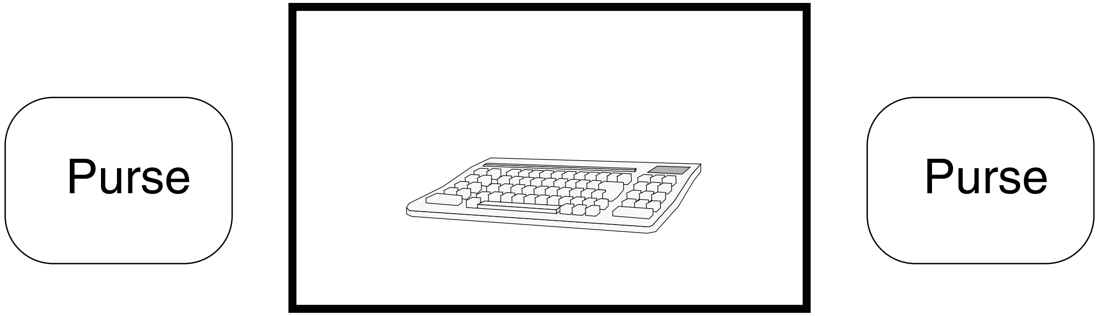
Retrenchment
and
Mondex

R. Banach, C. Jeske, CS Dept., University of Manchester, UK
M. Poppleton, ECS Dept., University of Southampton, UK
S. Stepney, CS Dept., University of York, UK

Contents:

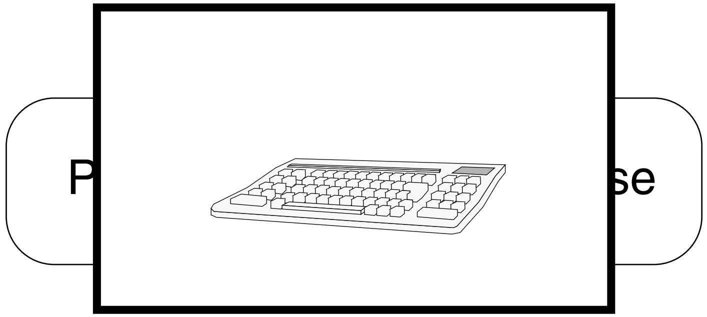
1. Mondex ... salient aspects of the model(s).

Cartoon:



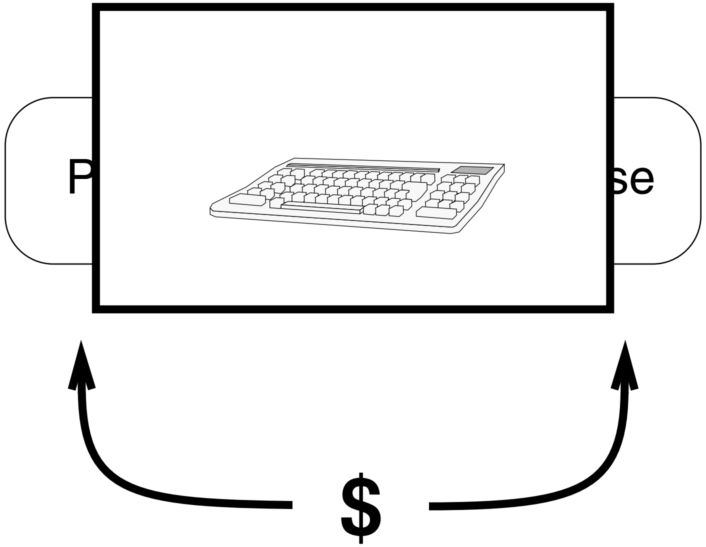
1. Mondex ... salient aspects of the model(s).

Cartoon:



1. Mondex ... salient aspects of the model(s).

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 stay in favour of the bank .....

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.

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 ...

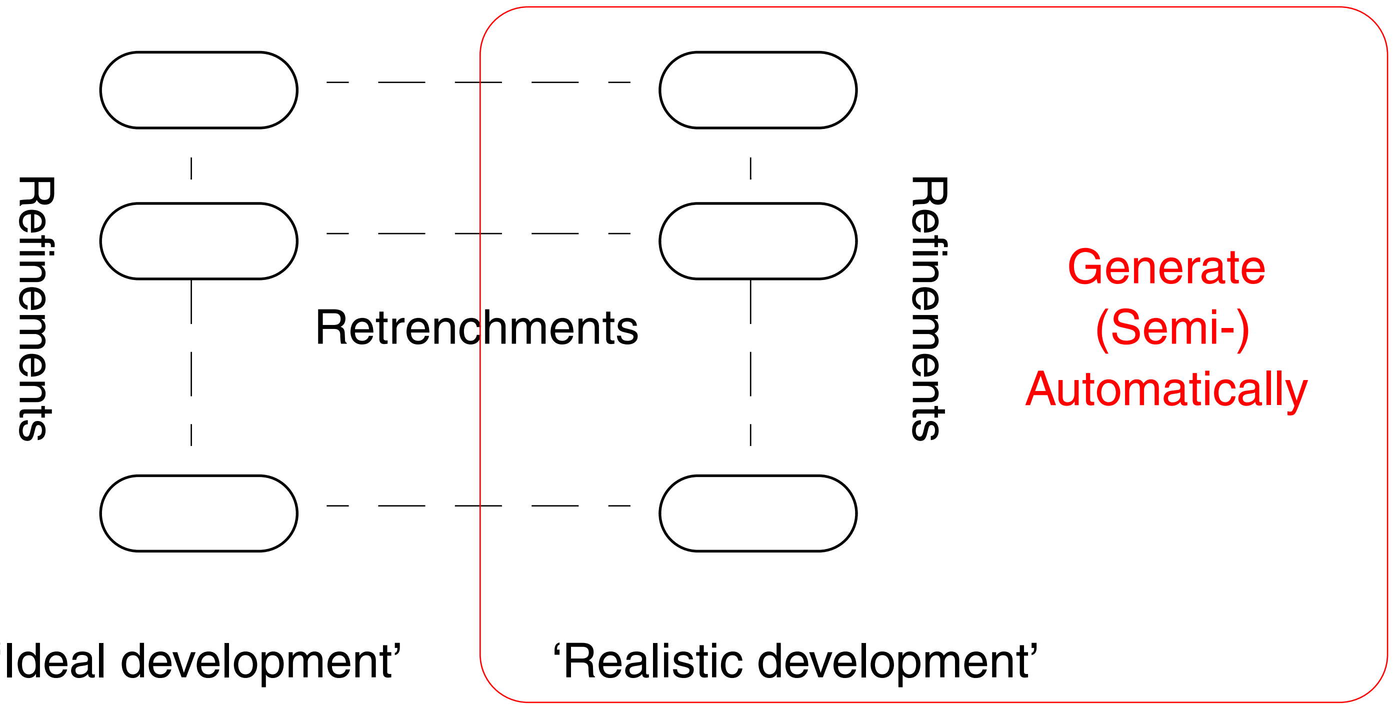
In fact, the Mondex refinement glossed over a number of issues, that were carefully omitted from the public documentation.

- 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).

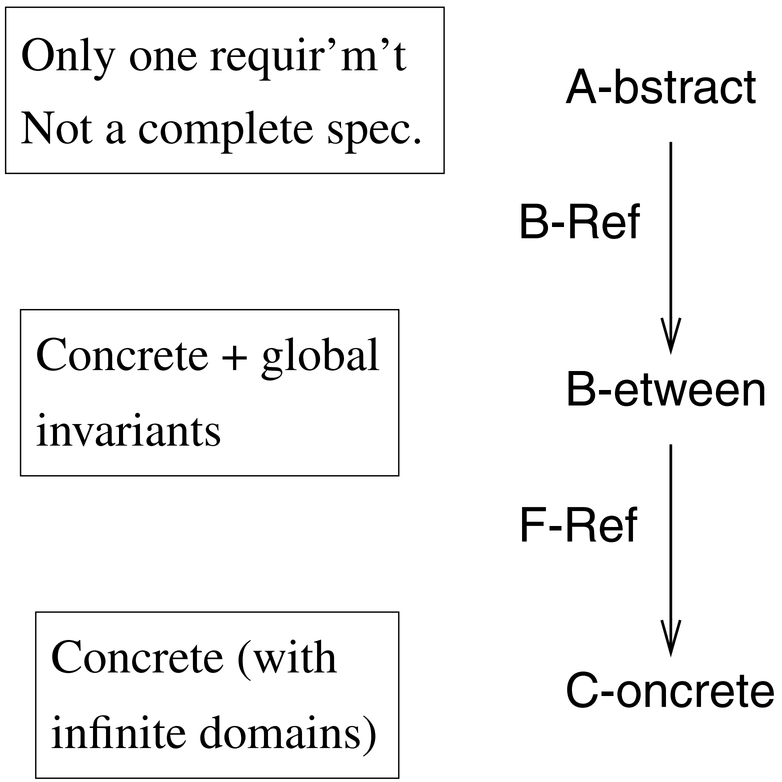
All of these can be addressed with retrenchment.

The Tower Pattern

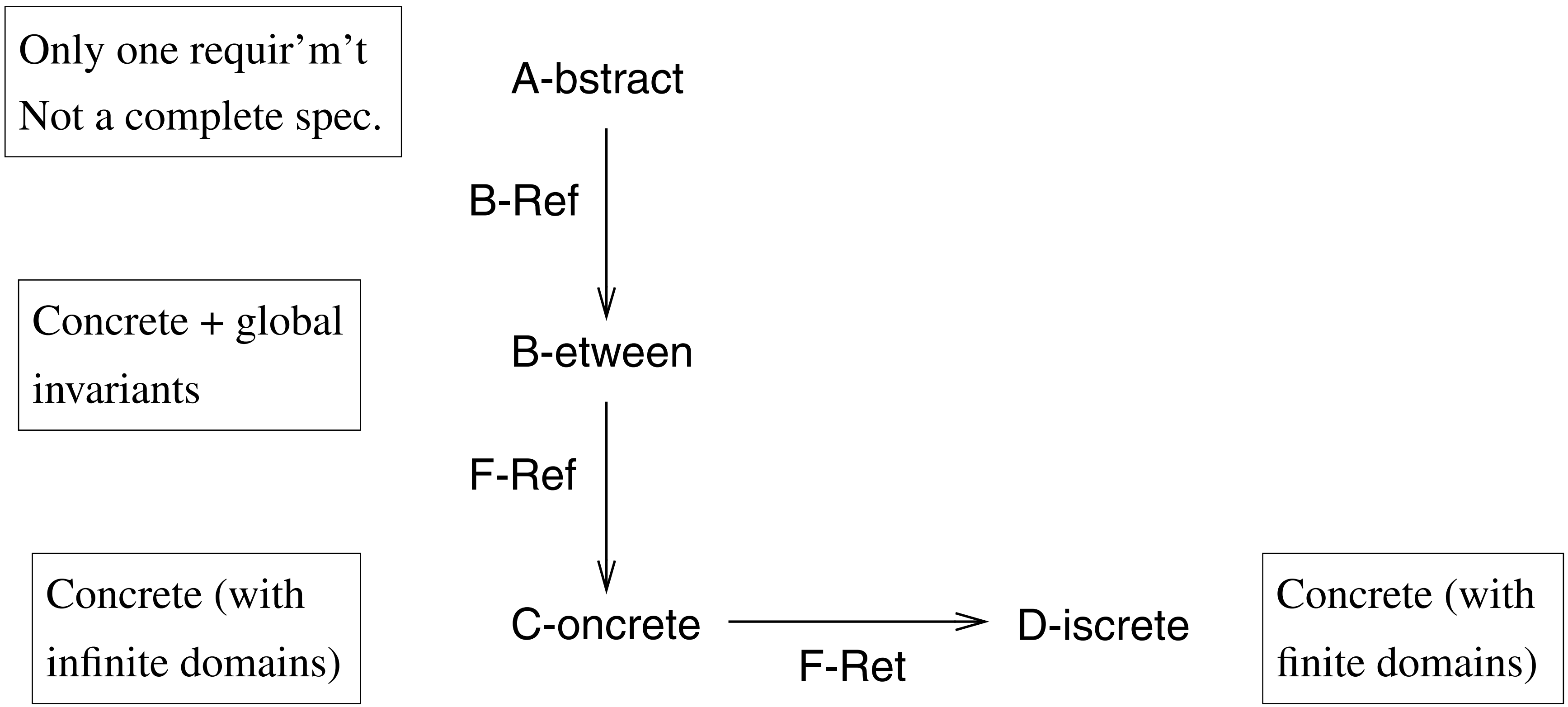
Mostly, the ‘retrenchment opportunities’ just identified can be dealt with using the Tower Pattern. In the case of Mondex, since an idealised refinement development is already available, the strategy is bottom up.



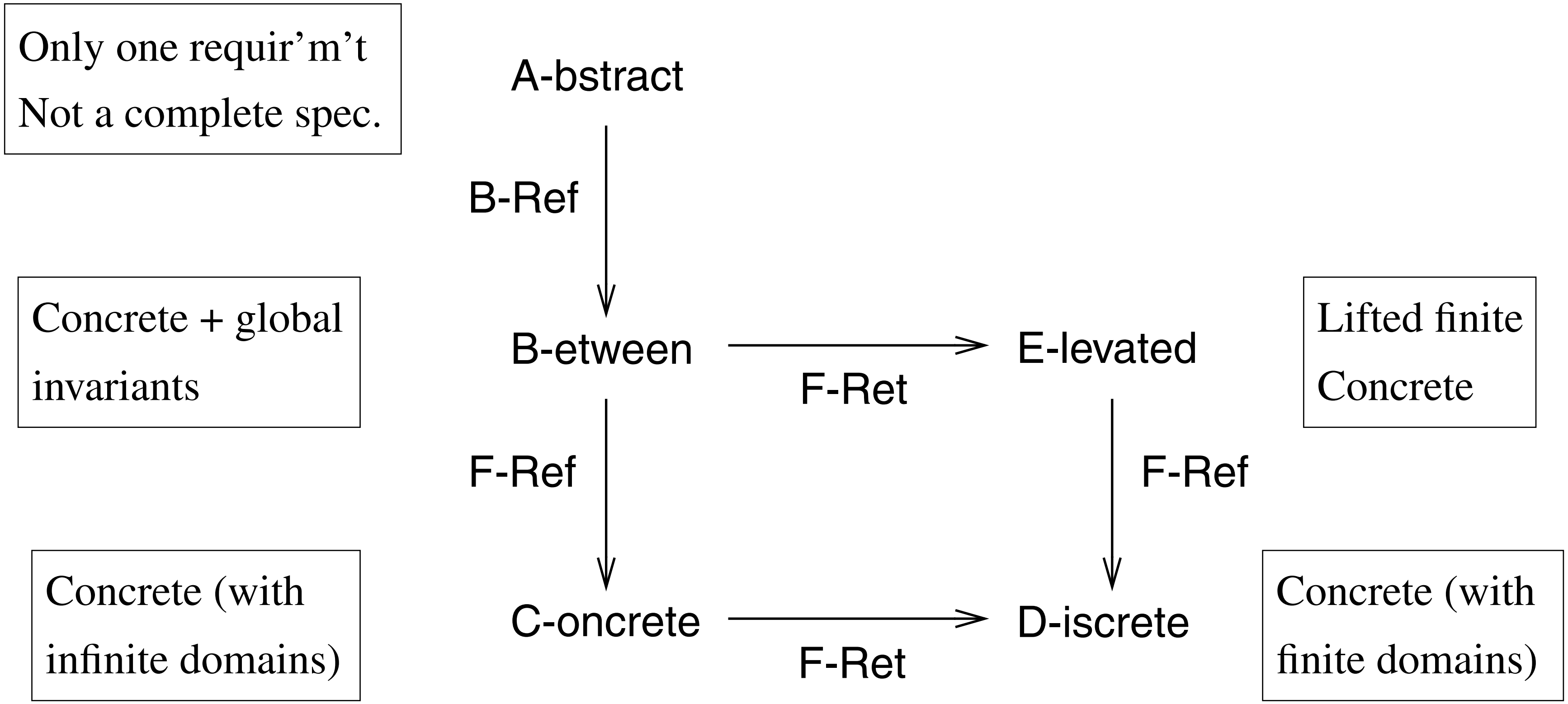
Model Architecture



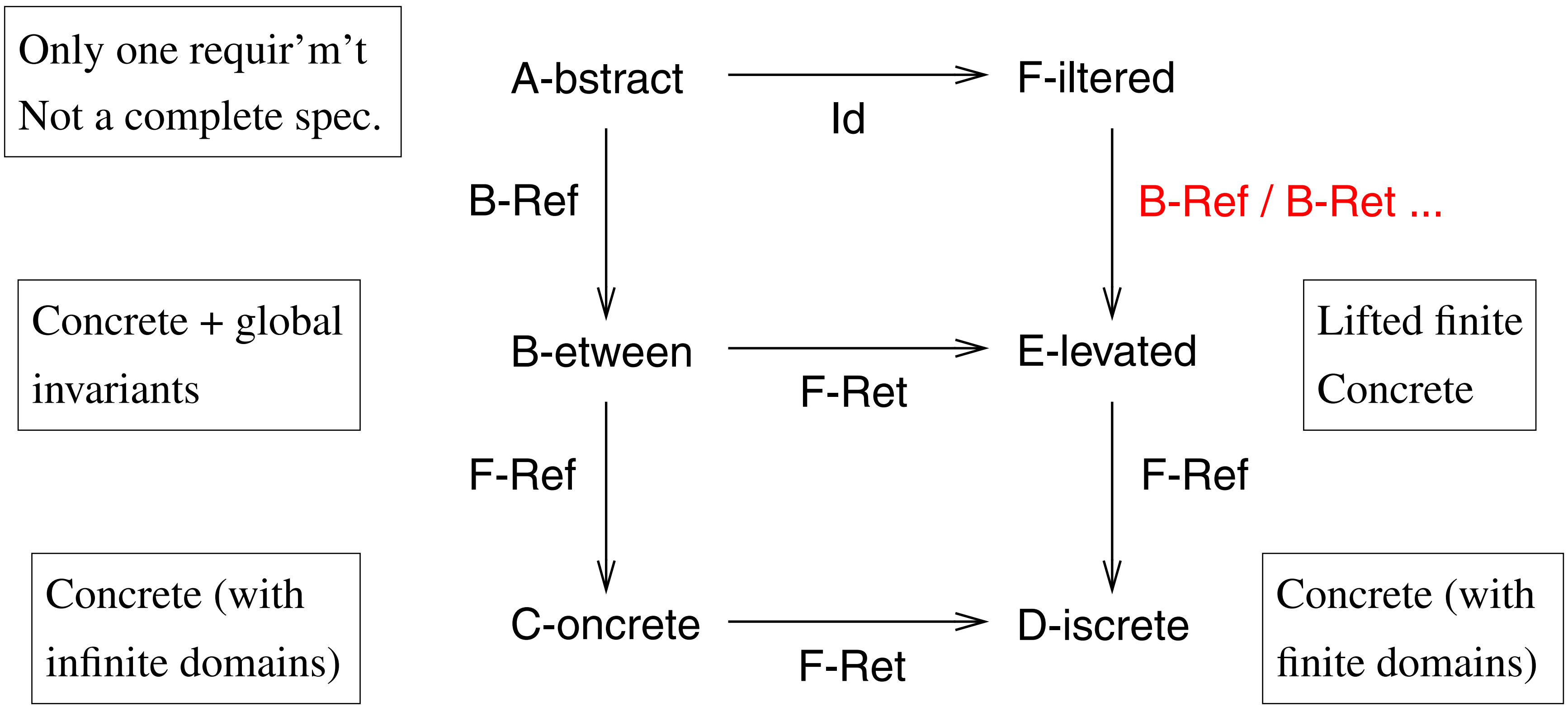
Model Architecture



Model Architecture



Model Architecture



2. Sequence number.

Each concrete purse has a sequence number *nextSeqNo*

Concrete model: $CnextSeqNo : \mathbb{N}$

In reality (and in Discrete model): $DnextSeqNo : [0 \,..\, \mathrm{BIGNUM}]$ where BIGNUM is T.B.D.

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

We will model $[0 \,..\, \mathrm{BIGNUM}]$ 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 $DnextSeqNo' \leq \mathrm{BIGNUM}$

2. Sequence number.

Each concrete purse has a sequence number *nextSeqNo*

Concrete model: $CnextSeqNo : \mathbb{N}$

In reality (and in Discrete model): $DnextSeqNo : [0 .. \text{BIGNUM}]$ where BIGNUM is T.B.D.

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

We will model $[0 .. \text{BIGNUM}]$ 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 $DnextSeqNo' \leq \text{BIGNUM}$

Operations using sequence number:

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

Minimally invasive D model ... per purse

C model

$$
\begin{array}{l}
\textit{CIncreasePurseOkay} \\
\hline
\Delta CConPurse \\
Cm?, Cm! : CMESSAGE \\
\hline
\Xi CConPurseIncrease \\
CnextSeqNo' \geq CnextSeqNo \\
Cm! = \bot
\end{array}
$$

D model

$$
\begin{array}{l}
\textit{DIncreasePurseOkay} \\
\hline
\Delta DConPurse \\
Dm?, Dm! : DMESSAGE \\
\hline
\Xi DConPurseIncrease \\
\\
(DnextSeqNo < \mathrm{BIGNUM} \Rightarrow \\
DnextSeqNo' \geq DnextSeqNo \\
Dm! = \bot) \\
\\
(DnextSeqNo = \mathrm{BIGNUM} \Rightarrow \\
DnextSeqNo' = DnextSeqNo \\
Dm! = DpurseBlocked\ Dname)
\end{array}
$$

Refinement case (first disjunct)

Exceptional case (second disjunct)

The D model can also be expressed in a more Z-like, if more verbose, way:

DIncreasePurseOkayInc

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

$$\Xi DConPurseIncrease$$
$$DnextSeqNo < \mathrm{BIGNUM}$$
$$DnextSeqNo' \geq DnextSeqNo$$
$$Dm! = \perp$$

DIncreasePurseOkayBlock

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

$$\Xi DConPurseIncrease$$
$$DnextSeqNo = \mathrm{BIGNUM}$$
$$DnextSeqNo' = DnextSeqNo$$
$$Dm! = DpurseBlocked\ Dname$$

The D model can also be expressed in a more Z-like, if more verbose, way:

DIncreasePurseOkayInc

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

$$\Xi DConPurseIncrease$$
$$DnextSeqNo < \mathrm{BIGNUM}$$
$$DnextSeqNo' \geq DnextSeqNo$$
$$Dm! = \bot$$

DIncreasePurseOkayBlock

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

$$\Xi DConPurseIncrease$$
$$DnextSeqNo = \mathrm{BIGNUM}$$
$$DnextSeqNo' = DnextSeqNo$$
$$Dm! = DpurseBlocked\ Dname$$

DIncreasePurseOkay

$$DIncreasePurseOkayInc \vee DIncreasePurseOkayBlock$$

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 $IncreasePurseOkay$:

$CConPurseIncrease'$ **"="** $DConPurseIncrease' \wedge CnextSeqNo' \geq DnextSeqNo' \wedge$
$Cm! = \perp \wedge Dm! = DpurseBlocked\ Dname$

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 $NAME$.

DConWorld

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

The D model ... framing schema

$$
\begin{array}{l}
\textbf{schema } \Phi DOp \\
\Delta DConWorld \\
\Delta DConPurse \\
Dm? \, , Dm! : DMESSAGE \\
Dname? : NAME \\
\hline
Dm? \in Dether \\
Dname? \in \mathrm{dom}\ DconAuthPurse \\
\theta DConPurse = DconAuthPurse\ Dname? \\
DconAuthPurse' = DconAuthPurse \oplus \{Dname? \mapsto \theta DConPurse'\} \\
Darchive' = Darchive \\
Dether' \subseteq Dether \cup \{Dm!\}
\end{array}
$$

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

ΦDOp

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

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

$DIncrease$

$DIgnore \vee (\exists\ \Delta DConPurse \bullet \Phi DOp \wedge DIncreasePurseOkay)$

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

- 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.

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:

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

Give up as soon as one purse concedes

Weak Promotion Retrieve relation:

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

Keep going while at least one purse still good

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

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

Keep track of which purses are still good

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:

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

Give up as soon as one purse concedes

Weak Promotion Retrieve relation:

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

Keep going while at least one purse still good

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

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

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:

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

World Level Within relation for *Increase* :

$$Cname? = Dname?$$
$$Cname? \in Dgood$$

World Level Output relation for *Increase* :

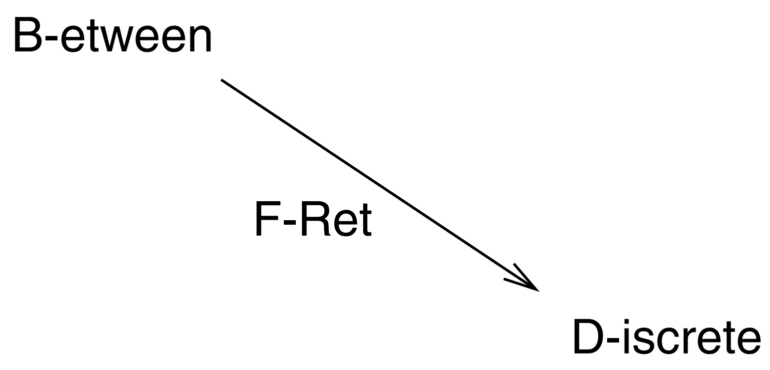
$$Dgood' = Dgood$$
$$Cm! = Dm!$$

World Level Concedes relation for *Increase* :

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

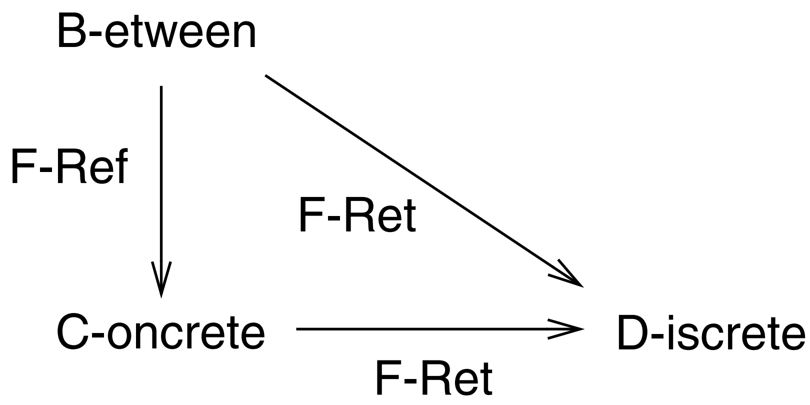
Lifting: E model

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



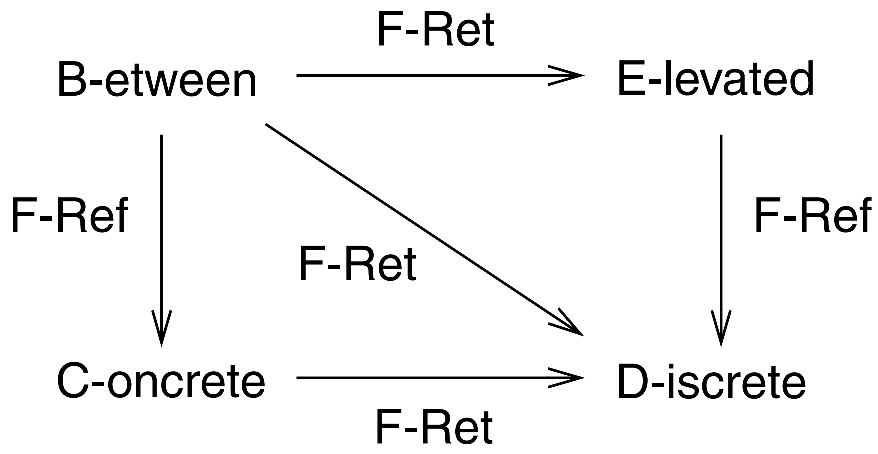
Lifting: E model

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



Lifting: E model

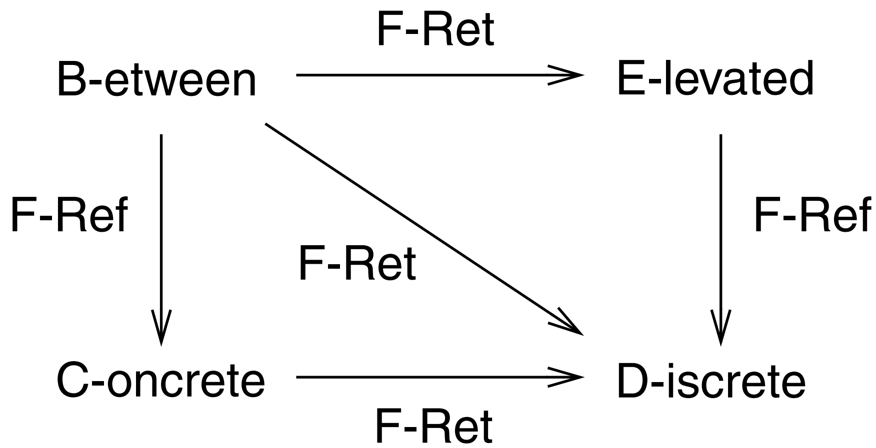
When there is a retrenchment such as from model B to model D, eg. via model C ...



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 ...



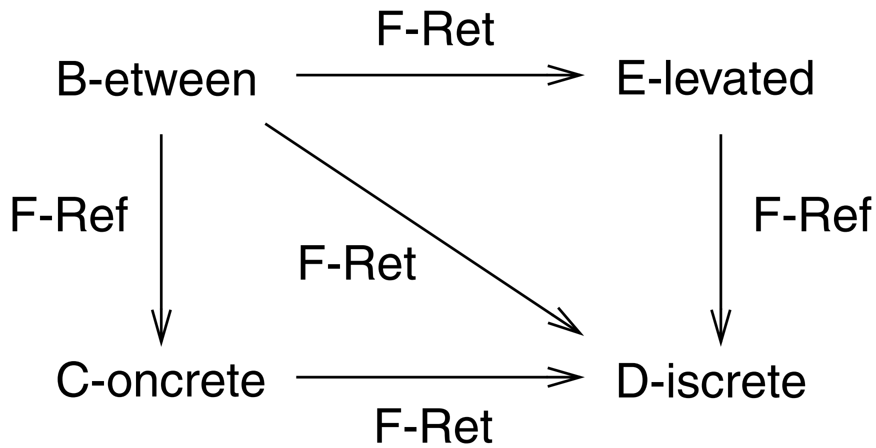
Then there is a lifting of the B-D retrenchment 'to the level of abstraction of B'.

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

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

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: $((u,w), (i,k), (u',w'), (o,q))$

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

N.B.
A simplified
special case.

The E model ... per purse

EprotoIncreasePurseOkay

$BIncreasePurseOkay \; \mathbf{;} \; \Delta DConPurse$
$Dm?, Dm! : DMESSAGE$

$(G \land P_{Increase} \land G' \land O_{Increase}) \lor (G \land P_{Increase} \land C_{Increase})$

... or, in detail ...

$$
\begin{array}{l}
\textit{EprotoIncreasePurseOkay} \\
\hline
\Delta BConPurse \; ; \; \Delta DConPurse \\
Bm?, Bm! : BMESSAGE \; ; \; Dm?, Dm! : DMESSAGE \\
\hline
(BConPurse \text{ “=” } DConPurse \land \\
\Xi BConPurseIncrease \land BnextSeqNo' \geq BnextSeqNo \land Bm! = \bot \land \\
BConPurse' \text{ “=” } DConPurse' \land Dm! = Bm!) \\
\quad \lor \\
(BConPurse \text{ “=” } DConPurse \land \\
\Xi BConPurseIncrease \land BnextSeqNo' \geq BnextSeqNo \land Bm! = \bot \land \\
BnextSeqNo' \geq DnextSeqNo' \land Dm! = DpurseBlocked\ Dname) \\
\hline
\end{array}
$$

Joint signature

Refinement case

C and D models part company

The E model ... per purse

Notes:

- The generic construction builds E data as pairs, eg.:
  $EConPurse = BConPurse \; ; \; DConPurse$
  $EMESSAGE = BMESSAGE \; ; \; DMESSAGE$ etc.
- The generic construction builds a model which is canonical within a class of similar factorisations. Thus the generated E model can be substituted by one with data isomorphic to the generated data.
- In our case, one can eg. substitute $(BnextSeqNo, DnextSeqNo)$ by an $EnextSeqNo$ which is a single natural number, but having appropriate properties at BIGNUM.
- Also one can discard excessive nondeterminism via refinement.

The E model ... per purse

Notes:

- The generic construction builds E data as pairs, eg.:
  EConPurse = *BConPurse* **;** *DConPurse*
  EMESSAGE = *BMESSAGE* **;** *DMESSAGE* etc.
- The generic construction builds a model which is canonical within a class of similar factorisations. Thus the generated E model can be substituted by one with data isomorphic to the generated data.
- In our case, one can eg. substitute (*BnextSeqNo*, *DnextSeqNo*) by an *EnextSeqNo* which is a single natural number, but having appropriate properties at BIGNUM.
- Also one can discard excessive nondeterminism via refinement.

... in the end we can reduce things to ...

EIncreasePurseOkay **“is as”** *DIncreasePurseOkay*

The E model ... raw world

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

EConWorld

$EconAuthPurse : NAME \rightarrowtail\!\!\!\!\!\!\!+\!\!\!\!\!\!\rightarrow EConPurse$
$Eether : \mathbb{P}\ EMESSAGE$
$Earchive : \mathbb{P}\ ELogbook$

$\forall\ n \in \mathrm{dom}\ EconAuthPurse \bullet (EconAuthPurse\ n).name = n$
$\forall\ nld \in Earchive \bullet first\ nld \in \mathrm{dom}\ EconAuthPurse$

EAuxWorld

$EConWorld$
Bunch of types of auxiliary variables

Bunch of definitions of auxiliary variables

The E model ... constrained world

ElevatedWorld
EAuxWorld
Bunch of constraints on the world

The constraints on the ether express things like:

- All *req* messages in the ether refer to authentic purses.
- There are no ‘future’ *req*, *val*, *ack*, messages in the ether.
- There are no ‘future’ *to* logs or ‘future’ *from* logs anywhere.
- etc.

In effect, the constraints capture needed aspects of the integrity of the protocol.

The E model ... constrained world

ElevatedWorld
EAuxWorld
Bunch of constraints on the world

The constraints on the ether express things like:

- All *req* messages in the ether refer to authentic purses.
- There are no ‘future’ *req*, *val*, *ack*, messages in the ether.
- There are no ‘future’ *to* logs or ‘future’ *from* logs anywhere.
- etc.

In effect, the constraints capture needed aspects of the integrity of the protocol.

D world forward refines the E world.

Building the E world (and the B world)

The B world arises as a bunch of individual purses and an ether.

- First, this is freely generated.
- Next, the constraints needed are imposed.

Obviously, the constrained B world arises as a forward refinement of the freely generated (raw) B world;

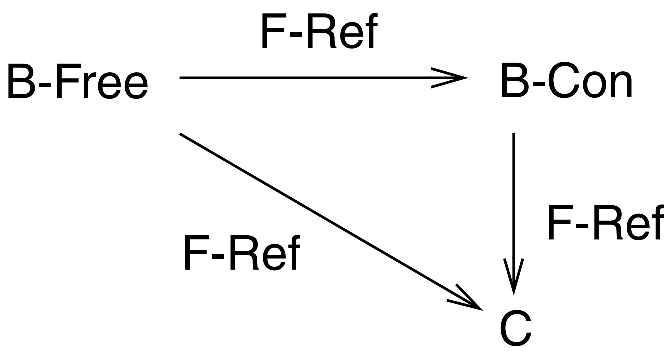
Building the E world (and the B world)

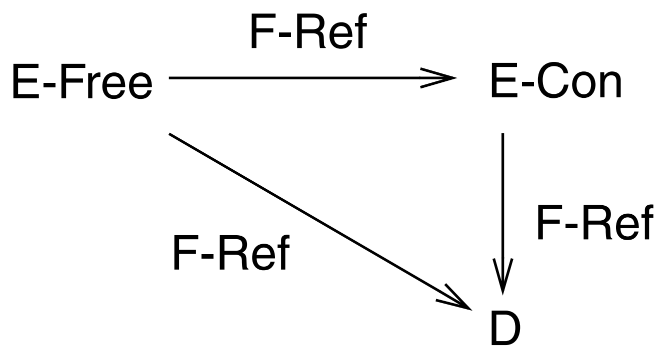
The B world arises as a bunch of individual purses and an ether.

- First, this is freely generated.
- Next, the constraints needed are imposed.

Obviously, the constrained B world arises as a forward refinement of the freely generated (raw) B world; and the C world is a forward refinement of the constrained B world, so it is also a forward refinement of the freely generated (raw) B world.

The same remarks apply to the E world and its relationship with the D world.





Filtering: F model

Since concrete outputs are *discarded* in PRG-126, and the only behaviour of the minimally invasive D model that differs from that of the C model is the production of "*DpurseBlocked Dname*?" messages, we are free to use the PRG-126 backward refinement to get back to the A model (as illustrated earlier).

Filtering: F model

Since concrete outputs are *discarded* in PRG-126, and the only behaviour of the minimally invasive D model that differs from that of the C model is the production of "*DpurseBlocked Dname?*" messages, we are free to use the PRG-126 backward refinement to get back to the A model (as illustrated earlier).

Then again ...

Filtering: F model

Since concrete outputs are *discarded* in PRG-126, and the only behaviour of the minimally invasive D model that differs from that of the C model is the production of "*DpurseBlocked Dname*?" messages, we are free to use the PRG-126 backward refinement to get back to the A model (as illustrated earlier).

Then again ...

Since outputs are *meaningful* in the Balance Enquiry operation, the "*DpurseBlocked Dname*?" messages must be taken account of explicitly. This is best done via a *backward retrenchment*.

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

The concession here would refer to the "*DpurseBlocked Dname*?" 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 $\mathrm{O}(10)$.

The determined shopper makes $\mathrm{O}(100)$ transactions a day. $\Delta nextSeqNo$ $\mathrm{O}(10^3)$ per day.

One year $\mathrm{O}(10^3)$ days. $\Delta nextSeqNo$ $\mathrm{O}(10^6)$ 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. $\Delta nextSeqNo$ $O(10^3)$ per day.

One year $O(10^3)$ days. $\Delta nextSeqNo$ $O(10^6)$ per year.

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

2. Say $\text{BIGNUM} = O(2^{32}) = O(4\times10^9)$ .... It's about 4000 years before we hit BIGNUM.

3. Say $\text{BIGNUM} = O(2^{64}) = O(16\times10^{18})$ .... It's about $O(16\times10^{12})$ years before we hit BIGNUM.

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. $\Delta nextSeqNo$ $O(10^3)$ per day.

One year $O(10^3)$ days. $\Delta nextSeqNo$ $O(10^6)$ per year.

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

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

3. Say $\text{BIGNUM} = O(2^{64}) = O(16\times10^{18})$ .... 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 $N(t)$:

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

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

$N(t)$ 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).

Stochastic aspects: likelihood of achieving concession

More sophisticated story

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

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

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

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

Stochastic aspects: likelihood of achieving concession

<u>More sophisticated story</u>

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

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

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

$N(t)$ 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 $N(\mathrm{BIGNUM})$.

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 $t = \mathrm{BIGNUM} = \mathrm{O}(2^{32})$, $\mu = \mathrm{O}(10)$, $N \to 4{\times}10^9/10$. At $\mathrm{O}(100)$ 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

<u>More sophisticated story</u>

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

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

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

$N(t)$ 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 $N(\mathrm{BIGNUM})$.

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 $t = \mathrm{BIGNUM} = \mathrm{O}(2^{32})$, $\mu = \mathrm{O}(10)$, $N \to 4{\times}10^9/10$. At $\mathrm{O}(100)$ 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.

Issues arising

- The (predominantly guard-driven) route was chosen to not disturb the validity of the remainder of the security proof, which we did not have time to redo. A better design would incorporate further purse states to express BIGNUM overflow, but would potentially entail much more work.

Issues arising

- The (predominantly guard-driven) route was chosen to not disturb the validity of the remainder of the security proof, which we did not have time to redo. A better design would incorporate further purse states to express BIGNUM overflow, but would potentially entail much more work.
- In fact *DIncreasePurseOkay* is a metaphor for a number of operations, some of which need to increase *DnextSeqNo* others not, some of which need to output nontrivial messages others not, etc. In reality some of these are modified nontrivially when you hit BIGNUM.

Issues arising

- The (predominantly guard-driven) route was chosen to not disturb the validity of the remainder of the security proof, which we did not have time to redo. A better design would incorporate further purse states to express BIGNUM overflow, but would potentially entail much more work.
- In fact *DIncreasePurseOkay* is a metaphor for a number of operations, some of which need to increase *DnextSeqNo* others not, some of which need to output nontrivial messages others not, etc. In reality some of these are modified nontrivially when you hit BIGNUM.
- Other operations, eg. *AbortPurseOkay*, *StartFromPurseEafromOkay*, *StartToPurse-EafromOkay* do nontrivial things *at the level of our models*, but the code is big. However the general idea is the same: original activity within a guard similar to $DnextSeqNo < \mathrm{BIGNUM}$, modified activity within a guard like $DnextSeqNo = \mathrm{BIGNUM}$.

Issues arising

- The (predominantly guard-driven) route was chosen to not disturb the validity of the remainder of the security proof, which we did not have time to redo. A better design would incorporate further purse states to express BIGNUM overflow, but would potentially entail much more work.
- In fact *DIncreasePurseOkay* is a metaphor for a number of operations, some of which need to increase *DnextSeqNo* others not, some of which need to output nontrivial messages others not, etc. In reality some of these are modified nontrivially when you hit BIGNUM.
- Other operations, eg. *AbortPurseOkay*, *StartFromPurseEafromOkay*, *StartToPurse-EafromOkay* do nontrivial things *at the level of our models*, but the code is big. However the general idea is the same: original activity within a guard similar to $DnextSeqNo < \text{BIGNUM}$, modified activity within a guard like $DnextSeqNo = \text{BIGNUM}$.
- Could have done it just with guards: ALL purse operations are $\vee$-ed with *Ignore* to totalise them. However this addresses full system requirements rather poorly. The actual implementation is strictly speaking NOT a refinement of the C model.

3. Log full.

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

3. Log full.

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

$\Delta CConPurse$

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

In reality, the log is not an (unbounded) set, but a sequence of length LOGMAX.

In reality, LOGMAX is *small* (about 5 in fact), so filling the log cannot realistically be postponed indefinitely far into the future. Compare:

- Sequence Number: Only total reached so far matters. Binary encoding OK.
  Few bits $\Rightarrow$ Large numbers.
- Exception Log: Each entry important in itself (and 'large'). Unary encoding needed.
  Few bits $\Rightarrow$ Small Log.

In reality, the log is not an (unbounded) set, but a sequence of length LOGMAX.

In reality, LOGMAX is *small* (about 5 in fact), so filling the log cannot realistically be postponed indefinitely far into the future. Compare:

- Sequence Number: Only total reached so far matters. Binary encoding OK.
  Few bits $\Rightarrow$ Large numbers.
- Exception Log: Each entry important in itself (and 'large'). Unary encoding needed.
  Few bits $\Rightarrow$ Small Log.

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. We introduce a new purse status $DexLogFull$.

DAbortPurseOkay

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

$$\Xi DConPurseAbort$$
$$DnextSeqNo' \geq DnextSeqNo$$
$$(\# DexLog < \text{LOGMAX} \Rightarrow DLogIfNecessary)$$
$$(\# DexLog \geq \text{LOGMAX} \Rightarrow DexLog' = DexLog)$$
$$(\# DexLog < \text{LOGMAX} - 1 \Rightarrow Dstatus' = DeaFrom \wedge Dm! = \perp)$$
$$(\# DexLog \geq \text{LOGMAX} - 1 \Rightarrow Dstatus' = DexLogFull \wedge Dm! = \text{``}\textit{Purse blocked. Go to bank.}\text{''})$$

where

DLogIfNecessary

$$\Delta DConPurse$$

$$DexLog' = DexLog \frown (\text{if } Dstatus \in \{Depv, Depa\} \text{ then } \langle DpdAuth \rangle \text{ else } \langle\rangle)$$

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

Retrieve relation for C-D model development step:

$CConPurseAbort$ **“=”** $DConPurseAbort \wedge$
$(Cstatus$ **“=”** $Dstatus \vee Dstatus = DexLogFull) \wedge$
$CnextSeqNo = DnextSeqNo \wedge$
$CexLog$ **“=”** $DexLog$

Within relation for *AbortPurseOkay* :

true

Output relation for *AbortPurseOkay* :

$Cm! = Dm!$

Concedes relation for $AbortPurseOkay$:

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

(N.B. “=” means C-variable $=$ D-variable in the obvious way, as before.)

Subsequent formal modelling proceeds much as in the Sequence Number scenario.

Validation

As noted, the structure of the retrenchment arguments in both the Sequence Number and Full Log cases is very similar.

However the validation in the two cases is very different.

Validation

As noted, the structure of the retrenchment arguments in both the Sequence Number and Full Log cases is very similar.

However the validation in the two cases is very different.

Sequence Number	Full Log
Few bits give large capacity.	Restricted space is tight constraint.

Validation

As noted, the structure of the retrenchment arguments in both the Sequence Number and Full Log cases is very similar.

However the validation in the two cases is very different.

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’.

Validation

As noted, the structure of the retrenchment arguments in both the Sequence Number and Full Log cases is very similar.

However the validation in the two cases is very different.

<u>Sequence Number</u>

Few bits give large capacity.

Large bound achievable means bound never reached.

<u>Full Log</u>

Restricted space is tight constraint.

Small capacity means reaching bound ‘inevitable’.

Design out possibility of further transactions when log full.

Validation

As noted, the structure of the retrenchment arguments in both the Sequence Number and Full Log cases is very similar.

However the validation in the two cases is very different.

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.

4. Hash function.

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

4. 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

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

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

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.

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. N.B. This goes beyond the chance receipt of spurious Clear messages in the C world (these are assumed not to occur, due to the assumed nonforgeability of the cryptographically protected protocol messages).

A job for retrenchment.

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

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

$$\mathrm{dom}\ CconAuthPurse = \mathrm{dom}\ DconAuthPurse \wedge$$
$$(\forall\ nm \in \mathrm{dom}\ CconAuthPurse \bullet CconAuthPurse\ nm \text{ “=” } DconAuthPurse\ nm) \wedge$$
$$Cether \text{ “=” } Dether$$

World Level Within relation for *ClearExceptionLog* :

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

World Level Output relation for *ClearExceptionLog* :

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

World Level Concedes relation for *ClearExceptionLog* :

$$CconAuthPurse'\ Cname?\ \text{“=”}\ DconAuthPurse'\ Dname? \wedge$$
$$\neg((CconAuthPurse\ Cname?).CexLog \subseteq \{Cname?\} \lhd Carchive \wedge$$
$$\quad (DconAuthPurse\ Dname?).DexLog \subseteq \{Dname?\} \lhd Darchive) \wedge$$
$$Cm! = Dm! \wedge$$
$$\underline{\neg CAllValueAccountedPurse'\ Cname?}$$

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* ...	$(2\times64+3\times32)\times5 \;\; \geq 1K$
• Hashed log entries *Dimage* ...	64?, 128?, 256?

Analysis

- To defend against malicious attack, only the hash offers any protection.

Analysis

- To defend against malicious attack, only the hash offers any protection.
  So need at least 256 bits of hash to give reasonable protection.

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

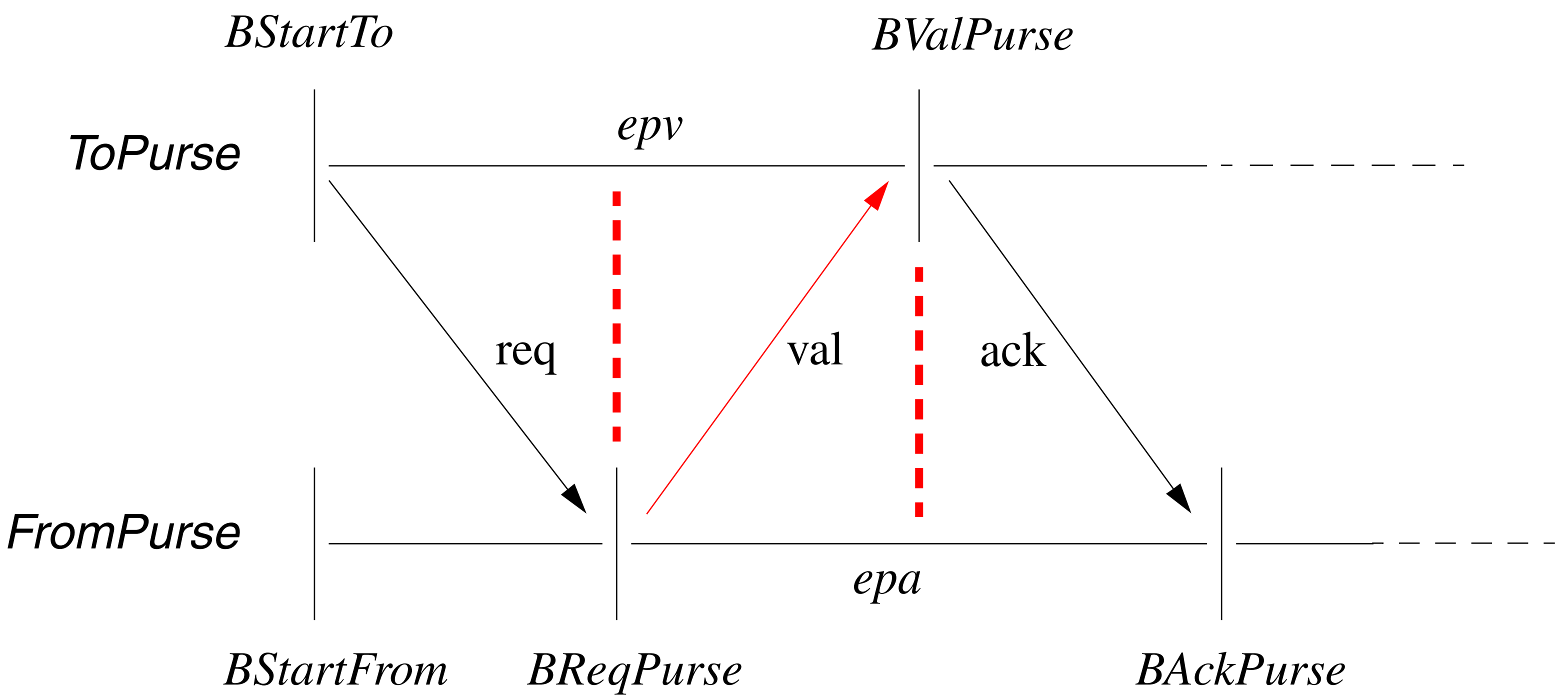
- To defend against malicious attack, only the hash offers any protection.
  So need at least 256 bits of hash to give reasonable protection.
  At 1 trial per ms., ... orders of magnitude longer than age of universe.
- For accidental erasure, the malicious threshold, 256 bits, is firmed up with (say):
  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.

5. Balance enquiry.

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

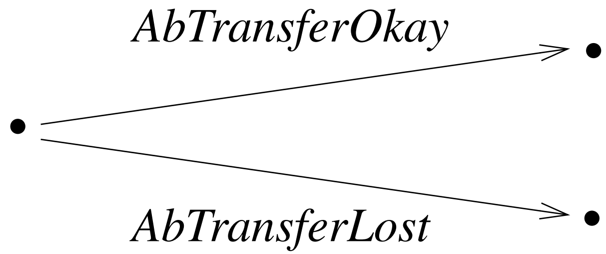
5. Balance enquiry.

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



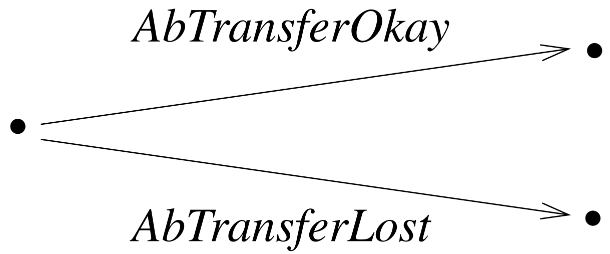
5. Balance enquiry.

Abstract level transaction:

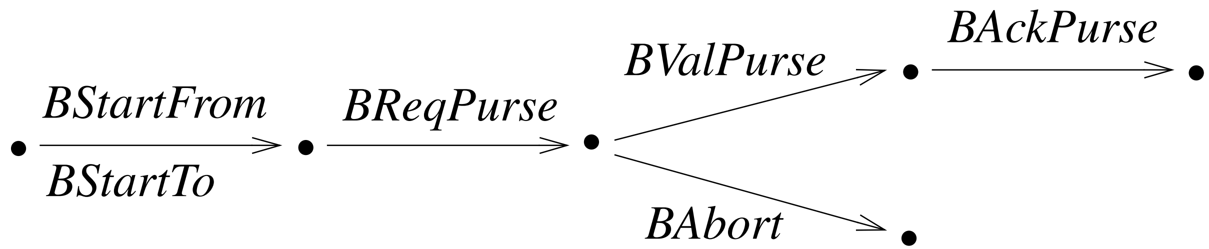


5. Balance enquiry.

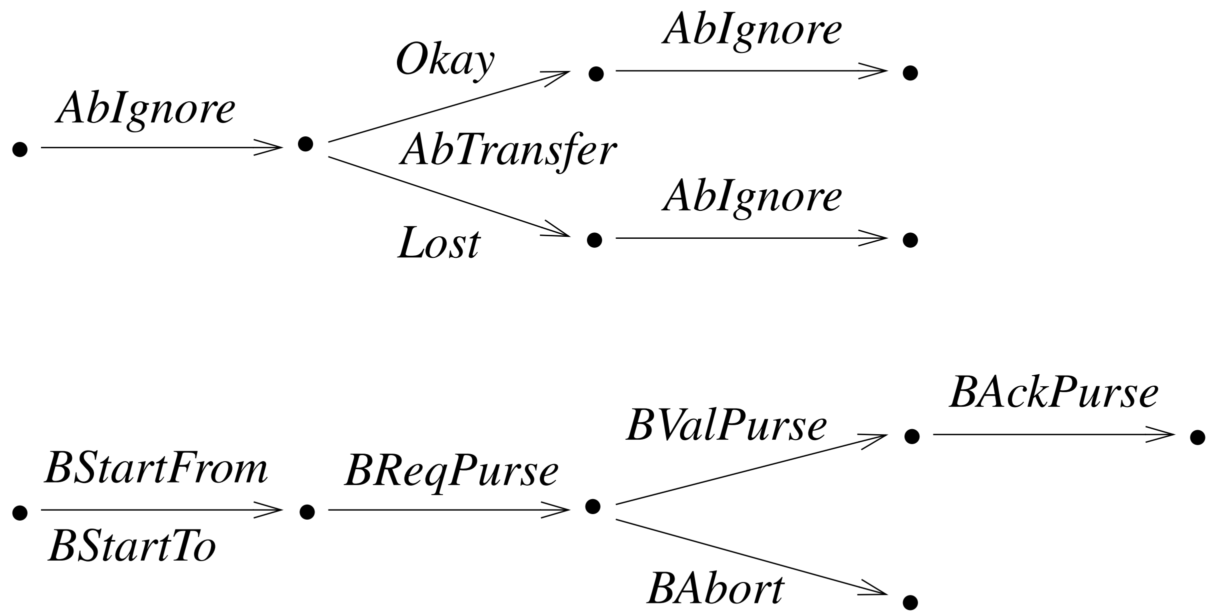
Abstract level transaction:



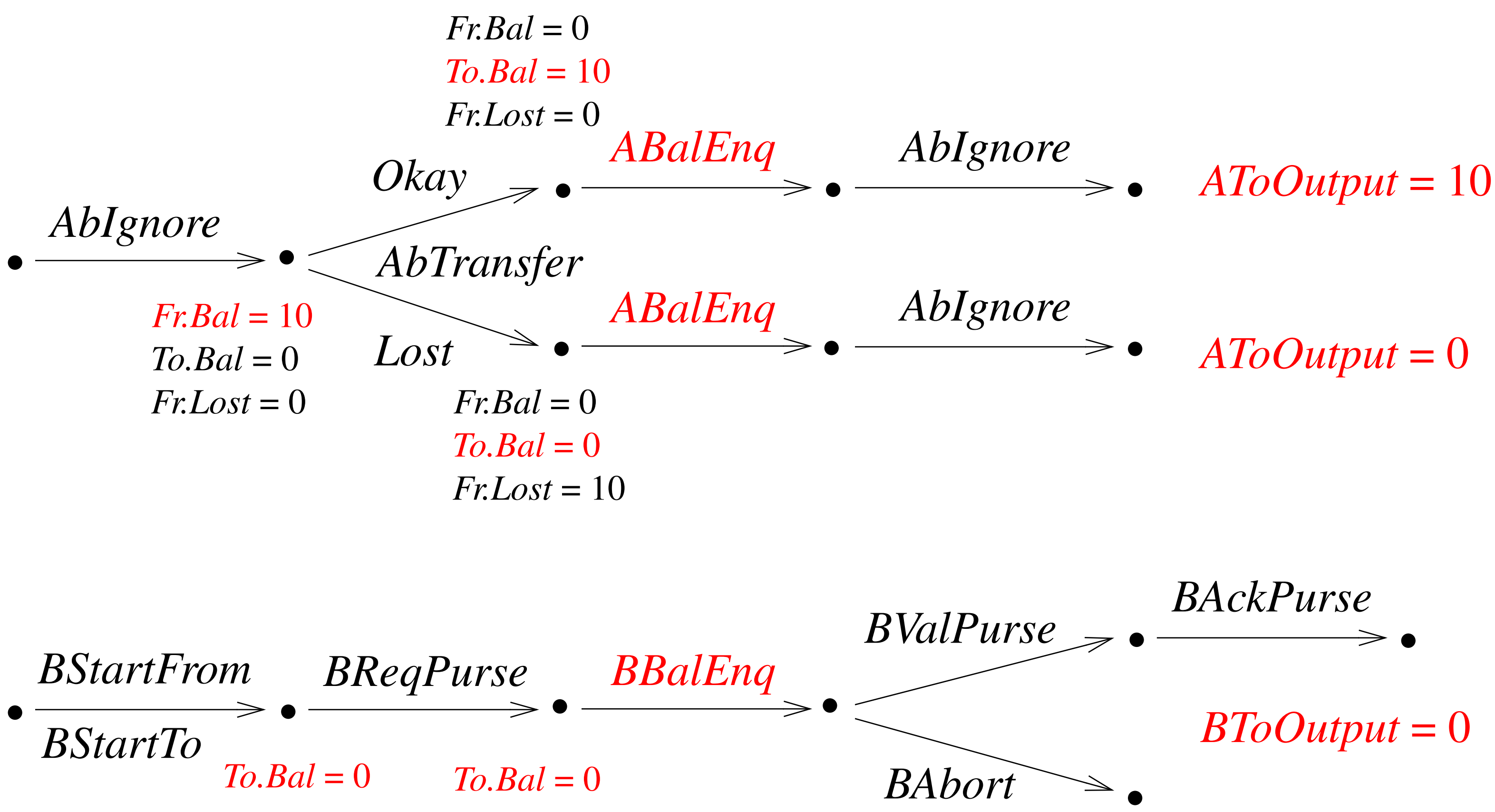
Between level transaction (main paths):



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



What if there's a *ToPurse* $BalanceEnquiry$ 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):

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

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):

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

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 ...
- 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):

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

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 ...
- 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:

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

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 $AToInput? = BToInput?$

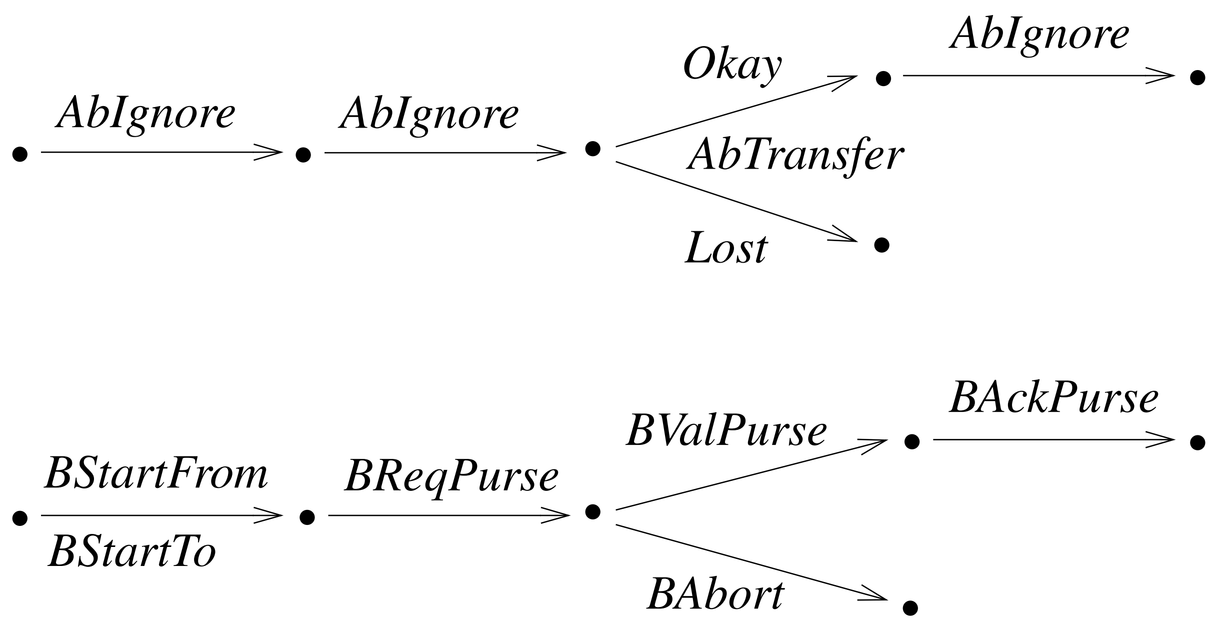
Story 2: Forward (1,1) refinement and retrenchment

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.

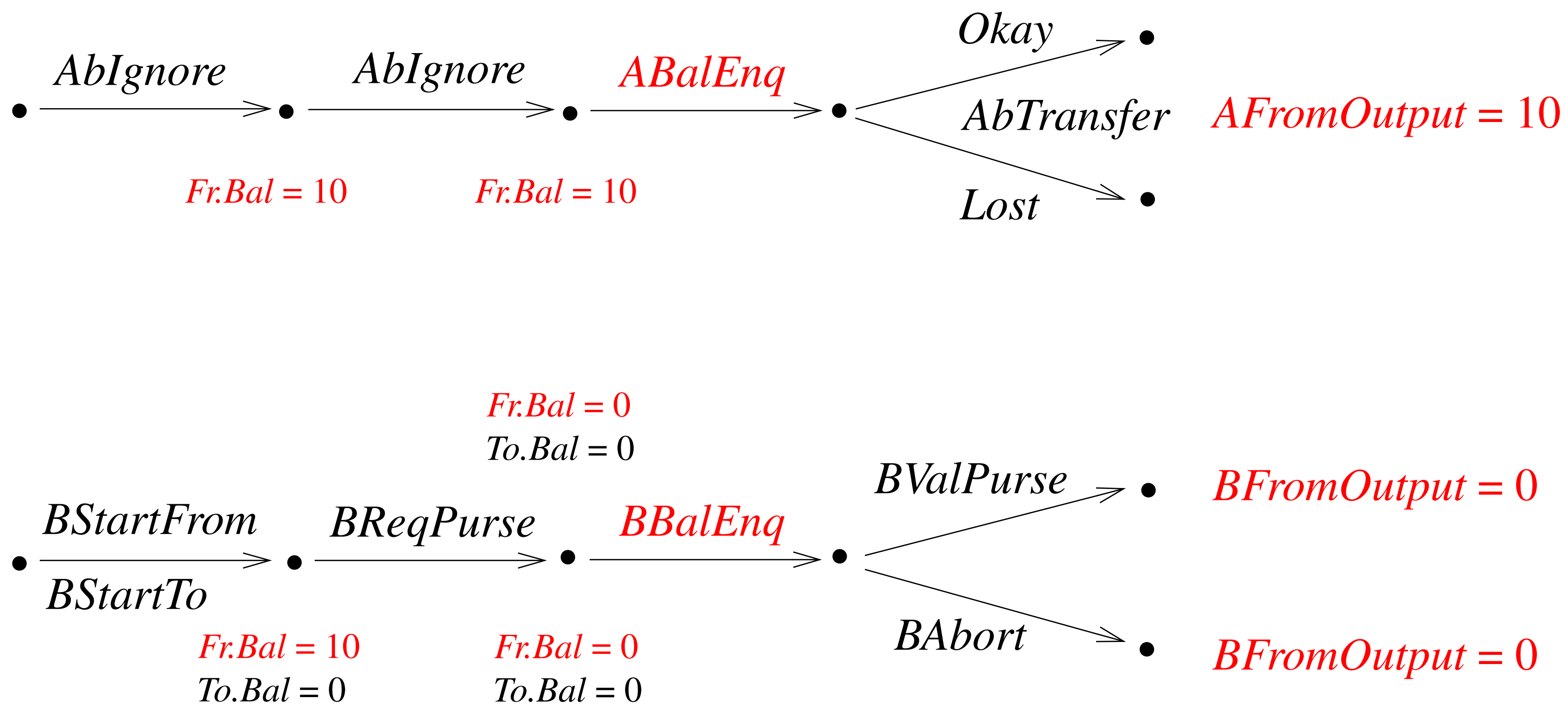
Story 2: Forward (1,1) refinement and retrenchment

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*):



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!!

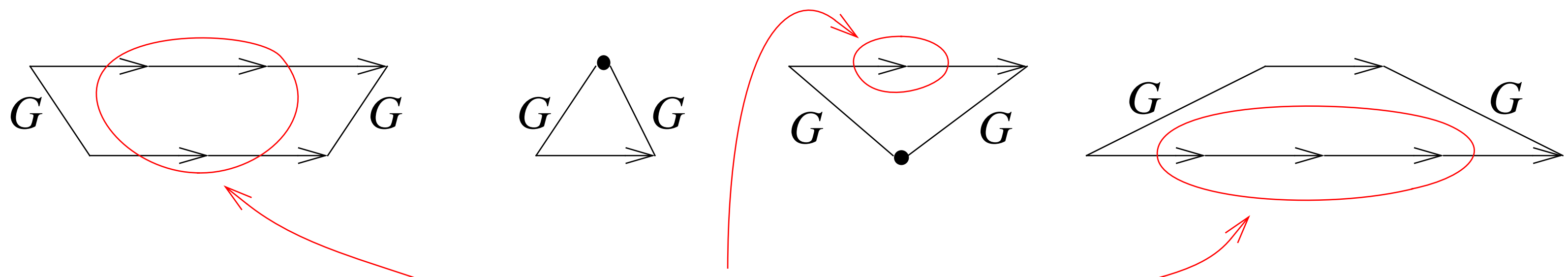
- This time the output relation $Out_{Op}(o,p)$ fails (provided it is identity).

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

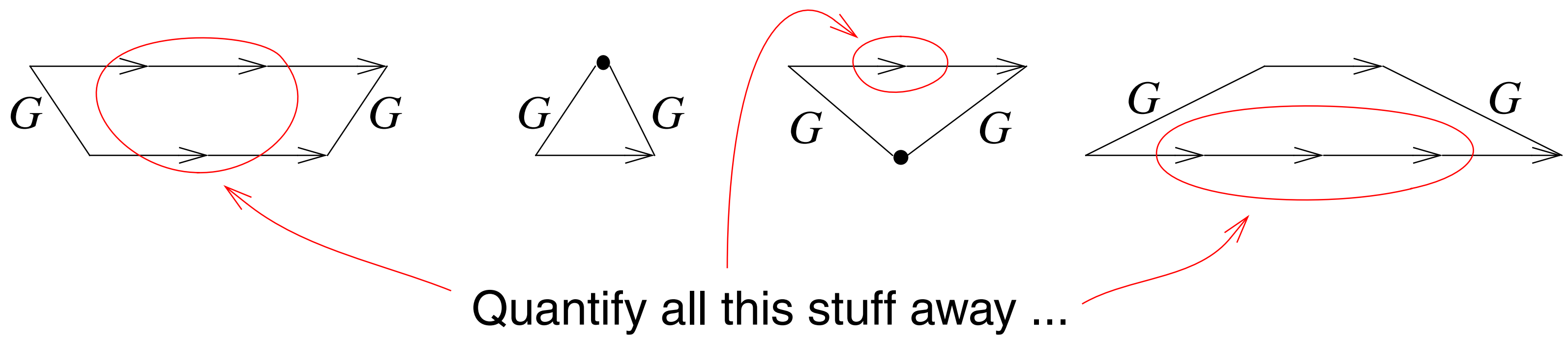
Generalised forward refinement gets away from the (1,1) refinement paradigm. The following are typical primitive refinement building blocks, which are stitched together to produce simulations:



Quantify all this stuff away ...

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.