scope of formal modelling.
- So at best, the abstract model ends up *already* in the discrete domain, bypassing most of the serious design.

Even for purely discrete applications:

- The real world never starts from a blank sheet, impeding ideal refinement.
- The complexity of real world applications can prohibit 100% faithful models.
- Management issues can prevent 100% adherence to refinement ideals.

Example: Small and large applications in general

‘Textbook’ world

Abs

↓ refine

↓ refine

↓ refine

Conc

‘Real’ world

‘True’ *Abs*

‘Conc’ *Abs*

↓ refine

Conc

Example: Small and large applications in general

‘Textbook’ world

Abs

↓ refine

↓ refine

↓ refine

Conc

‘Real’ world

‘True’ *Abs*

?

‘Conc’ *Abs*

↓ refine

Conc

This level becomes impenetrable.

The bigger the application gets, the worse the problem becomes.

And it is absolutely a question of ***SCALE.***

2. Retrenchment.

The ferocity of the refinement POs is what restricts their application in many areas.

2. Retrenchment.

The ferocity of the refinement POs is what restricts their application in many areas.

What can we do about the ferocity of the refinement POs?

We can attempt to judiciously weaken them.

2. Retrenchment.

The ferocity of the refinement POs is what restricts their application in many areas.

What can we do about the ferocity of the refinement POs?

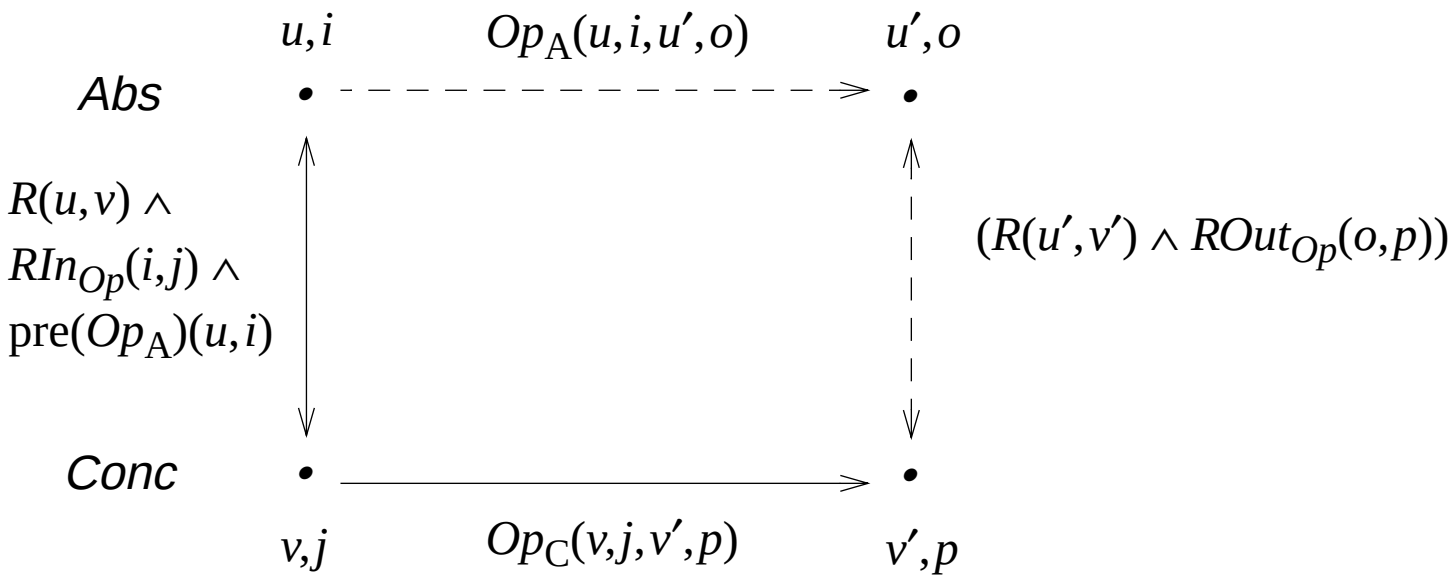
We can attempt to judiciously weaken them.

This of course would have consequences ... Refinement is **derived** from the prior assumption of substitutivity of concrete for abstract. Any interference with the refinement POs can fatally wound this link with substitutivity properties.

We are prepared to forgo this highly desirable aspect for the sake of:

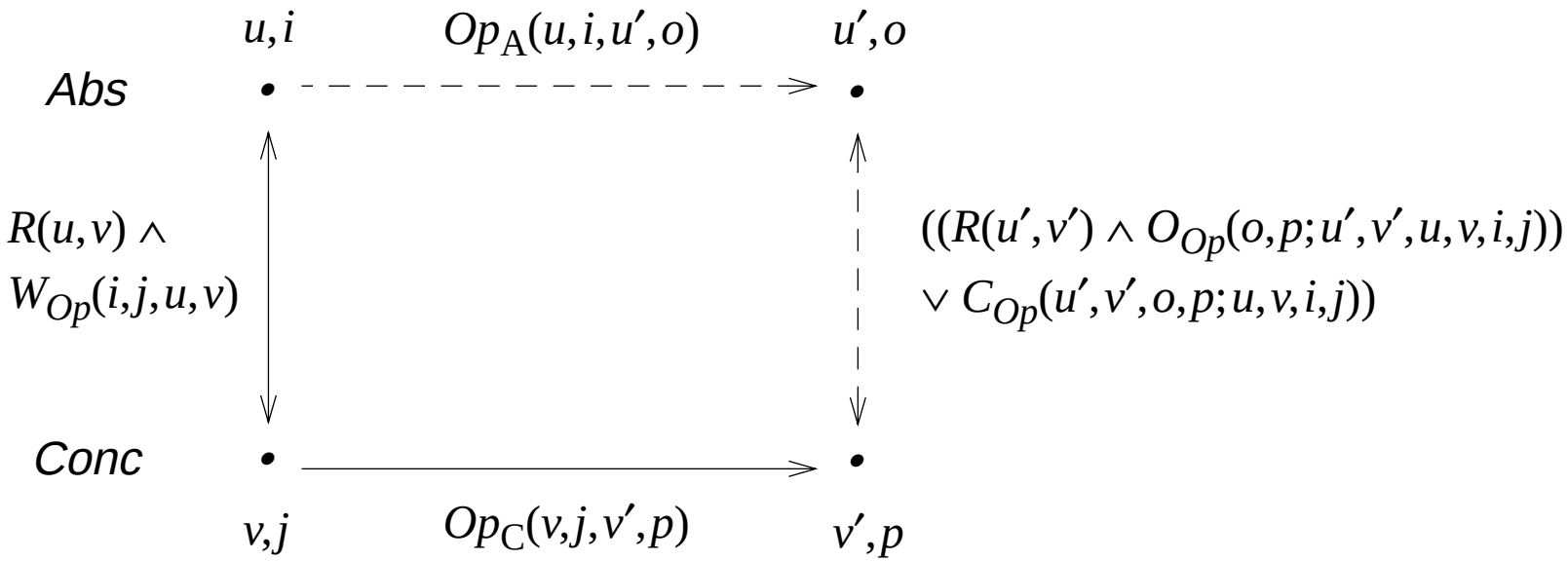
- Being able to address more (and more of) applications contexts formally.
- Being able to live with real world and management constraints.

The refinement PO



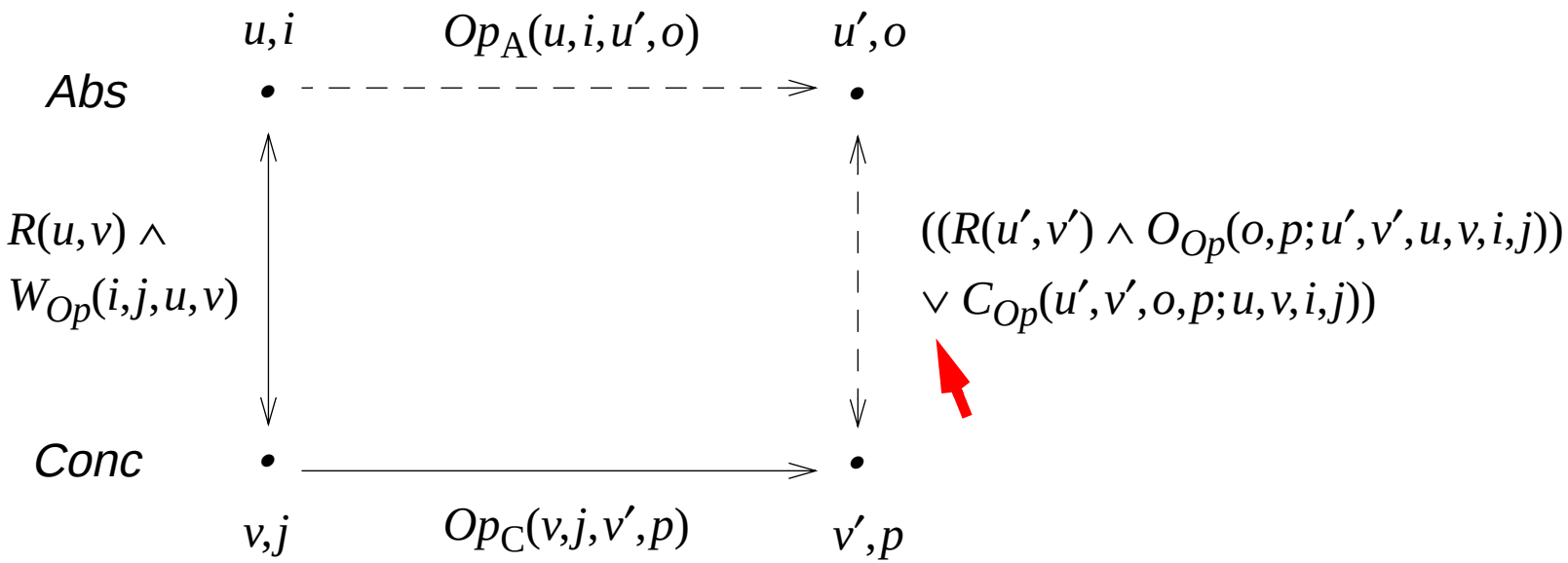
$$R(u,v) \wedge RIn_{Op}(i,j) \wedge \mathrm{pre}(Op_{\mathrm{A}})(u,i) \wedge Op_{\mathrm{C}}(v,j,v',p) \Rightarrow$$
$$(\exists\, u',o \bullet Op_{\mathrm{A}}(u,i,u',o) \wedge (R(u',v') \wedge ROut_{Op}(o,p)))$$

The retrenchment PO



$$R(u,v) \wedge W_{Op}(i,j,u,v) \wedge Op_{\mathrm{C}}(v,j,v',p) \Rightarrow$$
$$(\exists\, u',o \bullet Op_{\mathrm{A}}(u,i,u',o) \wedge ((R(u',v') \wedge O_{Op}(o,p;u',v',u,v,i,j))$$
$$\vee C_{Op}(u',v',o,p;u,v,i,j)))$$

The retrenchment PO



$$R(u,v) \wedge W_{Op}(i,j,u,v) \wedge Op_C(v,j,v',p) \Rightarrow$$
$$(\exists\, u',o \bullet Op_A(u,i,u',o) \wedge ((R(u',v') \wedge O_{Op}(o,p;u',v',u,v,i,j))$$
$$\vee\, C_{Op}(u',v',o,p;u,v,i,j)))$$

Definition of a retrenchment between *Abs* and *Conc*

A retrenchment is defined by the following data:

$\text{Ops}_\text{A} \subseteq \text{Ops}_\text{C}$ (the inclusion of operation names can be proper)

For each abstract operation name a relation $Op_\text{A} : \text{U} \times \text{I}_{Op_\text{A}} \leftrightarrow \text{U} \times \text{O}_{Op_\text{A}}$
For each concrete operation name a relation $Op_\text{C} : \text{V} \times \text{J}_{Op_\text{C}} \leftrightarrow \text{V} \times \text{P}_{Op_\text{C}}$

$R(u,v)$ (the retrieve (or glueing) relation)

$W_{Op}(i,j,u,v)$ (the within (or provided) relation)
$O_{Op}(o,p;u',v',u,v,i,j)$ (the output relation)
$C_{Op}(u',v',o,p;u,v,i,j)$ (the concedes relation)

} per $Op \in \text{Ops}_\text{A}$

Initialisation PO: $Init_\text{C}(v') \Rightarrow (\exists\, u' \bullet Init_\text{A}(u') \wedge R(u',v'))$ (as for refinement)

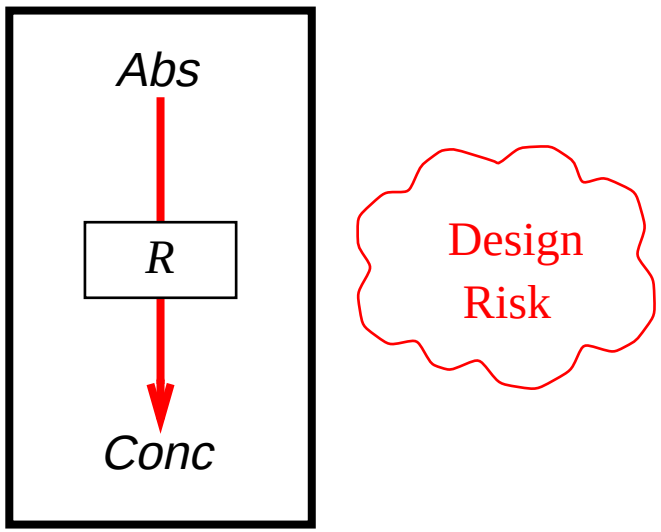
Operation PO:
(per $Op \in \text{Ops}_\text{A}$)

$$R(u,v) \wedge W_{Op}(i,j,u,v) \wedge Op_\text{C}(v,j,v',p) \Rightarrow (\exists\, u',o \bullet Op_\text{A}(u,i,u',o) \wedge ((R(u',v') \wedge O_{Op}(o,p;u',v',u,v,i,j)) \vee C_{Op}(u',v',o,p;u,v,i,j)))$$

Design and risk

Refinement

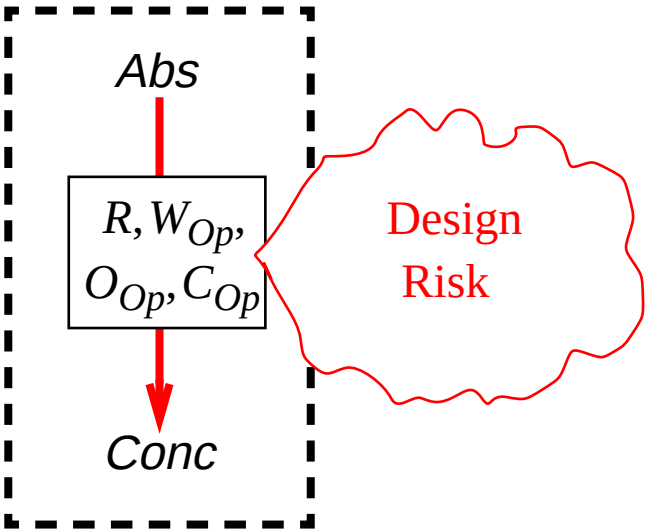
Refinement tends to expel design and risk from the concretisation/ accretion process. The process can become monolithic.



'Black Box'

Retrenchment

Retrenchment embraces design and risk within the evolution/ concretisation process. The process can separate concerns.



'Glass Box'

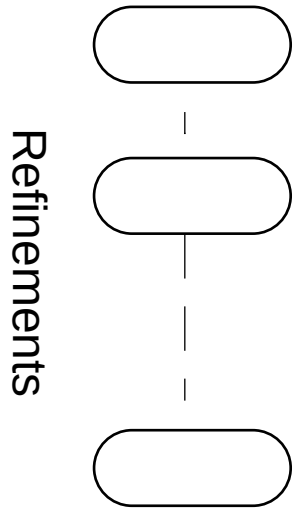
The Tower Pattern.

Just as software has identified commonly occurring ‘patterns’, we can see something similar for retrenchment, *independent of the issue dealt with in the retrenchment*.

The Tower Pattern.

Just as software has identified commonly occurring ‘patterns’, we can see something similar for retrenchment, *independent of the issue dealt with in the retrenchment*.

Most evident is the ***Tower Pattern***, built top down or bottom up ...:

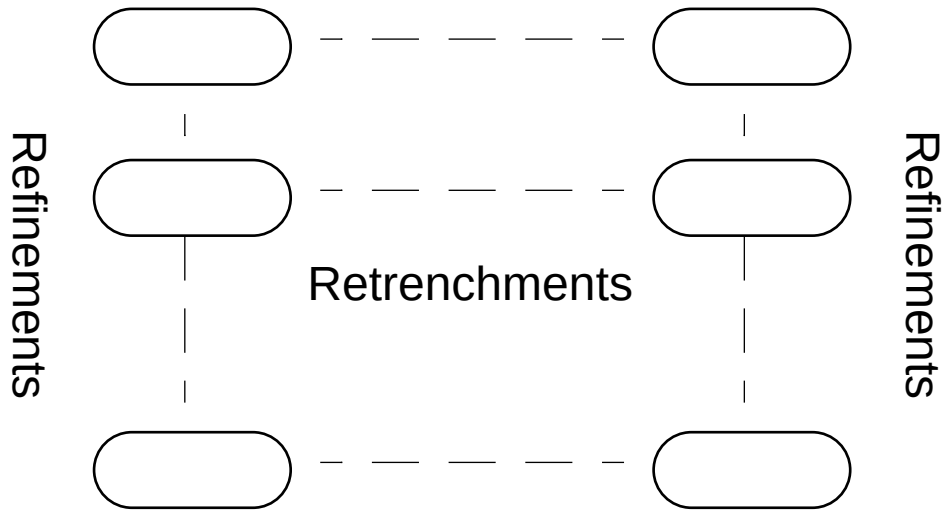


‘Ideal development’

The Tower Pattern.

Just as software has identified commonly occurring ‘patterns’, we can see something similar for retrenchment, *independent of the issue dealt with in the retrenchment*.

Most evident is the ***Tower Pattern***, built top down or bottom up ...:

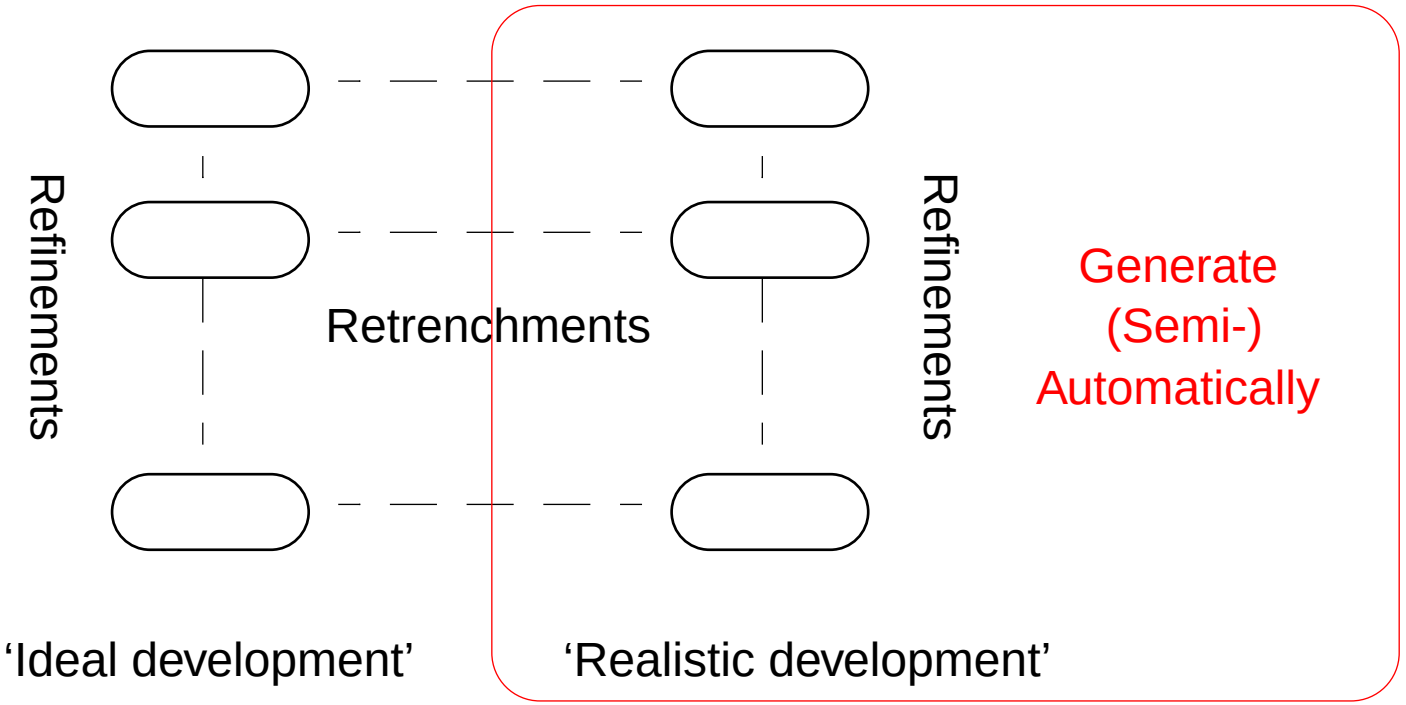


‘Ideal development’ ‘Realistic development’

The Tower Pattern.

Just as software has identified commonly occurring ‘patterns’, we can see something similar for retrenchment, *independent of the issue dealt with in the retrenchment*.

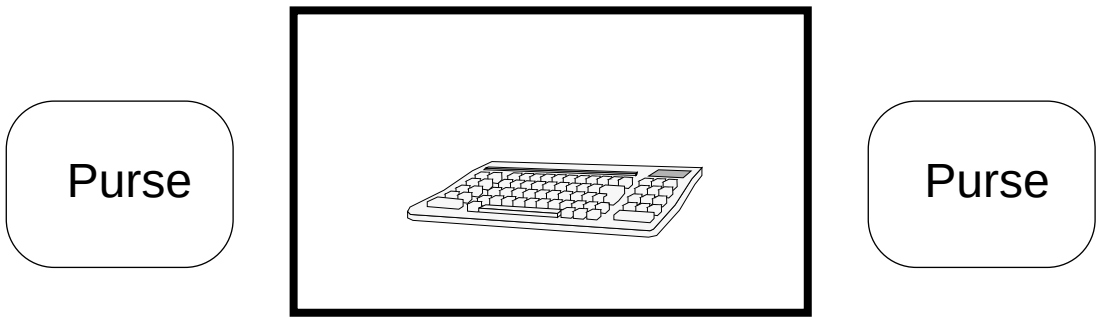
Most evident is the ***Tower Pattern***, built top down or bottom up ...:



3. The Mondex Purse.

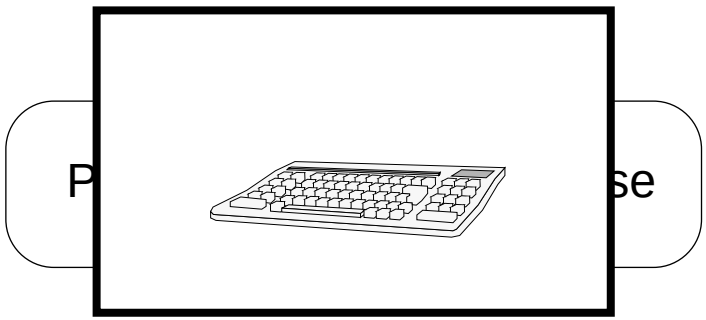
3. The Mondex Purse.

Cartoon:



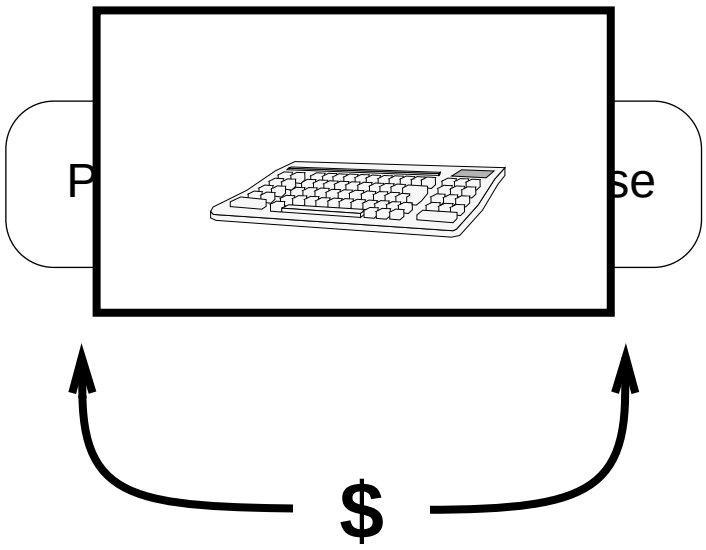
3. The Mondex Purse.

Cartoon:



3. The Mondex Purse.

Cartoon:



Basic Gameplan

- You put two purses into the ‘wallet’.
- You type in your instructions.
- The purses embark on a protocol to transfer money.

Basic Gameplan

- You put two purses into the ‘wallet’.
- You type in your instructions.
- The purses embark on a protocol to transfer money.

Points to bear in mind:

- The purses are on their own.
- The environment is hostile; the protocol can be halted/broken/spoofed etc. at any moment.
- At whatever point the protocol is halted, the total balance must (at worst) stay in favour of the bank .....

The Mondex refinement ...

The Mondex refinement ...
and the attendant retrenchment opportunities

The Mondex refinement ...
and the attendant retrenchment opportunities

The ‘public face’ of the Mondex development (as presented in the Oxford/Logica technical report PRG-126) presents a pristine example of refinement as applied to a real world scenario. Public Relations is a wonderful thing ...

The Mondex refinement ...
and the attendant retrenchment opportunities

The ‘public face’ of the Mondex development (as presented in the Oxford/Logica technical report PRG-126) presents a pristine example of refinement as applied to a real world scenario. Public Relations is a wonderful thing ...

In fact, the Mondex refinement glossed over a number of issues, that were carefully omitted from the public documentation.

- Purse sequence numbers are finite, not infinite (mentioned in PRG-126).
- The purse log is finite not infinite (concealed from PRG-126).
- The log archiving operation relies on a noninjective hash function rather than an injective function to validate clearing of purses’ logs (alluded to in PRG-126).
- The balance enquiry operation interacts badly with resolution of nondeterminism during money transfer (concealed from PRG-126).

All of these can be addressed with retrenchment.

Model Architecture

Only one requir'm't
Not a complete spec.

A-bstract

B-Ref

Concrete + global
invariants

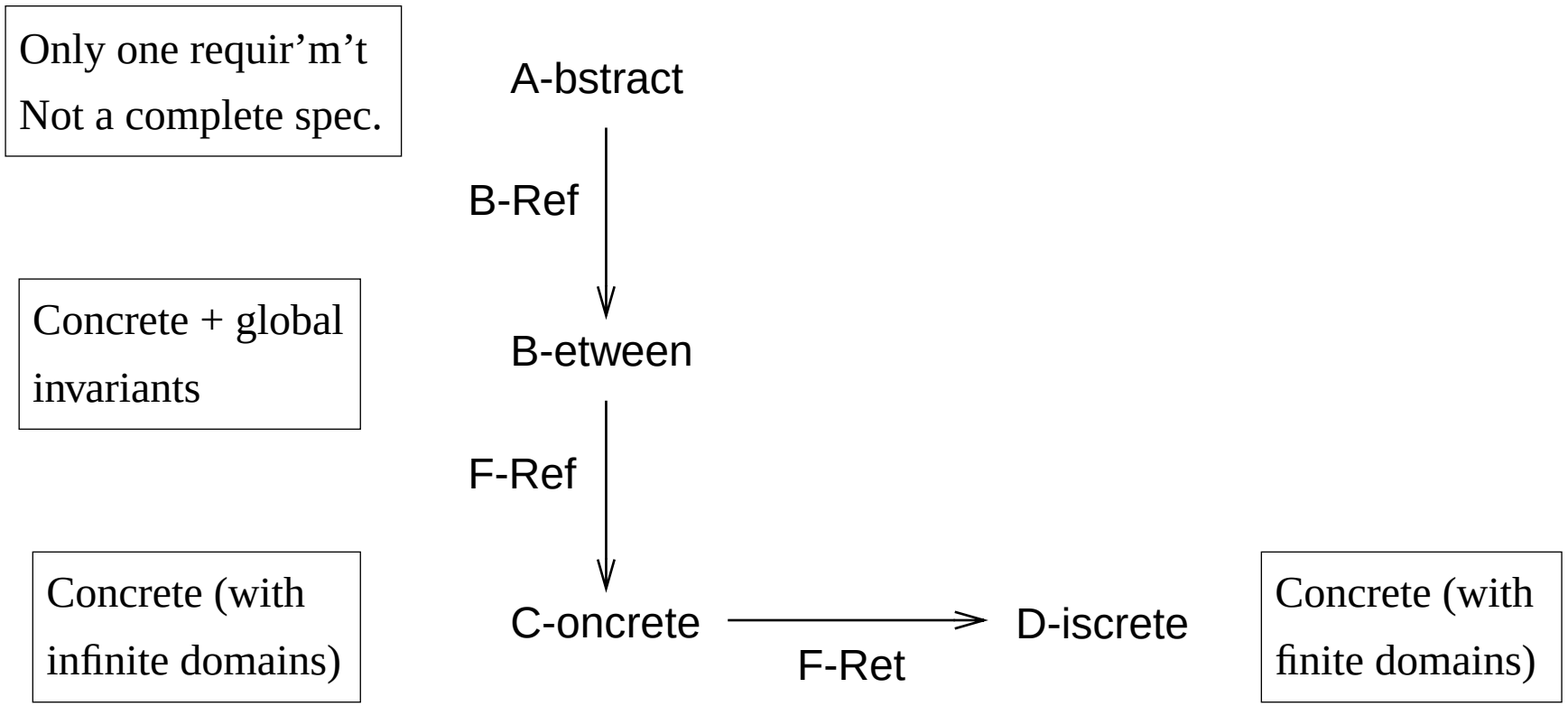
B-etween

F-Ref

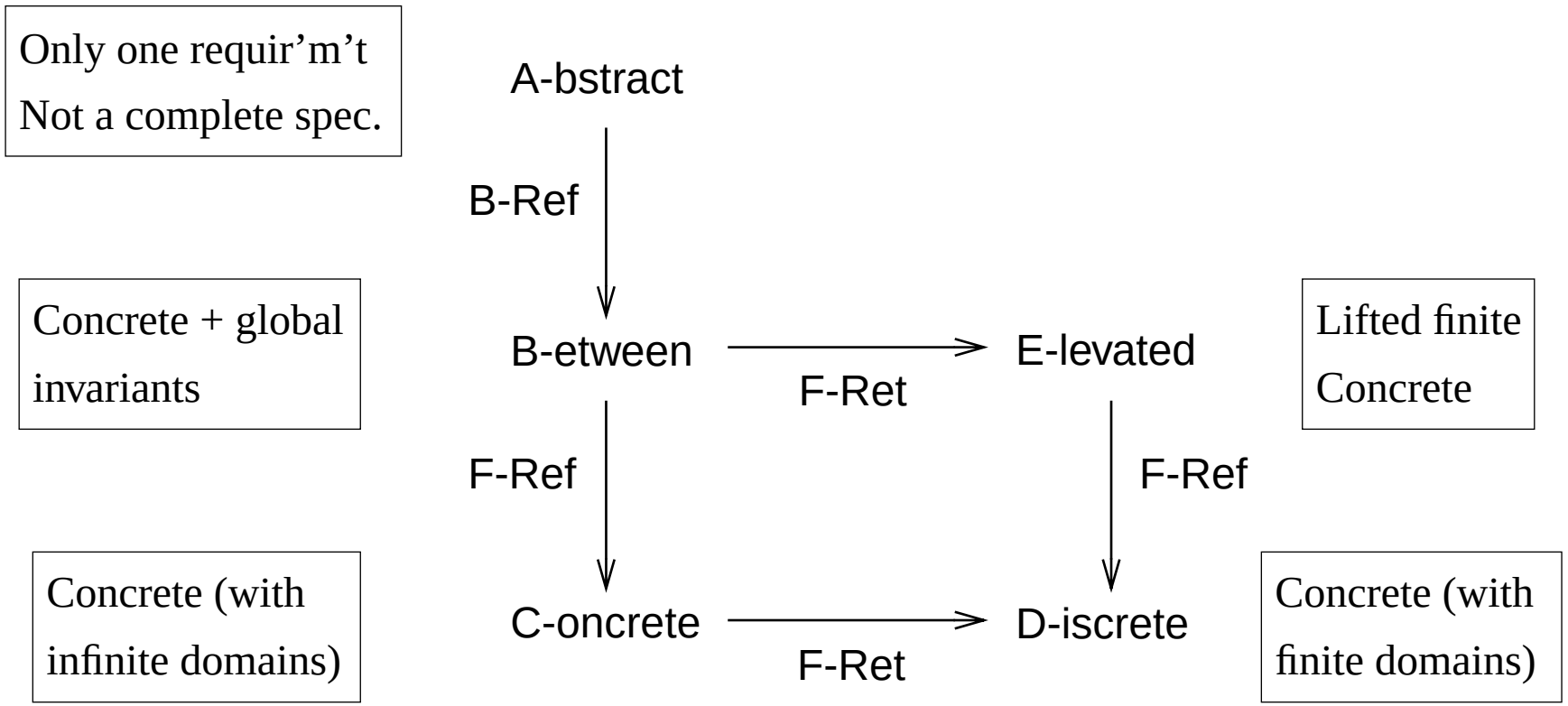
Concrete (with
infinite domains)

C-oncrete

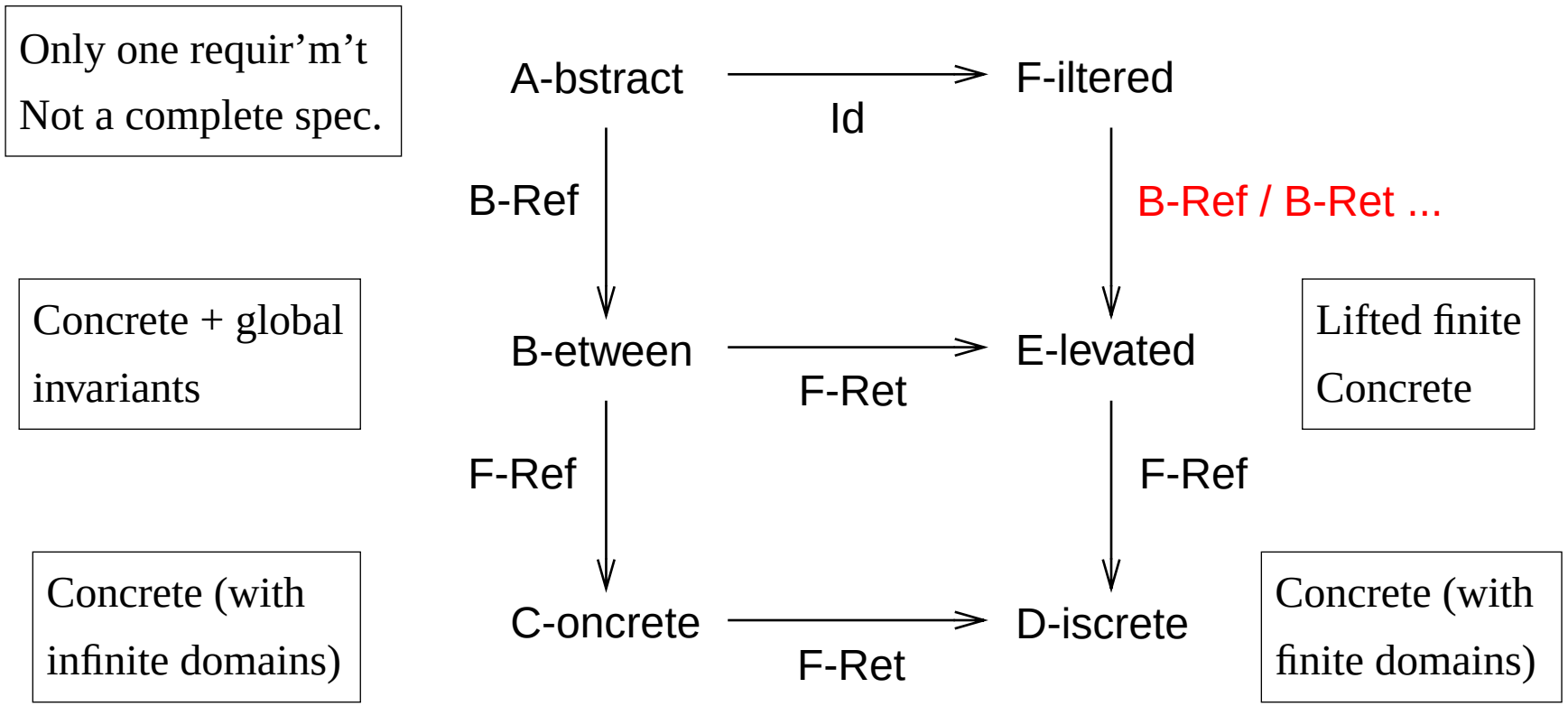
Model Architecture



Model Architecture



Model Architecture



4. Sequence Number.

Purse sequence numbers are finite, not infinite (mentioned in PRG-126).

- It is easy enough to build a D model with finite sequence numbers, and a C-to-D retrenchment.
- The C-to-D retrenchment can be lifted to give the E model.
- Under suitable circumstances, the E model can be a backward refinement of the A model, so A can be a copy of F. (In general a retrenchment is needed.)
- The retrenchment's concession yields an object in the formal development, which can be analysed during validation, in order to determine a value for the sequence number bound, large enough to guarantee it is never encountered during the purse's lifetime. (No such object using refinement alone.)

4. Sequence Number.

Each concrete purse has a sequence number *nextSeqNo*

Concrete model: *CnextSeqNo* : ℕ

In reality (and in Discrete model): *DnextSeqNo* : [0 .. BIGNUM] where BIGNUM is T.B.D.

One of the jobs of the Discrete model is to take this on board.

We will model [0 .. BIGNUM] as a subset of Z naturals; i.e. all the arithmetic and relational equipment of Z naturals is available *provided* it delivers an in-range answer.

So $DnextSeqNo' > DnextSeqNo$ implies that $DnextSeqNo' \leq$ BIGNUM

Operations using sequence number:

CIncreasePurseOkay, *CAbortPurseOkay*,
CStartFromPurseEafromOkay, *CStartToPurseEafromOkay*.

Minimally invasive D model ... per purse

C model

CIncreasePurseOkay

$\Delta CConPurse$
$Cm?, Cm! : CMESSAGE$

$\Xi CConPurseIncrease$
$CnextSeqNo' \geq CnextSeqNo$
$Cm! = \bot$

D model

DIncreasePurseOkay

$\Delta DConPurse$
$Dm?, Dm! : DMESSAGE$

$\Xi DConPurseIncrease$

$(DnextSeqNo < \text{BIGNUM} \Rightarrow$ *Refinement case*
$DnextSeqNo' \geq DnextSeqNo$
$Dm! = \bot)$

$(DnextSeqNo = \text{BIGNUM} \Rightarrow$ *Exceptional case*
$DnextSeqNo' = DnextSeqNo$
$Dm! = DpurseBlocked\ Dname)$

Retrenchment ... per purse:
CIncreasePurseOkay to *DIncreasePurseOkay*

Retrieve relation for C-D model development step:

$CConPurse$ **"="** $DConPurse$

Within relation for *IncreasePurseOkay* :

true

Output relation for *IncreasePurseOkay* :

$Cm! = Dm!$

Concedes relation for *IncreasePurseOkay* :

$CConPurseIncrease' \textbf{ "="} DConPurseIncrease' \land CnextSeqNo' \geq DnextSeqNo' \land$
$Cm! = \bot \land Dm! = DpurseBlocked\ Dname$

N.B. **"="** means C-variable = D-variable in the obvious way. Shorthand for a schema.

The D model ... raw world

The D world promotes a bunch of individual purses indexed by elements of $NAME$.

$$
\begin{array}{l}
\textit{DConWorld} \\
\hline
DconAuthPurse : NAME \rightarrowtail\!\!\!\!\!\!+\!\!\!\!\!\!\rightarrow DConPurse \\
Dether : \mathbb{P}\ DMESSAGE \\
Darchive : \mathbb{P}\ DLogbook \\
\hline
\forall\ n \in \mathrm{dom}\ DconAuthPurse \bullet (DconAuthPurse\ n).Dname = n \\
\forall\ nld \in Darchive \bullet first\ nld \in \mathrm{dom}\ DconAuthPurse \\
\hline
\end{array}
$$

The D model ... framing schema

ΦDOp

$\Delta DConWorld$
$\Delta DConPurse$
$Dm? , Dm! : DMESSAGE$
$Dname? : NAME$

$Dm? \in Dether$
$Dname? \in \mathrm{dom}\ DconAuthPurse$
$\theta DConPurse = DconAuthPurse\ Dname?$
$DconAuthPurse' = DconAuthPurse \oplus \{Dname? \mapsto \theta DConPurse'\}$
$Darchive' = Darchive$
$Dether' \subseteq Dether \cup \{Dm!\}$

The D model ... framing schema ... and *DIncrease*

ΦDOp

$\Delta DConWorld$
$\Delta DConPurse$
$Dm? , Dm! : DMESSAGE$
$Dname? : NAME$

$Dm? \in Dether$
$Dname? \in \mathrm{dom}\ DconAuthPurse$
$\theta DConPurse = DconAuthPurse\ Dname?$
$DconAuthPurse' = DconAuthPurse \oplus \{Dname? \mapsto \theta DConPurse'\}$
$Darchive' = Darchive$
$Dether' \subseteq Dether \cup \{Dm!\}$

DIncrease

$DIgnore \lor (\exists \Delta DConPurse \bullet \Phi DOp \land DIncreasePurseOkay)$

Retrenchment ... world level:
CIncrease to *DIncrease*

- The original Z refinement built the community of purses using Z promotion.

Retrenchment ... world level:
CIncrease to *DIncrease*

- The original Z refinement built the community of purses using Z promotion.
- There is more to the promotion of retrenchments of a collection of components than there is in the promotion of refinements of a collection of components, because the individual components can violate the retrieve relation individually.
- Component 1 may still be retrieving while component 2 has already conceded.
- Therefore in effect we get a $NAME$-indexed family of retrenchments, each referring to the retrenchment of component *name* within the whole.
- This basic formulation leads to many potential strengthenings, as, although it *might* be the case that component 2 has already conceded, it is not *necessarily* the case that it did so.
- Ultimately this collection of retrenchments can be disjunctively composed to give a full blown retrenchment between the worlds, but this goes beyond Z promotion.

Retrenchment and Promotion: A quick Survey

Because a retrenchment does not guarantee to re-establish the retrieve relation, there is more than one way of dealing with the promotion of a retrenchment; cf. the retrieve relation.

Strong Promotion Retrieve relation:

$$\mathrm{dom}\ CconAuthPurse = \mathrm{dom}\ DconAuthPurse \land (\forall\ name \in \mathrm{dom}\ CconAuthPurse \bullet etc.\ )$$

Give up as soon as one purse concedes

Weak Promotion Retrieve relation:

$$\mathrm{dom}\ CconAuthPurse = \mathrm{dom}\ DconAuthPurse \land (\exists\ name \in \mathrm{dom}\ CconAuthPurse \bullet etc.\ )$$

Keep going while at least one purse still good

Precise Promotion Retrieve relation ... (requires a separation axiom to hold):

$$\mathrm{dom}\ CconAuthPurse = \mathrm{dom}\ DconAuthPurse \land (\forall\ name \in Dgood \bullet etc.\ )$$

Keep track of which purses are still good

... *Focused* and *Inclusive* variants ...

Retrenchment ... world level: *CIncrease* to $DIncrease^{PP}$

World Level Retrieve relation for C-D model development step:

$$
\begin{array}{l}
\mathrm{dom}\ CconAuthPurse = \mathrm{dom}\ DconAuthPurse \\
(\forall\ Dnm \in Dgood \bullet \\
\quad (CconAuthPurse'\ Dnm?).CnextSeqNo = (DconAuthPurse'\ Dnm?).DnextSeqNo \\
\quad CconAuthPurse\ name\ \textbf{“=”}\ DconAuthPurse\ name) \\
Dgood \lhd Carchive\ \textbf{“=”}\ Dgood \lhd Darchive) \\
(Dgood \times Dgood) \lhd Cether\ \textbf{“=”}\ (Dgood \times Dgood) \lhd Dether
\end{array}
$$

World Level Within relation for *Increase* :

$$
\begin{array}{l}
Cname? = Dname? \\
Cname? \in Dgood
\end{array}
$$

World Level Output relation for *Increase* :

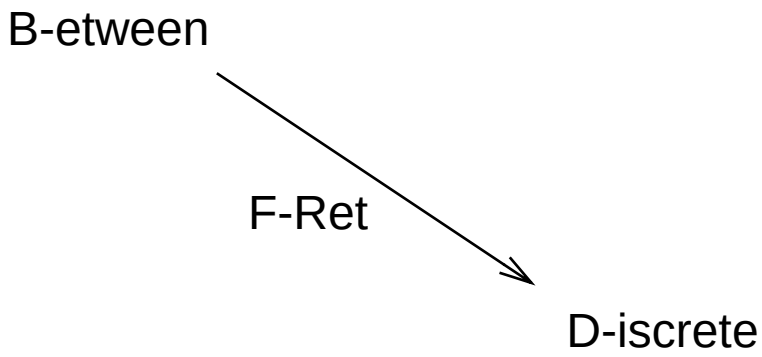
$Dgood' = Dgood$
$Cm! = Dm!$

World Level Concedes relation for *Increase* :

$Dgood' = Dgood - \{Dname?\}$
$(CconAuthPurse'\ Cname?).CConPurseIncrease'$ **"="**
$\quad (DconAuthPurse'\ Dname?).DConPurseIncrease'$
$(CconAuthPurse'\ Cname?).CnextSeqNo' \geq$
$\quad (DconAuthPurse'\ Dname?).DnextSeqNo'$
$Cm! = \bot$
$Dm! = DpurseBlocked\ Dname?$
$Dgood' \lhd Carchive'$ **"="** $Dgood' \lhd Darchive'$
$(Dgood' \times Dgood') \lhd Cether'$ **"="** $(Dgood' \times Dgood') \lhd Dether'$

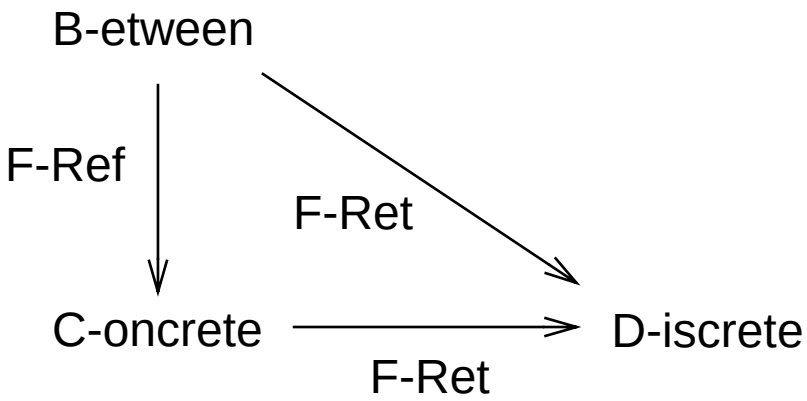
Lifting: E model

When there is a retrenchment such as from model B to model D,



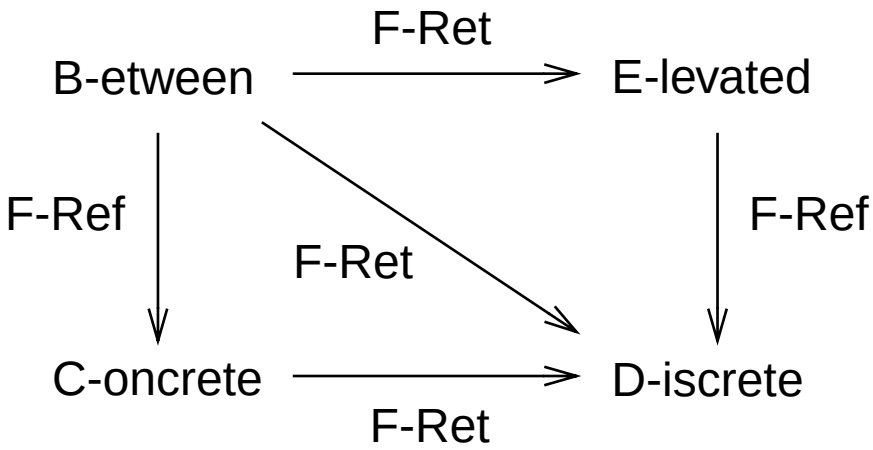
Lifting: E model

When there is a retrenchment such as from model B to model D, eg. via model C ...



Lifting: E model

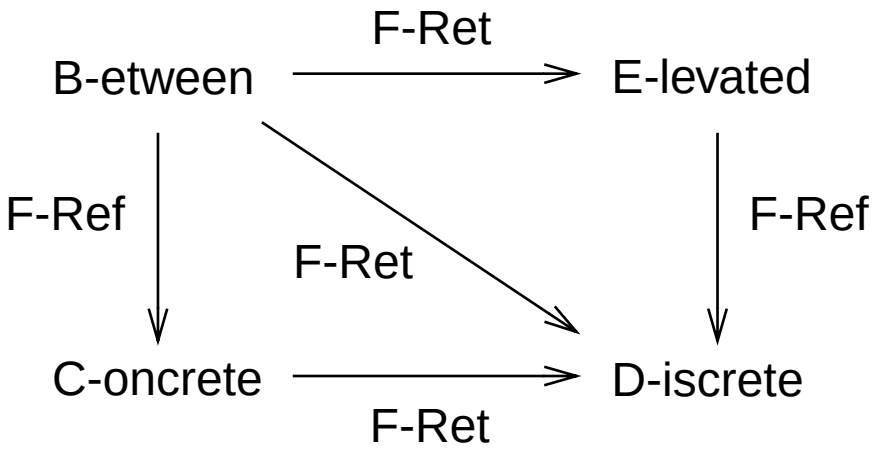
When there is a retrenchment such as from model B to model D, eg. via model C ...



Then there is a lifting of the B-D retrenchment ‘to the level of abstraction of B’.

Lifting: E model

When there is a retrenchment such as from model B to model D, eg. via model C ...



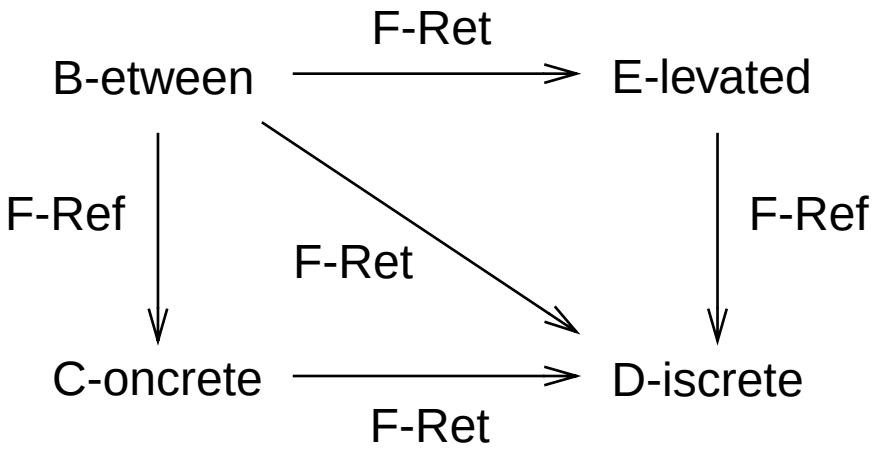
Then there is a lifting of the B-D retrenchment ‘to the level of abstraction of B’.

B variables: (u, i, u', o) ; D variables: (w, k, w', q) ; E variables: $((u,w), (i,k), (u',w'), (o,q))$

$$stp_{Op_{\mathrm{E}}}((u,w), (i,k), (u',w'), (o,q)) \;=$$
$$G(u,w) \wedge P_{Op}(i,k,u,w) \wedge stp_{Op_{\mathrm{B}}}(u,i,u',o) \wedge$$
$$((G(u',w') \wedge O_{Op}(o,q;u',w',i,k,u,w)) \vee C_{Op}(u',w',o,q;i,k,u,w))$$

Lifting: E model

When there is a retrenchment such as from model B to model D, eg. via model C ...



Then there is a lifting of the B-D retrenchment ‘to the level of abstraction of B’.

B variables: (u, i, u', o) ; D variables: (w, k, w', q) ; E variables: $((u,w), (i,k), (u',w'), (o,q))$

$$stp_{Op_{\mathrm{E}}}((u,w), (i,k), (u',w'), (o,q)) \;=$$
$$G(u,w) \wedge P_{Op}(i,k,u,w) \wedge stp_{Op_{\mathrm{B}}}(u,i,u',o) \wedge$$
$$((G(u',w') \wedge O_{Op}(o,q;u',w',i,k,u,w)) \vee C_{Op}(u',w',o,q;i,k,u,w))$$

N.B.
A simplified special case.

Stochastic aspects: likelihood of achieving concession

The purse just blocks if the limit on *nextSeqNo* is reached ... is this reasonable?

Stochastic aspects: likelihood of achieving concession

The purse just blocks if the limit on *nextSeqNo* is reached ... is this reasonable?

Dumb story

To preclude covert channels, *nextSeqNo* increments are not fixed, but are identically distributed random variables drawn from a probability distribution Θ say. Suppose Θ has mean and variance both $O(10)$.

Stochastic aspects: likelihood of achieving concession

The purse just blocks if the limit on *nextSeqNo* is reached ... is this reasonable?

Dumb story

To preclude covert channels, *nextSeqNo* increments are not fixed, but are identically distributed random variables drawn from a probability distribution Θ say. Suppose Θ has mean and variance both $O(10)$.

The determined shopper makes $O(100)$ transactions a day. Δ*nextSeqNo* $O(10^3)$ per day.

One year $O(10^3)$ days. Δ*nextSeqNo* $O(10^6)$ per year.

Stochastic aspects: likelihood of achieving concession

The purse just blocks if the limit on *nextSeqNo* is reached ... is this reasonable?

Dumb story

To preclude covert channels, *nextSeqNo* increments are not fixed, but are identically distributed random variables drawn from a probability distribution Θ say. Suppose Θ has mean and variance both $O(10)$.

The determined shopper makes $O(100)$ transactions a day. Δ*nextSeqNo* $O(10^3)$ per day.

One year $O(10^3)$ days. Δ*nextSeqNo* $O(10^6)$ per year.

1. Say BIGNUM = $O(2^{16}) = O(64\times10^3)$ .... We hit BIGNUM in a couple of months.

2. Say BIGNUM = $O(2^{32}) = O(4\times10^9)$ .... It's about 4000 years before we hit BIGNUM.
The banking system underpinning the purse will have fallen before then. No worries.

3. Say BIGNUM = $O(2^{64}) = O(16\times10^{18})$ .... It's about $O(16\times10^{12})$ years before we hit BIGNUM. Many orders of magnitude longer than age of universe etc.

Stochastic aspects: likelihood of achieving concession

More sophisticated story

The increments of *nextSeqNo*, δSN_i are the 'arrivals' of a renewal process $N(t)$:

$$nextSeqNo_n = \delta SN_0 + \delta SN_1 + \ldots + \delta SN_n$$

$$N(t) = \max\{n \mid nextSeqNo_n \leq t\}$$

$N(t)$ is the random variable saying how likely it is that different numbers n of increments will get *nextSeqNo* up to value t. We are interested in the distribution of $N(\text{BIGNUM})$.

Stochastic aspects: likelihood of achieving concession

More sophisticated story

The increments of *nextSeqNo*, δSN_i are the 'arrivals' of a renewal process $N(t)$:

$$nextSeqNo_n = \delta SN_0 + \delta SN_1 + \ldots + \delta SN_n$$

$$N(t) = \max\{n \mid nextSeqNo_n \leq t\}$$

$N(t)$ is the random variable saying how likely it is that different numbers n of increments will get *nextSeqNo* up to value t. We are interested in the distribution of $N(\text{BIGNUM})$.

This is textbook stuff (fortunately).

First order theory says: $N(t) \xrightarrow{\text{a.s.}} t/\mu$ and $\mathbf{E}[N(t)] \to t/\mu$

For $t = \text{BIGNUM} = O(2^{32})$, $\mu = O(10)$, $N \to 4\times10^9/10$. At $O(100)$ transactions a day: 4K yrs.

Stochastic aspects: likelihood of achieving concession

More sophisticated story

The increments of *nextSeqNo*, δSN_i are the 'arrivals' of a renewal process $N(t)$:

$$nextSeqNo_n = \delta SN_0 + \delta SN_1 + \ldots + \delta SN_n$$

$$N(t) = \max\{n \mid nextSeqNo_n \leq t\}$$

$N(t)$ is the random variable saying how likely it is that different numbers n of increments will get *nextSeqNo* up to value t. We are interested in the distribution of $N(\text{BIGNUM})$.

This is textbook stuff (fortunately).

First order theory says: $N(t) \xrightarrow{\text{a.s.}} t/\mu$ and $\mathbf{E}[N(t)] \to t/\mu$

For $t = \text{BIGNUM} = O(2^{32})$, $\mu = O(10)$, $N \to 4\times10^9/10$. At $O(100)$ transactions a day: 4K yrs.

Second order theory says: $\dfrac{(N(t) - t/\mu)}{\sqrt{t\sigma^2/\mu^3}} \xrightarrow{\text{D}} N(0,1)$

Stochastic aspects: likelihood of achieving concession

More sophisticated story

The increments of *nextSeqNo*, δSN_i are the 'arrivals' of a renewal process $N(t)$:

$$nextSeqNo_n = \delta SN_0 + \delta SN_1 + \ldots + \delta SN_n$$

$$N(t) = \max\{n \mid nextSeqNo_n \leq t\}$$

$N(t)$ is the random variable saying how likely it is that different numbers n of increments will get *nextSeqNo* up to value t. We are interested in the distribution of $N(\text{BIGNUM})$.

This is textbook stuff (fortunately).

First order theory says: $N(t) \xrightarrow{\text{a.s.}} t/\mu$ and $\mathbf{E}[N(t)] \to t/\mu$

For $t = \text{BIGNUM} = O(2^{32})$, $\mu = O(10)$, $N \to 4\times10^9/10$. At $O(100)$ transactions a day: 4K yrs.

Second order theory says: $\dfrac{(N(t) - t/\mu)}{\sqrt{t\sigma^2/\mu^3}} \xrightarrow{\text{D}} N(0,1)$

Variance: ... about a week!
Dumb story stands up OK.

4. Purse Log.

The purse log is finite not infinite (concealed from PRG-126).

Summary:

- It is easy enough to build a D model with a finite purse log, and a C-to-D retrenchment.
- The C-to-D retrenchment can be lifted to give the E model.
- Under suitable circumstances, the E model can be a backward refinement of the A model, so A can be a copy of F. (In general a retrenchment is needed.)
- The retrenchment's concession yields an object in the formal development, which can be analysed during validation, to check how the non-refinement part of the design addresses overall system requirements.

One operation involved in adding to the log: *CAbortPurseOkay*.

4. Purse Log.

An aborting transaction is logged by a participating purse made aware of such.

CAbortPurseOkay

$\Delta CConPurse$
$Cm?, Cm! : CMESSAGE$

$\Xi CConPurseAbort$
$CLogIfNecessary$ — *Bounded in D world*
$CnextSeqNo' \geq CnextSeqNo$ — *Bounded in D world ... for simplicity, ignore boundedness here*
$Cstatus' = CeaFrom$

CLogIfNecessary

$\Delta CConPurse$

$CexLog' = CexLog \cup (\text{if } Cstatus \in \{Cepv, Cepa\} \text{ then } \{CpdAuth\} \text{ else } \emptyset)$

In reality, the log is not an (unbounded) set, but of maximum size LOGMAX.

In reality, LOGMAX is *small* (about 5 in fact), since smartcard memory is tight, and a full record of a transaction is of non-trivial size.

When a failing transaction fills the last log slot, we must prevent further transactions (since there won't be anywhere to log *them* if *they* fail).

In reality, the log is not an (unbounded) set, but of maximum size LOGMAX.

In reality, LOGMAX is *small* (about 5 in fact), since smartcard memory is tight, and a full record of a transaction is of non-trivial size.

When a failing transaction fills the last log slot, we must prevent further transactions (since there won't be anywhere to log *them* if *they* fail).

Various possibilities. Each can be described via an appropriate retrenchment.

In reality, the log is not an (unbounded) set, but of maximum size LOGMAX.

In reality, LOGMAX is *small* (about 5 in fact), since smartcard memory is tight, and a full record of a transaction is of non-trivial size.

When a failing transaction fills the last log slot, we must prevent further transactions (since there won't be anywhere to log *them* if *they* fail).

Various possibilities. Each can be described via an appropriate retrenchment.

We introduce a new purse status *DexLogFull*, for when the log has filled.

Fortunately, all non-trivial operations are guarded on their before-state, so that no other non-trivial operation is affected by the introduction of *DexLogFull* (aside from *CClearExceptionLogPurseOkay*).

DAbortPurseOkay

$\Delta DConPurse$

$Dm?, Dm! : DMESSAGE$

$\Xi DConPurseAbort$

$DnextSeqNo' \geq DnextSeqNo$

$(\#\, DexLog < \text{LOGMAX} \Rightarrow DLogIfNecessary)$

$(\#\, DexLog \geq \text{LOGMAX} \Rightarrow DexLog' = DexLog)$

$(\#\, DexLog < \text{LOGMAX} - 1 \Rightarrow Dstatus' = DeaFrom \wedge Dm! = \bot)$

$(\#\, DexLog \geq \text{LOGMAX} - 1 \Rightarrow Dstatus' = DexLogFull \wedge Dm! = \text{“}Purse\ blocked.\ Go\ to\ bank.\text{”})$

where

DLogIfNecessary

$\Delta DConPurse$

$DexLog' = DexLog \cup (\text{if } Dstatus \in \{Depv, Depa\} \text{ then } \{DpdAuth\} \text{ else } \emptyset)$

Retrenchment ... per purse:
CAbortPurseOkay to *DAbortPurseOkay*

Retrieve relation for C-D model development step:

$$CConPurseAbort \text{ ``=''} DConPurseAbort \wedge$$
$$(Cstatus \text{ ``=''} Dstatus \vee Dstatus = DexLogFull) \wedge$$
$$CnextSeqNo = DnextSeqNo \wedge$$
$$CexLog \text{ ``=''} DexLog$$

Within relation for *AbortPurseOkay* :

true

Output relation for *AbortPurseOkay* :

$Cm! = Dm!$

Concedes relation for *AbortPurseOkay* :

$$
\begin{aligned}
&\mathit{CConPurseAbort}' \text{ “=” } \mathit{DConPurseAbort}' \wedge \\
&\mathit{CexLog}' \supseteq \mathit{DexLog}' \wedge \\
&\mathit{CnextSeqNo}' = \mathit{DnextSeqNo}' \wedge \\
&\mathit{Cstatus}' = \mathit{DeaFrom} \wedge \\
&\mathit{Dstatus}' = \mathit{DexLogFull} \wedge \\
&\mathit{Cm}! = \bot \wedge \\
&\mathit{Dm}! = \text{“}\mathit{Purse\ blocked.\ Go\ to\ bank.}\text{”}
\end{aligned}
$$

N.B. **“=”** means *C*-variable = *D*-variable in the obvious way. Shorthand for a schema.

Retrenchment ... world level: *CAbort* to $DAbort^{WP}$

World Level Retrieve relation for C-D model development step:

dom *CconAuthPurse* = dom *DconAuthPurse*
$\exists$ *Dname*? $\in$ dom *CconAuthPurse* $\cap$ dom *DconAuthPurse* • $RBody^{PW}_{CD}$

World Level Within relation for *Abort* :

Cname? = *Dname*?
$RBody^{PW}_{CD}$

where $RBody^{PW}_{CD}$ is:

Dname? $\in$ dom *CconAuthPurse* $\cap$ dom *DconAuthPurse*
“ CDnamedConPurseAbortRetrieve *Dname*? ”
{*Dname*?} $\lhd$ *Carchive* = {*Dname*?} $\lhd$ *Darchive*)
({*Dname*?} $\times$ {*Dname*?}) $\lhd$ *Cether* = ({*Dname*?} $\times$ {*Dname*?}) $\lhd$ *Dether*

World Level Output relation for *Abort* :

$Cm! = Dm!$

World Level Concedes relation for *Abort* :

“ CDnamedConPurseAbortEquality′ *Dname*? ”
“ CDnamedConPurseAbortConcession *Dname*? ”
$\{Dname?\} \lhd Carchive' = \{Dname?\} \lhd Darchive'$
$(\{Dname?\} \times \{Dname?\}) \lhd Cether' = (\{Dname?\} \times \{Dname?\}) \lhd Dether'$

Validating the small

There are many implementation design decisions that depend on a finite limit.

- Usually the limit is *large* compared with amount used in practice (cf. sequence numbers). So usually, the validation checks that the bound is big enough.
- In the log case, the limit is small.
- In the log case, log entries are large, and matter for their contents (i.e. you don't just want to count them).

Validating the small

There are many implementation design decisions that depend on a finite limit.

- Usually the limit is *large* compared with amount used in practice (cf. sequence numbers). So usually, the validation checks that the bound is big enough.
- In the log case, the limit is small.
- In the log case, log entries are large, and matter for their contents (i.e. you don't just want to count them).

Validation for the bounded log design:

- You can't afford to lose a log entry.
- So when log full, you cannot afford to start any activity that could generate one (i.e. a fresh transaction).
- The design for the retrenched system achieves this. So it's OK.

Validating Sequence Numbers vs. Small Logs

Sequence Number

Few bits give large capacity.

Full Log

Restricted space is tight constraint.

Validating Sequence Numbers vs. Small Logs

Sequence Number	Full Log
Few bits give large capacity.	Restricted space is tight constraint.
Large bound achievable means bound never reached.	Small capacity means reaching bound ‘inevitable’.

Validating Sequence Numbers vs. Small Logs

Sequence Number

Few bits give large capacity.

Large bound achievable means bound never reached.

Full Log

Restricted space is tight constraint.

Small capacity means reaching bound ‘inevitable’.

Design out possibility of further transactions when log full.

Validating Sequence Numbers vs. Small Logs

Sequence Number	Full Log
Few bits give large capacity.	Restricted space is tight constraint.
Large bound achievable means bound never reached.	Small capacity means reaching bound ‘inevitable’.
	Design out possibility of further transactions when log full.
Probability of *DnextSeqNo* overrun: zero.	Probability of *DexLog* overrun: zero.

5. Hash Function.

The log archiving operation relies on a noninjective hash function rather than an injective function to validate clearing of purses' logs (alluded to in PRG-126).

- It is easy enough to build a D model using a noninjective hash function, and a C-to-D retrenchment.
- The C-to-D retrenchment can be lifted to give the E model.
- The retrenchment's concession gives an object in the formal development, which can be analysed during validation, in order to determine the likelihood of a purse's receiving in error a 'believable' clear-log message due to the noninjectivity of the hash function. The concession refers to the failure of the 'all value accounted' security property.

5. Hash function.

Once you've arrived at the bank with your non-functioning purse, a protocol is engaged in to return to normal working.

First the purse is asked for the log records, and sends them one by one to the archive:

CReadExceptionLogPurseEafromOkay

$$\Xi CConPurse$$
$$Cm?, Cm! : CMESSAGE$$

$$Cm? = CreadExceptionLog$$
$$Cstatus = CeaFrom$$
$$Cm! \in \{\bot\} \cup \{Cld : CexLog' \bullet CexceptionLogResult(Cname, Cld)\}$$

Then the archive sends a Clear message in order to instruct the purse to clear its log (on the assumption that all the log entries are safely stored in the archive).

CClearExceptionLogPurseEafromOkay

$\Delta CConPurse$
$Cm?, Cm! : CMESSAGE$

$CexLog \neq \emptyset$
$Cm? = CexceptionLogClear(Cname, Cimage\, CexLog)$
$Cstatus = CeaFrom$
$\Xi CConPurseClear$
$CexLog' = \emptyset$
$Cm! = \bot$

The function *Cimage* is an injection so the purse knows it's being told to clear exactly the right entries:

$$Cimage : \mathbb{P}_1\, CPayDetails \rightarrowtail CLEAR$$

But in reality, in the D world, it is a cryptographically strong hash of *PayDetails* subsets.

$$Dimage : \mathbb{P}_1\, DPayDetails \rightarrow CLEAR$$

(Aside from this, the other details of the D world's *DClearExceptionLogPurseEafromOkay* operation are as for the C world.)

The function *Cimage* is an injection so the purse knows it's being told to clear exactly the right entries:

$$Cimage : \mathbb{P}_1\, CPayDetails \rightarrowtail CLEAR$$

But in reality, in the D world, it is a cryptographically strong hash of *PayDetails* subsets.

$$Dimage : \mathbb{P}_1\, DPayDetails \rightarrow CLEAR$$

(Aside from this, the other details of the D world's *DClearExceptionLogPurseEafromOkay* operation are as for the C world.)

If it's a hash ... there can be a clash.

Such a clash can lead to the premature deletion of log entries if an unfortunate purse receives a spurious Clear message which contains the same hash as is generated by its currently unarchived log entries. Such an event would destroy the *AllValueAccounted* security invariant.

A job for retrenchment.

Retrenchment ... world level:
CClearExceptionLog to *DClearExceptionLog*

World Level Retrieve relation for C-D model development step:

$$\begin{aligned}&\text{dom } CconAuthPurse = \text{dom } DconAuthPurse \wedge \\ &(\forall\, nm \in \text{dom } CconAuthPurse \bullet CconAuthPurse\ nm \text{ “=” } DconAuthPurse\ nm) \wedge \\ &Cether \text{ “=” } Dether\end{aligned}$$

World Level Within relation for *ClearExceptionLog* :

$$\begin{aligned}&Cname? = Dname? \wedge \\ &(\{Cname?\} \lhd Carchive) \cup (CconAuthPurse\ Cname?).CexLog \text{ “=”} \\ &\quad (\{Dname?\} \lhd Darchive) \cup (DconAuthPurse\ Dname?).DexLog \wedge \\ &Cm? = CexceptionLogClear(Cname, Cimage\ (CconAuthPurse\ Cname?).CexLog) \wedge \\ &Dm? = DexceptionLogClear(Dname, Dimage\ (DconAuthPurse\ Dname?).DexLog)\end{aligned}$$

World Level Output relation for *ClearExceptionLog* :

$$
\begin{array}{l}
(CconAuthPurse\ Cname?).CexLog \subseteq \{Cname?\} \lhd Carchive \land \\
(DconAuthPurse\ Dname?).DexLog \subseteq \{Dname?\} \lhd Darchive \land \\
Cm! = Dm! \land \\
CAllValueAccountedPurse'\ Cname?
\end{array}
$$

World Level Concedes relation for *ClearExceptionLog* :

$$
\begin{array}{l}
CconAuthPurse'\ Cname?\ \textbf{“=”}\ DconAuthPurse'\ Dname? \land \\
\neg((CconAuthPurse\ Cname?).CexLog \subseteq \{Cname?\} \lhd Carchive \land \\
\quad (DconAuthPurse\ Dname?).DexLog \subseteq \{Dname?\} \lhd Darchive) \land \\
Cm! = Dm! \land \\
\neg CAllValueAccountedPurse'\ Cname?
\end{array}
$$

N.B. *CAllValueAccountedPurse* is a purse-specific *AllValueAccounted* property, rather more sensitive than the corresponding PRG-126 property.

F model

Construction of the E model goes much as in previous cases.

Two plausible strategies for the F model.

F model

Construction of the E model goes much as in previous cases.

Two plausible strategies for the F model.

1. Keep the F model as a copy of the A model ... but change the retrieve relation in the backward refinement to express:

   “ Abs Lost $\geq$ Conc Lost **”** (instead of equality)

F model

Construction of the E model goes much as in previous cases.

Two plausible strategies for the F model.

1. Keep the F model as a copy of the A model ... but change the retrieve relation in the backward refinement to express:

   “ Abs Lost $\geq$ Conc Lost **”** (instead of equality)

2. Introduce a fresh operation at the abstract level to nondeterministically:
   either, *skip*,
   or, lose some of the Abs Lost value.

   This entails a straightforward retrenchment from the A model to the F model.

Stochastic aspects: likelihood of achieving concession

The purse erroneously discards log entries if a spurious *exceptionLogClear* message is received ... in particular there is no purse state interlock to record if log entries have been sent. Is this reasonable?

Stochastic aspects: likelihood of achieving concession

The purse erroneously discards log entries if a spurious *exceptionLogClear* message is received ... in particular there is no purse state interlock to record if log entries have been sent. Is this reasonable?

Well:

- Sending an entry offers no guarantee of arrival.
- The log is used as backup for ensuring the reliability of the value transfer protocol, one of whose sources of failure is the failure of transmission of securely encoded messages. If these might be failing, what's the point of trying a more robust log archiving protocol, prone to the same weakness?

So fair enough.

Analysis

Assume the arrival of a spurious *exceptionLogClear* message is a chance event.

- If it's a fortuitous occurrence, then all bits of the message are random variables.
- If it's a malicious attack, assume all but hashed bits are known.

Message details	Bits
• Header "*CexceptionLogClear*"	4
• Purse Id *Dname*?	64
• Bijected log entries *Cimage* ...	$(2\times64+3\times32)\times5 \geq 1\text{K}$
• Hashed log entries *Dimage* ...	64?, 128?, 256?

Analysis

- To defend against malicious attack, only the hash offers any protection.
  So need at least 256 bits of hash to give reasonable protection.
  At 1 trial per ms., ... orders of magnitude longer than age of universe.

Analysis

- To defend against malicious attack, only the hash offers any protection.
  So need at least 256 bits of hash to give reasonable protection.
  At 1 trial per ms., ... orders of magnitude longer than age of universe.
- For accidental erasure, the malicious threshold, 256 bits, is firmed up with (say):
  4 bits of header, 64 bits of purse id, likelihood of bank-comms failure on the precise link to the archive being used at the time ... so erasure is *EXTREMELY* unlikely, even when the injective clear is not used.

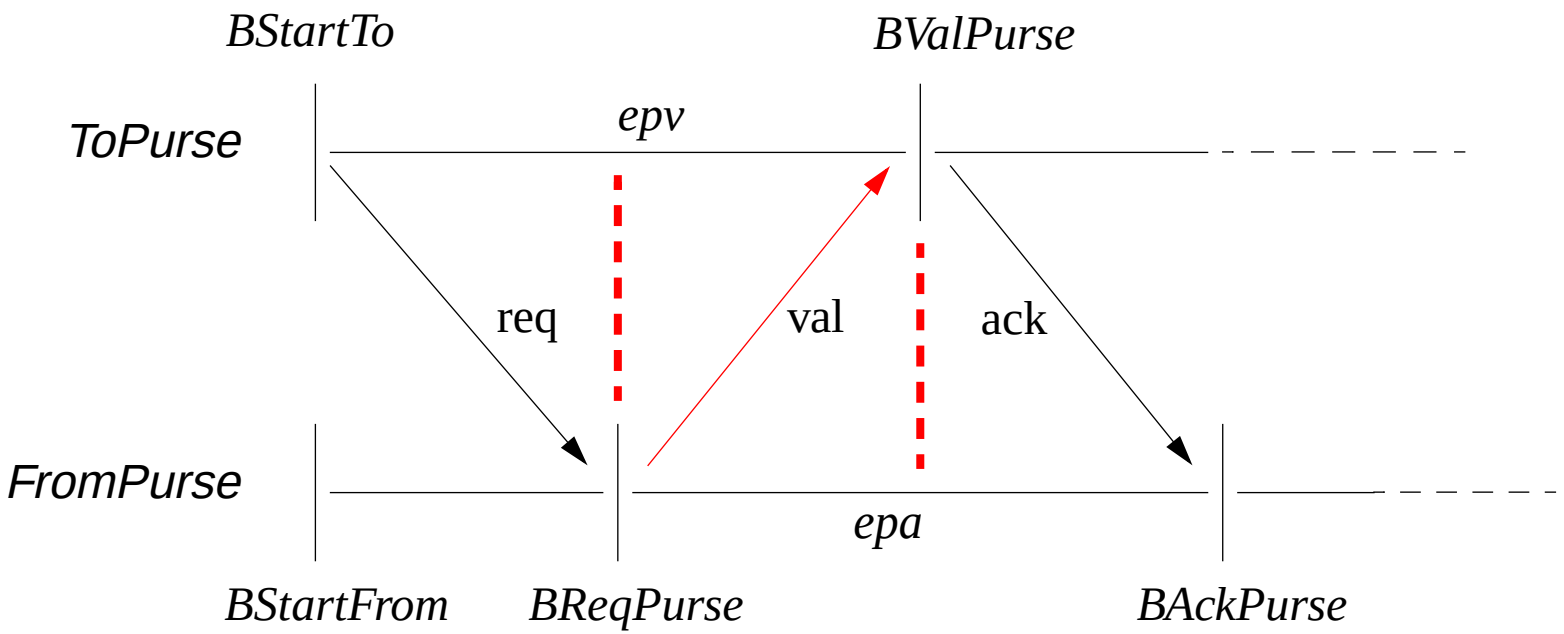
6. Balance Enquiry.

The balance enquiry operation interacts badly with resolution of nondeterminism during money transfer (concealed from PRG-126).

6. Balance Enquiry.

The balance enquiry operation interacts badly with resolution of nondeterminism during money transfer (concealed from PRG-126).

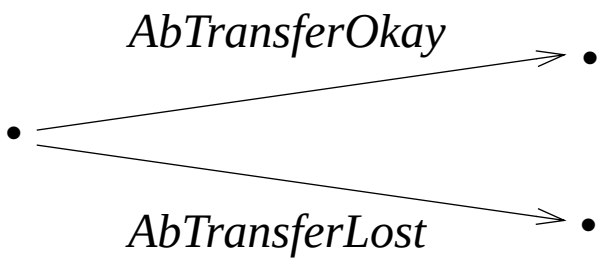
Unlike the other ‘retrenchment opportunities’, the Balance Enquiry Quandary requires consideration of the Mondex protocol as a whole.



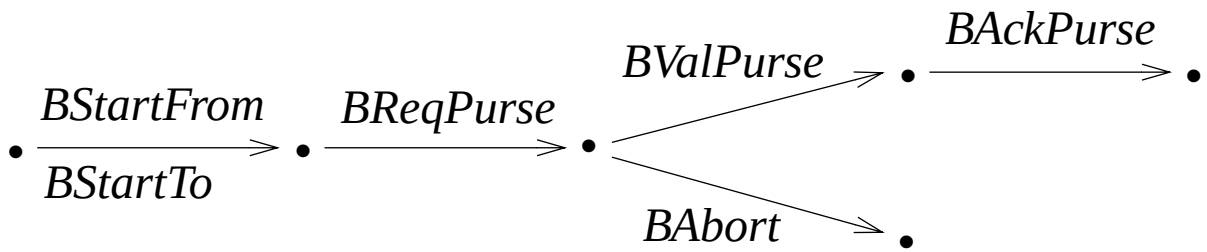
6. Balance Enquiry.

The balance enquiry operation interacts badly with resolution of nondeterminism during money transfer (concealed from PRG-126).

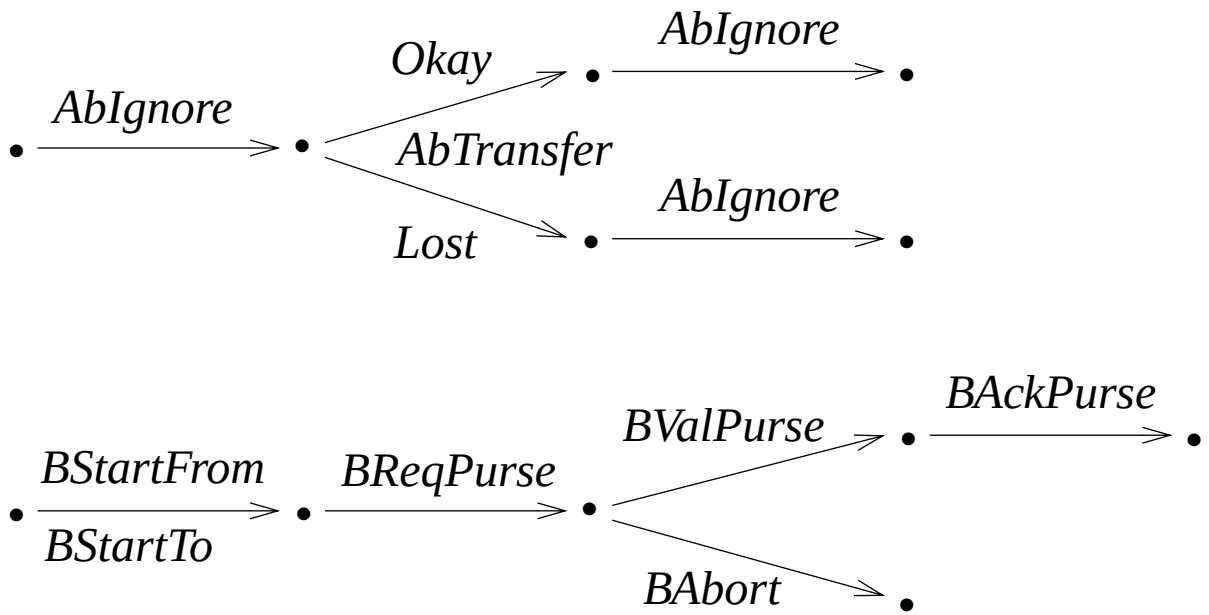
Abstract transaction:



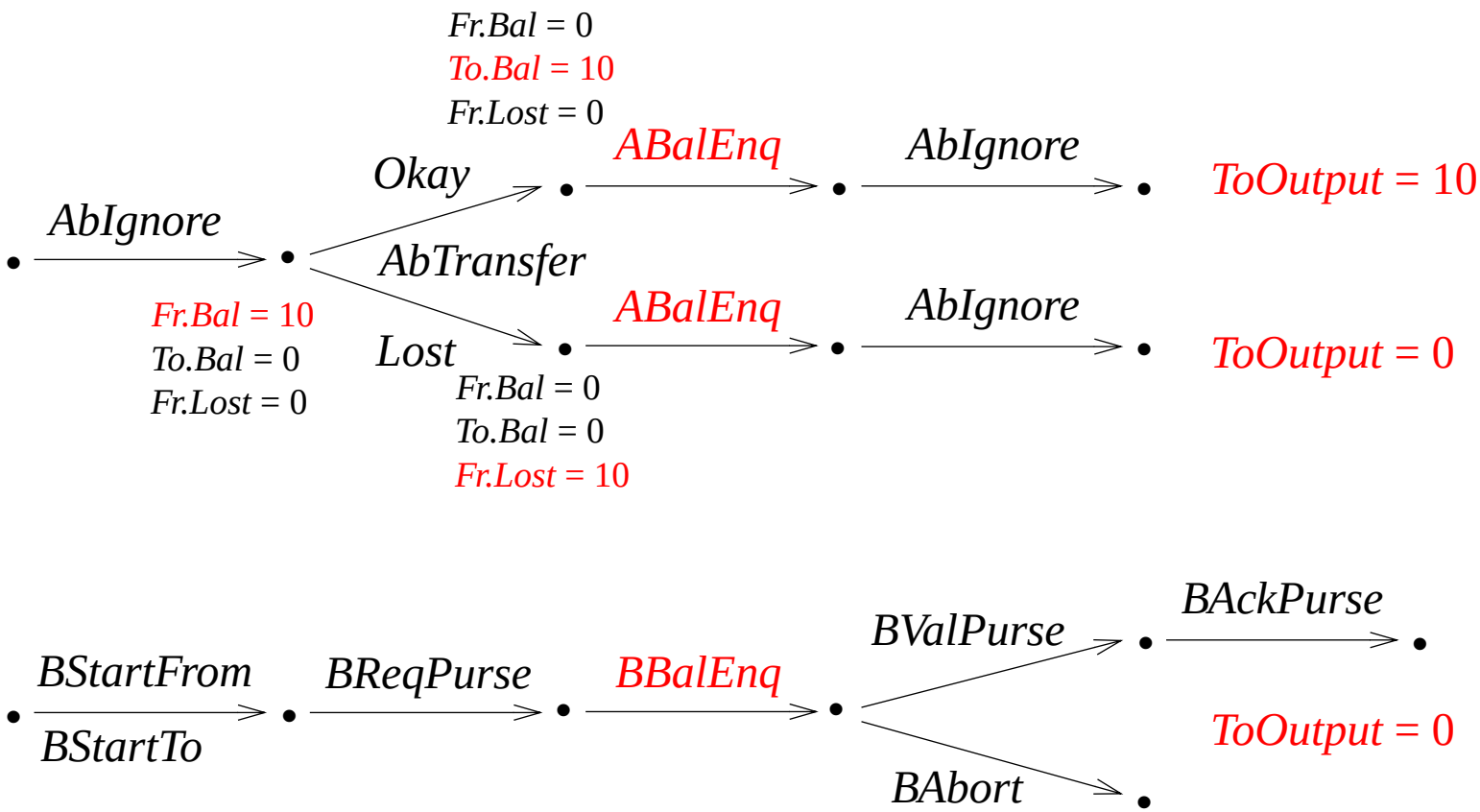
Between transaction (main paths):



The PRG-126 backwards refinement (*in a world of just two purses*):



What if there's a *BalanceEnquiry* between the Between transaction's *BReqPurse* and (*BValPurse* or *BAbort*) steps?



What sort of a problem is this ... and what to do about it?

What sort of a problem is this ... and what to do about it?

Various approaches.

We consider three.

Story 1: Backward (1,1) refinement and retrenchment

Is it a problem at all? ... Yes, but only just.

Story 1: Backward (1,1) refinement and retrenchment

Is it a problem at all? ... Yes, but only just.

Backward refinement PO (simplified):

$$G(u',v') \wedge Out_{Op}(o,p) \wedge Op_{\mathrm{C}}(v,j,v',p) \;\Rightarrow\; (\exists\, u,i \bullet Op_{\mathrm{A}}(u,i,u',o) \wedge G(u,v) \wedge In_{Op}(i,j))$$

Take G' valid, Out_{Op} is equality, Op_{C} skips ... so make Op_{A} skip ... Hooray!!

Story 1: Backward (1,1) refinement and retrenchment

Is it a problem at all? ... Yes, but only just.

Backward refinement PO (simplified):

$$G(u',v') \wedge Out_{Op}(o,p) \wedge Op_{\mathrm{C}}(v,j,v',p) \;\Rightarrow\; (\exists\, u,i \bullet Op_{\mathrm{A}}(u,i,u',o) \wedge G(u,v) \wedge In_{Op}(i,j))$$

Take G' valid, Out_{Op} is equality, Op_{C} skips ... so make Op_{A} skip ... Hooray!!

- Unfortunately it discharges the PO trivially in the $ATOOutput = 10$ case ...

Story 1: Backward (1,1) refinement and retrenchment

Is it a problem at all? ... Yes, but only just.

Backward refinement PO (simplified):

$$G(u',v') \wedge Out_{Op}(o,p) \wedge Op_{C}(v,j,v',p) \Rightarrow (\exists\, u,i \bullet Op_{A}(u,i,u',o) \wedge G(u,v) \wedge In_{Op}(i,j))$$

Take G' valid, Out_{Op} is equality, Op_{C} skips ... so make Op_{A} skip ... Hooray!!

- Unfortunately it discharges the PO trivially in the $AToOutput = 10$ case ...
- i.e. the $AToOutput = 10$ case is just ignored as being of no significance.

Story 1: Backward (1,1) refinement and retrenchment

Is it a problem at all? ... Yes, but only just.

Backward refinement PO (simplified):

$$G(u',v') \wedge Out_{Op}(o,p) \wedge Op_{C}(v,j,v',p) \Rightarrow (\exists\, u,i \bullet Op_{A}(u,i,u',o) \wedge G(u,v) \wedge In_{Op}(i,j))$$

Take G' valid, Out_{Op} is equality, Op_{C} skips ... so make Op_{A} skip ... Hooray!!

- Unfortunately it discharges the PO trivially in the $AToOutput = 10$ case ...
- i.e. the $AToOutput = 10$ case is just ignored as being of no significance.

To bring the $AToOutput = 10$ case within the scope of the PO antecedent, we need to make $Out_{Op}(o,p)$ trivial (done in PRG-126). ... Now what?

Now the PO still discharges OK, but it misses the point! The output discrepancy is invisible! This addresses the system requirements rather poorly ...

Story 1: Backward (1,1) refinement and retrenchment

Is it a problem at all? ... Yes, but only just.

Backward refinement PO (simplified):

$$G(u',v') \wedge Out_{Op}(o,p) \wedge Op_{C}(v,j,v',p) \Rightarrow (\exists\, u,i \bullet Op_{A}(u,i,u',o) \wedge G(u,v) \wedge In_{Op}(i,j))$$

Take G' valid, Out_{Op} is equality, Op_{C} skips ... so make Op_{A} skip ... Hooray!!

- Unfortunately it discharges the PO trivially in the $AToOutput = 10$ case ...
- i.e. the $AToOutput = 10$ case is just ignored as being of no significance.

To bring the $AToOutput = 10$ case within the scope of the PO antecedent, we need to make $Out_{Op}(o,p)$ trivial (done in PRG-126). ... Now what?

Now the PO still discharges OK, but it misses the point! The output discrepancy is invisible! This addresses the system requirements rather poorly ...

Only in output finalisation is there a demand that the outputs match. This fails.

Story 1: Backward (1,1) refinement and retrenchment

The retrenchment story uses a backward retrenchment PO
(the retrenchment analogue of backward refinement) to capture the situation:

$$((G(u',v') \wedge O_{Op}(o,p;u',v')) \vee C_{Op}(u',v',o,p)) \wedge Op_{\mathrm{C}}(v,j,v',p) \Rightarrow$$
$$(\exists\, u,i \bullet Op_{\mathrm{A}}(u,i,u',o) \wedge G(u,v) \wedge P_{Op}(i,j,u,v;u',v',o,p))$$

Story 1: Backward (1,1) refinement and retrenchment

The retrenchment story uses a backward retrenchment PO
(the retrenchment analogue of backward refinement) to capture the situation:

$$((G(u',v') \wedge O_{Op}(o,p;u',v')) \vee C_{Op}(u',v',o,p)) \wedge Op_{C}(v,j,v',p) \Rightarrow$$
$$(\exists\, u,i \bullet Op_{A}(u,i,u',o) \wedge G(u,v) \wedge P_{Op}(i,j,u,v;u',v',o,p))$$

ToPurse output relation for *BalEnq* :

$$(\exists\, Bfromstatus' , Bfrompaydetails' \bullet$$
$$(\neg(Btostatus' = epv \wedge Bfromstatus' = epa \wedge Bfrompaydetails' = Btopaydetails') \wedge$$
$$AToOutput! = BToOutput!) \vee$$
$$((Btostatus' = epv \wedge Bfromstatus' = epa \wedge Bfrompaydetails' = Btopaydetails') \wedge$$
$$AToOutput! - BToOutput! = Atobalance - Btobalance = Btopaydetails'))$$

ToPurse concedes relation and within relation for *BalEnq* :

false and $AToInput? = BToInput?$

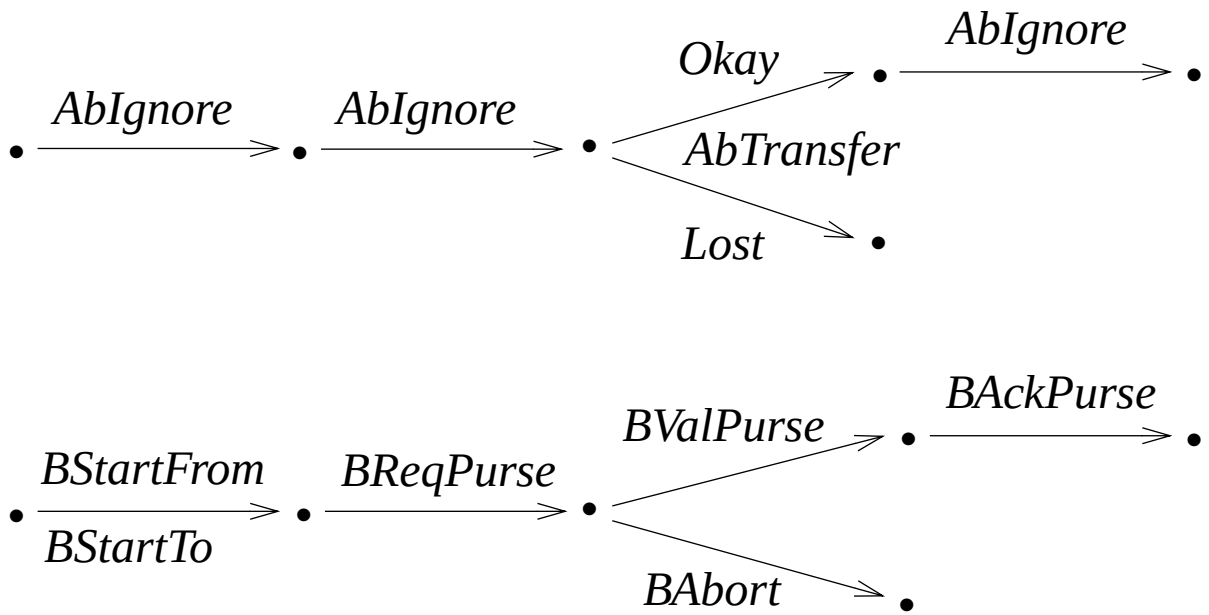
Story 2: Forward (1,1) refinement and retrenchment

Until recently it was believed (by most) that backwards refinement was *needed* for the Mondex A –> B refinement (when done in a (1,1) way). This is now known not to be the case.

Story 2: Forward (1,1) refinement and retrenchment

Until recently it was believed (by most) that backwards refinement was *needed* for the Mondex A –> B refinement (when done in a (1,1) way). This is now known not to be the case.

A forwards refinement (*in a world of just two purses*):



Key idea

In the Mondex protocol, by far the most subtle aspect is the $BAbort$ operation.

Key idea

In the Mondex protocol, by far the most subtle aspect is the *BAbort* operation.

To get a forward refinement, you need to decompose *BAbort*.

$$
\begin{aligned}
&BAbort = \\
&\quad BAbortBenign \ \vee \\
&\quad BAbortStart \ \vee \\
&\quad BAbortComplete
\end{aligned}
$$

Key idea

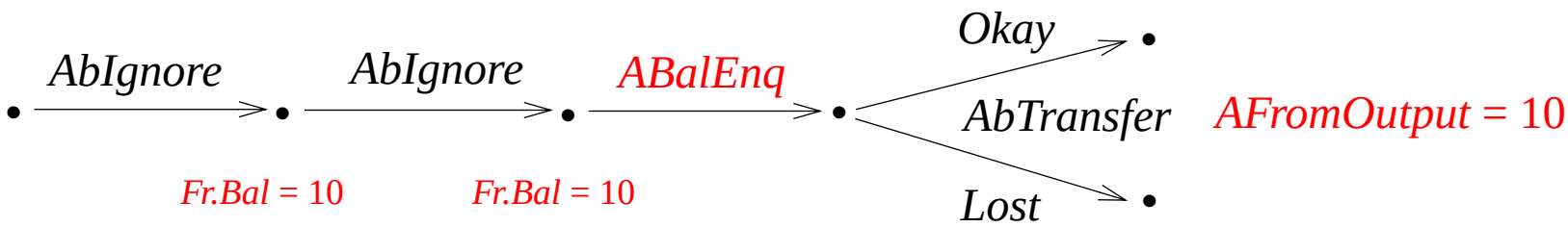
In the Mondex protocol, by far the most subtle aspect is the *BAbort* operation.

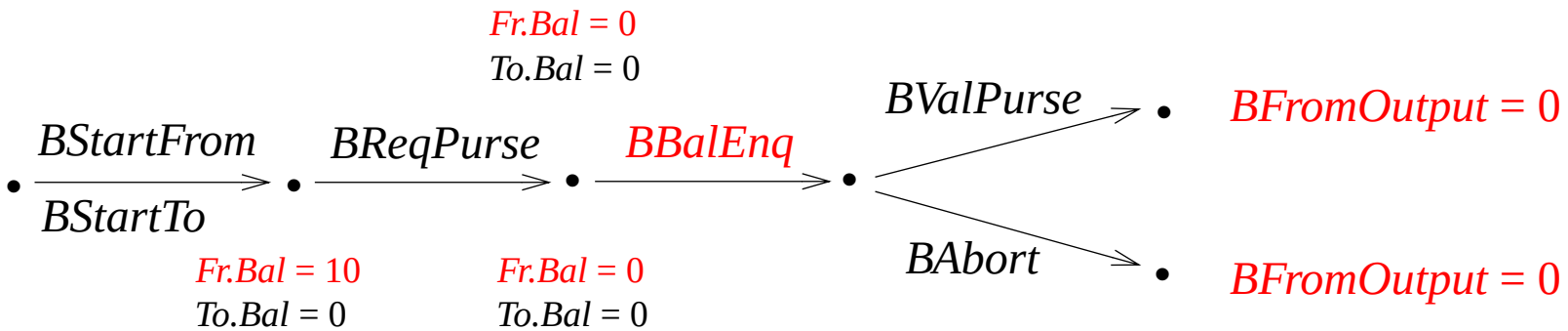
To get a forward refinement, you need to decompose *BAbort*.

$$
\begin{aligned}
&BAbort = \\
&\quad BAbortBenign \ \vee \\
&\quad BAbortStart \ \vee \\
&\quad BAbortComplete
\end{aligned}
$$

- Distinction between *BAbortBenign* and other cases is constructive.
- Distinction between *BAbortStart* and *BAbortComplete* is *nonconstructive*.

What if there's a *FromPurse* *BalanceEnquiry* between the Between transaction's *BReqPurse* and (*BValPurse* or *BAbort*) steps?





Points to note

In the forward case the failure of refinement is more incisive:

Forward refinement PO:

$$G(u,v) \wedge In_{Op}(i,j) \wedge Op_{\mathrm{C}}(v,j,v',p) \;\Rightarrow\; (\exists\, u',o \bullet Op_{\mathrm{A}}(u,i,u',o) \wedge G(u',v') \wedge Out_{Op}(o,p))$$

Take G valid, In_{Op} is equality, Op_{C} skips ... so make Op_{A} skip ... OOPS!!

Points to note

In the forward case the failure of refinement is more incisive:

Forward refinement PO:

$$G(u,v) \wedge In_{Op}(i,j) \wedge Op_{\mathrm{C}}(v,j,v',p) \;\Rightarrow\; (\exists\, u',o \bullet Op_{\mathrm{A}}(u,i,u',o) \wedge G(u',v') \wedge Out_{Op}(o,p))$$

Take G valid, In_{Op} is equality, Op_{C} skips ... so make Op_{A} skip ... OOPS!!

- This time the output relation $Out_{Op}(o,p)$ fails (provided it is identity).

And it's the operation PO itself that fails, not just finalisation.

Points to note

In the forward case the failure of refinement is more incisive:

Forward refinement PO:

$$G(u,v) \wedge In_{Op}(i,j) \wedge Op_{\mathrm{C}}(v,j,v',p) \;\Rightarrow\; (\exists\, u',o \bullet Op_{\mathrm{A}}(u,i,u',o) \wedge G(u',v') \wedge Out_{Op}(o,p))$$

Take G valid, In_{Op} is equality, Op_{C} skips ... so make Op_{A} skip ... OOPS!!

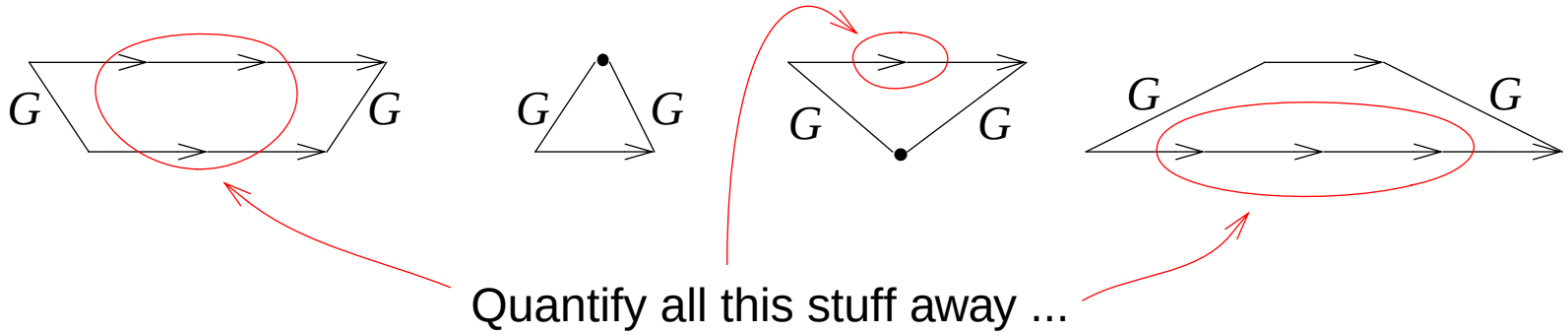
- This time the output relation $Out_{Op}(o,p)$ fails (provided it is identity).

And it's the operation PO itself that fails, not just finalisation.

The problem can be overcome by a (forward) retrenchment, with output relation essentially identical to that for the backward case.

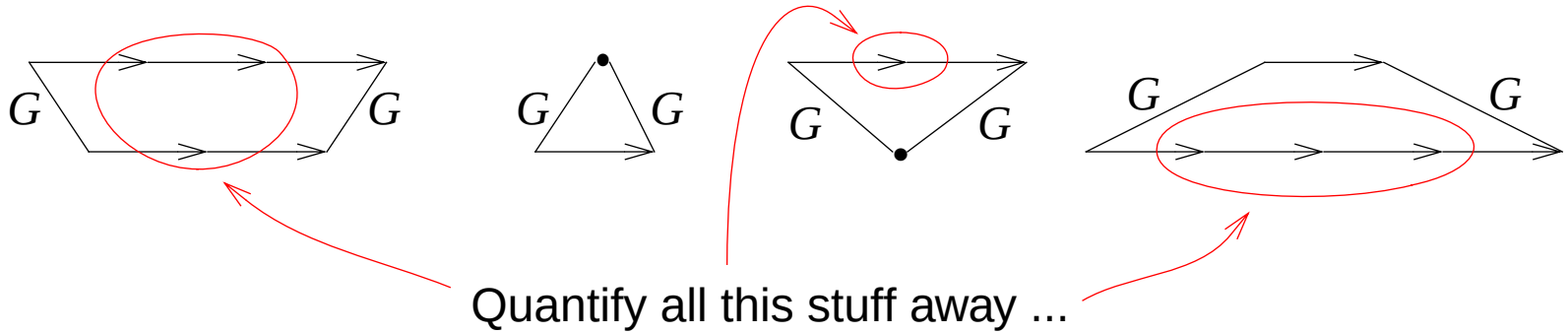
Story 3: Generalised forward refinement

Generalised forward refinement gets away from the (1,1) refinement paradigm. The following are typical primitive refinement building blocks, which are stitched together to produce simulations:



Story 3: Generalised forward refinement

Generalised forward refinement gets away from the (1,1) refinement paradigm. The following are typical primitive refinement building blocks, which are stitched together to produce simulations:



The key problem is to identify enough building blocks to enable *any* concrete run to be simulated.

Story 3: Generalised forward refinement

Key building blocks ... (m,n) pairs ... for the generalised forward refinement are constructed as follows.

Story 3: Generalised forward refinement

Key building blocks ... (m,n) pairs ... for the generalised forward refinement are constructed as follows.

Runs of the concrete protocol are prefixes of the strings of operation names generated by the following regular expression (modulo some bookkeeping):

$$(BStartFrom\ ;\ BStartTo + BStartTo\ ;\ BStartFrom)\ ;\ BReqPurse\ ;\ BValPurse\ ;\ BAckPurse$$

... optionally followed by a $BAbort$ for either or both purses.

Story 3: Generalised forward refinement

Key building blocks ... (m,n) pairs ... for the generalised forward refinement are constructed as follows.

Runs of the concrete protocol are prefixes of the strings of operation names generated by the following regular expression (modulo some bookkeeping):

$$(BStartFrom\ ;\ BStartTo + BStartTo\ ;\ BStartFrom)\ ;\ BReqPurse\ ;\ BValPurse\ ;\ BAckPurse$$

... optionally followed by a $BAbort$ for either or both purses.

Into these runs, one can interleave arbitrary numbers of $BBalEnq$ operations (for both the *From* and the *To* purse), to give the set of concrete sequences to take into account in building (m,n) pairs.

Story 3: Generalised forward refinement

Each concrete sequence is simulated by an abstract sequence according to the following rules:

- If the concrete run succeeded, the abstract sequence contains an *AbTransferOkay*
  ElseIf the concrete run failed, the abstract sequence contains an *AbTransferLost*
- If the concrete run contained k *BToBalEnq* operations before the *BValPurse* and h *BToBalEnq* operations after it, then the abstract sequence contains k *AToBalEnq* operations before the *AbTransfer* and h *AToBalEnq* operations after it (with obvious defaults).
- If the concrete run contained k *BFromBalEnq* operations before the *BReqPurse* and h *BFromBalEnq* operations after it, then the abstract sequence contains k *AFromBalEnq* operations before the *AbTransfer* and h *AFromBalEnq* operations after it (with obvious defaults).

Compare and Contrast

The mismatch between Abstract and Between worlds is fundamentally due to a clash between abstract atomicity and concrete non-atomicity.

Backward (1,1) refinement:

- Protocol critical section start points synchronised, but abstract endpoint too early, hence balance mismatch on arrival side.

Forward (1,1) refinement:

- Protocol critical section endpoints synchronised, but abstract start point too late, hence balance mismatch on departure side.

Generalised forward refinement:

- Protocol critical section start points and endpoints synchronised via generalised simulation shape, so all mismatches concealed, regardless of protocol outcome.

Compare and Contrast

The mismatch between Abstract and Between worlds is fundamentally due to a clash between abstract atomicity and concrete non-atomicity.

Backward (1,1) refinement:

- Protocol critical section start points synchronised, but abstract endpoint too early, hence balance mismatch on arrival side.

Forward (1,1) refinement:

- Protocol critical section endpoints synchronised, but abstract start point too late, hence balance mismatch on departure side.

Generalised forward refinement:

- Protocol critical section start points and endpoints synchronised via generalised simulation shape, so all mismatches concealed, regardless of protocol outcome.

N.B. All three approaches utilise different retrieve relations etc.

The PRG-126 story

PRG-126 is based on the backwards refinement. So Story 1 applies.

Can’t use an equality output relation since an important case goes out of scope.

So trivial output relation used.

So *BalanceEnquiry* operation becomes trivial ...
(i.e. a read-only operation for which any old outputs are deemed OK!).

The PRG-126 story

PRG-126 is based on the backwards refinement. So Story 1 applies.

Can’t use an equality output relation since an important case goes out of scope.

So trivial output relation used.

So *BalanceEnquiry* operation becomes trivial ...
(i.e. a read-only operation for which any old outputs are deemed OK!).

It looks crazy (unless you understand why in detail) ... so it was left out of PRG-126.

Can check that nothing is amiss by finalising immediately after the BalanceEnquiry operations and confirming that states are behaving well.

6. Conclusions.

The retrenchment analyses enabled issues which fell outside the scope of the refinement treatment to be nevertheless analysed formally, in a manner that blended smoothly with the refinement treatment.

In hindsight, some (if not all) of these issues proved to be treatable via refinement after all ... the needed refinements were *engineered* out of the retrenchment analyses. A good thing.

The retrenchment analyses led to the discovery of the (1,1) forward refinement. Another benefit.

All in all, a thorough vindication of retrenchment.

Atomicity. (W.I.P.)

We can construct yet another route from the atomic model to the protocol model.

The A model:

Abworld

$$Afrombal, Afromlost, Atobal, Atolost : \mathbb{N}$$

AbTransferOkay

$$\Delta Abworld$$
$$Avalue? : \mathbb{N}$$

$$0 < Avalue? \leq Afrombal$$
$$Afrombal' = Afrombal - Avalue?$$
$$Atobal' = Atobal + Avalue?$$
$$Afromlost' = Afromlost$$
$$Atolost' = Atolost$$

AbTransferLost

$$\Delta Abworld$$
$$Avalue? : \mathbb{N}$$

$$0 < Avalue? \leq Afrombal$$
$$Afrombal' = Afrombal - Avalue?$$
$$Atobal' = Atobal$$
$$Afromlost' = Afromlost + Avalue?$$
$$Atolost' = Atolost$$

The AA model:

AAbworld

$AAfrombal : \mathbb{N}$
$AAfromlost : \mathbb{N}$
$AAtobal : \mathbb{N}$
$AAtolost : \mathbb{N}$
$AAval : \mathbb{N}$

AAbTransferStart

$\Delta AAbworld$
$AAvalue? : \mathbb{N}$

$AAval = 0$
$0 < AAvalue? \leq AAfrombal$
$AAval' = AAvalue?$
$AAfrombal' = AAfrombal - AAvalue?$
RestSame.....

AAbTransferOkay

$\Delta AAbworld$

$AAval > 0$
$AAval' = 0$
$AAtobal' = AAtobal + AAval$
RestSame.....

AAbTransferLost

$\Delta AAbworld$

$AAval > 0$
$AAval' = 0$
$AAfromlost' = AAfromlost + AAval$
RestSame.....

Now forward refine:

- *AAbTtansferStart* to *BReqPurse*
- *AAbTtansferOkay* to *BValPurse*
- *AAbTtansferLost* to *BAbort*

..... etc.

The Retrenchment Homepage

See the Retrenchment Homepage: Google knows where it is.

`http://www.cs.man.ac.uk/retrenchment`

- Bibliographical pointers.
- The Retrenchment tutorial.
- Etc.