Event Model Decomposition

(version 1.3 April 2009)

J.R. Abrial ETHZ

1 Introduction

Developing an event model by successive refinements usually starts with very few events (sometimes even a single event) and with a very few state variables. On the contrary, it usually ends up with a last refinement step dealing with many events and many variables. This is because one of the most important mechanism of the Event-B approach consists in introducing *new* events during refinement steps. The refinement mechanism is also used at the same time to significantly enlarge the number of state variables.

At some point, we might have so many events and so many state variables that the refinement process might become quite heavy. And we may also figure out that the refinement steps we are trying to undertake are not involving any more the totality of our system (as was the case at the beginning of the development), only a few variables and events are concerned, the others playing a passive role only.

The idea of *model decomposition* is thus clearly very attractive: it consists in cutting an heavy event system into *smaller pieces* which can be handled more comfortably than the whole. More precisely, each piece should be able to be refined *independently* of the others. But, of course, the constraint that must be satisfied by this decomposition is that such independently refined pieces could always (in principle) be easily re-composed. This re-composition process should then result in a system which could have been obtained directly without the decomposition, which thus appears to be just a kind of "divide-and-conquer" artefact.

But this is clearly not the only interesting methodological outcome of decomposition. It also allows us to build up the architecture of our future system by dividing it into independent components with *well defined relationship*. This last important point will be fully explained in what follows.

This paper contains the feasibility study of such a mechanism. After proposing an informal definition of decomposition in the next section, we outline the methodological outcome and constraints of this approach (sections 3). We then present its main difficulty (section 4) and propose a solution to it (section 5), which nevertheless presents a certain limitation (as explained in section 5.1). In section 6, a short example shows the mechanism at work. In section 7, we present another example explaining how the main limitation of this approach, as described in section 5.1, can be overcome. The proof that guarantees a well-defined mathematical approach to decomposition is developed in the Appendix.

2 Informal Definition

Decomposing an event model $\mathcal{M}$ is defined as follows:

1. $\mathcal{M}$ is split into several sub-models, say $\mathcal{N}, \ldots, \mathcal{P}$.

2. The events of $\mathcal{M}$ are partitioned and the elements of this partition form the events of the sub-models.

3. We shall see in section 4 how the variables of model $\mathcal{M}$ are also distributed among the sub-models.

4. These sub-models are then refined several times *independently* yielding eventually $\mathcal{NR}, \ldots, \mathcal{PR}$.

5. These refined models might be put together to form a re-composed model $\mathcal{MR}$.

6. The re-composed model $\mathcal{MR}$ must be *guaranteed* to be a refinement of $\mathcal{M}$.

This process is illustrated in the following diagram:

$$r$$

$$\begin{array}{ccccc} & \text{Decomposition} & \text{Refinement} & \text{Recomposition} & \\ \mathcal{M} & \rightarrow & \left\{ \begin{array}{ccc} \mathcal{N} & \rightarrow & \mathcal{NR} \\ & \cdots & \\ \mathcal{P} & \rightarrow & \mathcal{PR} \end{array} \right\} & \rightarrow & \mathcal{MR} \end{array}$$

It is important to notice here that point 5 above (the recomposition) will *never* be performed in practice. One has only to figure out that it *can be done* and that the refinement condition stated in point 6 ($\mathcal{MR}$ is a refinement of $\mathcal{M}$) must then be satisfied.

3 Methodological Outcome and Constraints of Decomposition

This decomposition process may play five important practical methodological roles:

1. It is certainly easier to refine sub-models $\mathcal{N}, \ldots, \mathcal{P}$ independently rather than together.

2. Refinements of sub-models $\mathcal{N}, \ldots, \mathcal{P}$ can be further decomposed in the same way, and so on.

3. Sub-models $\mathcal{N}, \ldots, \mathcal{P}$ could already possess some refinements able to be reused in several projects.

4. Decomposition is the basis for building the architecture of the final system we want to build.

5. A consequence of decomposition is that several independent users can work on the sub-models

Point 3 above is methodologically very important. It shows how decomposition and composition are tied together. In other words, we decompose in order to be able to compose several existing models taken off the shelf. By having a decomposition phase done before the composition one, we ensure that the composition leads to a correct global model, namely the one presented before the decomposition.

Our decomposition approach shall obey two main constraints:

1. We must define a process that is *totally robust* mathematically speaking.

2. We do not want to modify in any way the mathematical definition and concept of refinement.

4 The Main Difficulty: Variable Distribution

The difficulty with the variable distribution is better illustrated with a simple example.

Suppose that we have a certain model, $\mathcal{M}$, with four events `e1`, `e2`, `e3` et `e4`. We would like to decompose $\mathcal{M}$ into two separate models: (1) $\mathcal{N}$ dealing with events `e1` and `e2`, and (2) $\mathcal{P}$ dealing with events e3 and e4. We are interested in doing this decomposition because we know that there are some nice refinements that can be performed on e1 and e2 and, independently, on e3 and e4.

But in doing this *event partition* we must also perform a certain *variable distribution*. Suppose that we have three variables $v1$, $v2$ and $v3$ in $\mathcal{M}$. Like the events, the variables must be split too. For instance, we might put $v1$ and $v2$ with $\mathcal{N}$ (because e1 and e2 are supposedly working with them and not with $v3$). As a result, $v3$ goes, quite naturally, with $\mathcal{P}$. But the difficulty here is that e3 and e4, which work with $v3$, *might also work with* $v2$. So, besides $v3$, $\mathcal{P}$ certainly also requires $v2$ to deal correctly with e3 and e4. In summary: (1) e1 and e2 work with $v1$ and $v2$, and (2) e3 and e4 work with $v2$ and $v3$. So $\mathcal{N}$ must have variables $v1$ and $v2$ and $\mathcal{P}$ must have variables $v2$ and $v3$. Variable $v2$ is common to both $\mathcal{N}$ and $\mathcal{P}$.

The problem seems unsolvable since, apparently, there will always be some variables that are needed in several decomposed models. In other words, the splitting of the events will always conflict with that of the variables. Notice however that this should not be surprising: after all, variable $v2$ is simply the *communication channel* situated between sub-models $\mathcal{N}$ and $\mathcal{P}$.

So, the question of common variables, $v2$ in our example, is unavoidable. How are we going to solve this difficulty?

5 The Solution: Shared Variables and External Events

5.1 Shared Variables

We have no choice: the shared common variables must clearly be *replicated* in the various components of our decomposition. Notice that the shared variables in question can be modified by any of the components: we do not want to make any specialization of the components, some of them being only allowed to read, while some other to write these variables. We know that it is not possible in general.

The new difficulty that arises immediately at this point concerns the problem of refinement. In principle, each component can freely data-refine its state space. So that the *same* replicated variable could, in principle, be refined in one way in one component and differently in another: this is not acceptable since then the two components cannot communicate as they are not using the same conventions on the shared variable.

The price to pay in order to solve this difficulty is to give the replicated variables a *special status* in the components where they stay. A shared variable has a simple limitation: it must always be present in the state space of any refinement of the component. In other words, a shared variable *cannot be data-refined*. We shall see in section 7 how this limitation can be partially overcome.

Notice that there is no theoretical impossibilities to have the shared variables being refined. But then again the same shared variable has to be refined in the same way in each independent sub-model. This is not very convenient as we want the sub-models to be genuinely independent.

5.2 External Events

The notion of shared replicated variables we introduced in the previous section is not sufficient. Suppose that in a certain component a shared variable is only read, not written. The trouble with that shared variable is that it has suddenly become a constant in that component, which is certainly not what we want.

What we need thus in each component, is a number of additional events *simulating* the way our shared variables are handled in the non-decomposed model. Such events are called *external events*. Each of them mimics, using the shared variables only, an event of the non-decomposed model that modifies the shared variables in question. The reader has understood: mimic simply means "is an abstraction of". Of course such external events cannot be refined in their component. In section 5.4 we explain how the external events are practically constructed.

Notice that there is a distinction to be made between a shared variable and an external event. A shared variable is shared in all sub-models where it can be found, whereas an external event always has a non-external counterpart elsewhere. An event, however, can be external in several sub-models.

Notice finally that the external events are just modeling artefacts. In fact, the final code produced out of the last refinements of the sub-models does not contain any translation attached to the external events.

5.3 About the Invariants

In previous sections, we explained how the events and the variables are distributed among the sub-models. But we did not mentioned the invariants. Their destination is simple:

1. An Invariant dealing with the private variables of a sub-model is copied in that sub-model.

2. An invariant involving a shared variable only is copied in the sub-models where this variable is shared.

3. An invariant involving a shared variable together with other variables is not copied.

5.4 Final Recomposition

The re-composition of refinements of the various sub-models is now extremely simple. We put together all the variables of the individual sub-models (with the various shared variables incorporated only once) and we simply throw away all the external events of each sub-model.

It remains now for us to prove that the re-composed model is indeed a refinement of the initial one. Notice again that this re-composition will never be done: it is simply a thought experiment. In other words, it is just something that could be done, and which must then yield a refinement of the initial non-decomposed model. The proof that the re-composition is indeed a refinement of the original non-decomposed model is shown in the Appendix.

5.5 Practical Construction of External Events

An external event is the *projection* of the original event on the state of the sub-model. Practically, this can be done in a very systematic manner by replacing the disappearing variables by simple parameters of the external events. This can be illustrated on a simple example. Let e2 be an event of the original model. Suppose that the original model deals with variables $v1$, $v2$, and $v3$. Suppose that the event e2 does not work with variable $v3$. Here is thus the event e2:

```
e2
  when
    G(v1, v2)
  then
    v1 := E(v1, v2)
    v2 := F(v1, v2)
  end
```

The projection of e2 on a state made of variable $v3$ together with shared variable $v2$ is as follows:

```
external_e2
  any x1 where
    G(x1, v2)
  then
    v2 := F(x1, v2)
  end
```

As can be seen, the variable $v1$ occurring in e2 has been replaced by the parameter $x1$ and the assignment to $v1$ in e2 has disappeared. It might sometimes be necessary to add an additional guard in order to define the type of parameter $x1$. In the more general case where the action in non-deterministic as in:

```
e2
  when
    G(v1, v2)
  then
    v1, v2 :| P(v1, v2, v1', v2')
  end
```

then the projection of e2 on a state made of variable $v3$ together with shared variable $v2$ is as follows:

```
external_e2
  any  x1  where
    G(x1, v2)
  then
    v2 :| ∃y1 · P(x1, v2, y1, v2')
  end
```

5.6 Tool Requirements for Decomposition

The requirements for a tool able to help users decomposing a model are rather simple:

1. Given a model to be decomposed, the user must be able to designate the various elements of the decomposed sub-models: what are their private events, what are their external events, what are their private variables, what are the shared variables.

2. The tool must verify that all events and all variables are indeed well partitioned and distributed.

3. The tool must then generate the external events (as shown in the previous section) to be incorporated in each decomposed sub-model.

4. Finally, the tool must generate the various independent decomposed sub-model.

Notice that some extensions to the basic Rodin Platform tools have to be undertaken. We have to introduce the notion of shared variables in a model as well as that of external events. Let us recall that shared variables and external events must not be refined.

6 A Short Example

In this section,we propose a short example. It is a rather toyish and very artificial example however. Its role is simply to illustrate what we have described so far.

6.1 The Initial Model

In an initial model $\mathcal{M}$, we have three variables a, b, and c handling some natural number values. These variables are "controlled" by three boolean variables m, n, and p respectively. We also have four events named in_a, a_2_b, b_2_c, and out_c. These events respectively insert a new value in a (in_a), move values from a to b (a_2_b), and from b to c (b_2_c), and finally remove values from c (out_c). This is shown in the following diagram:

$$\xrightarrow{\text{in\_a}} \; a \; \xrightarrow{\text{a\_2\_b}} \; b \; \xrightarrow{\text{b\_2\_c}} \; c \; \xrightarrow{\text{out\_c}}$$

The moving of the values should respect some constraints handled by the boolean control variables m, n, and p. When the control variable m is equal to TRUE, this means that the value of a is fresh so that it can be sent to b. This, however, can only be performed provided the value of b is old, meaning that it has already been sent to c: this fact is recorded by the value of n, the boolean variable controlling b, when it is equal to FALSE. Boolean variable p play a similar role with c. Next are the typing of the variables and the definition of the various events:

variables: a
m
b
n
c
p

inv0_1: $a \in \mathbb{N}$
inv0_2: $m \in \text{BOOL}$
inv0_3: $b \in \mathbb{N}$
inv0_4: $n \in \text{BOOL}$
inv0_5: $c \in \mathbb{N}$
inv0_6: $p \in \text{BOOL}$

INIT
$a := 0$
$m := \text{FALSE}$
$b := 0$
$n := \text{FALSE}$
$c := 0$
$p := \text{FALSE}$

in_a
when
$m = \text{FALSE}$
then
$a :\in \mathbb{N}$
$m := \text{TRUE}$
end

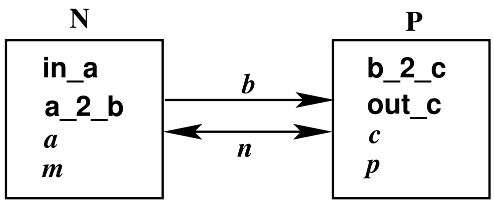
a_2_b
when
$m = \text{TRUE}$
$n = \text{FALSE}$
then
$b := a$
$m := \text{FALSE}$
$n := \text{TRUE}$
end

b_2_c
when
$n = \text{TRUE}$
$p = \text{FALSE}$
then
$c := b$
$n := \text{FALSE}$
$p := \text{TRUE}$
end

out_c
when
$p = \text{TRUE}$
then
$p := \text{FALSE}$
end

6.2 Preparing the Decomposition

Our purpose is now to decompose the model $\mathcal{M}$ presented in previous section into two separate sub-models $\mathcal{N}$ and $\mathcal{P}$ as shown in the following figure. Sub-model $\mathcal{N}$ contains events in_a and a_2_b together with variables a and m, whereas sub-model $\mathcal{P}$ contains events b_2_c and out_c together with variables c and p:



Variables b and n are the shared variables forming the channels between the two sub-models. We can see that sub-model $\mathcal{N}$ modifies the shared variable b (write) while sub-model $\mathcal{P}$ only access it (read). However, both sub-models modify and access the shared variable n. In other words, the "channel" n is full-duplex.

It might be interesting to simplify this and replace the full-duplex channel n by two simple channels. So, before doing our decomposition we might first refine our model $\mathcal{M}$ to model $\mathcal{M}1$. We replace the variable n by two bit variables r and s. The variable r is modified by the event a_2_b, whereas the variable s is accessed only by this event. We have a symmetric situation with event b_2_c. Here are the variables and events of refinement $\mathcal{M}1$:

variables: a
m
b
c
p
r
s

inv1_1: $r \in \{0, 1\}$
inv1_2: $s \in \{0, 1\}$
inv1_3: $r = s \;\Leftrightarrow\; n = \text{FALSE}$

INIT
$a := 0$
$m := \text{FALSE}$
$b := 0$
$c := 0$
$p := \text{FALSE}$
$r := 0$
$s := 0$

Invariant **inv1_3** shows how concrete variables r and s are glued to the abstract variable n. Notice that this invariant could have been written equivalently as $n = \text{bool}(r \neq s)$.

in_a
when
$m = \text{FALSE}$
then
$a :\in \mathbb{N}$
$m := \text{TRUE}$
end

a_2_b
when
$m = \text{TRUE}$
$r = s$
then
$b := a$
$m := \text{FALSE}$
$r := 1 - r$
end

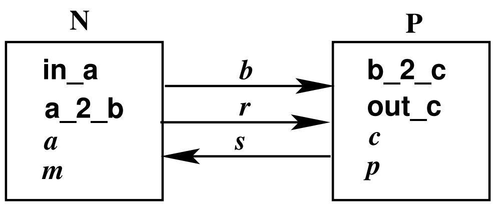
b_2_c
when
$r \neq s$
$p = \text{FALSE}$
then
$c := b$
$s := 1 - s$
$p := \text{TRUE}$
end

out_c
when
$p = \text{TRUE}$
then
$p := \text{FALSE}$
end

What is shown here with variables r and s is the classical alternating bit protocol.

6.3 The Decomposition

We now propose the following decomposition with simple channels only:



The model $\mathcal{N}$ has proper events in_a and a_2_b, which are thus exact copies of events bearing the same names in model $\mathcal{M}1$. Moreover, the external event external_b_2_c, is then a projection of the event bearing the name b_2_c in the initial model $\mathcal{M}1$. Model $\mathcal{N}$ has five variables a, m, b, r, and s. The last three are shared as indicated below.

variables: a
m
shared: b
r
s

inv_N0_1: $a \in \mathbb{N}$
inv_N0_2: $m \in \text{BOOL}$
inv_N0_3: $b \in \mathbb{N}$
inv_N0_4: $r \in \{0, 1\}$
inv_N0_5: $s \in \{0, 1\}$

INIT
$a := 0$
$m := \text{FALSE}$
$b := 0$
$r := 0$
$s := 0$

Notice that in the INIT event, the initialization of the variable s is rather an external initialization.

in_a
when
$m = \text{FALSE}$
then
$a :\in \mathbb{N}$
$m := \text{TRUE}$
end

a_2_b
when
$m = \text{TRUE}$
$r = s$
then
$b := a$
$m := \text{FALSE}$
$r := 1 - r$
end

external_b_2_c
when
$r \neq s$
then
$s := 1 - s$
end

Notice that the event external_b_2_c should be the following according to what has been said in section 5.4, but this is clearly equivalent to what we wrote above because the parameter xp has no influence on the action and the guard $\exists xp \cdot xp = \text{FALSE}$ holds trivially:

external_b_2_c
any xp **where**
$xp = \text{FALSE}$
$r \neq s$
then
$s := 1 - s$
end

Next is our second decomposed model $\mathcal{P}$. It is organized symmetrically to $\mathcal{N}$. Likewise, the external event external_a_2_b is a projection of the event a_2_b in $\mathcal{M}1$.

variables: c, p
shared: b, r, s

inv_P0_1: $c \in \mathbb{N}$
inv_P0_2: $p \in \text{BOOL}$
inv_P0_3: $b \in \mathbb{N}$
inv_P0_4: $r \in \{0,1\}$
inv_P0_4: $s \in \{0,1\}$

INIT
$c := 0$
$p := \text{FALSE}$
$b := 0$
$r := 0$
$s := 0$

Similarly to what we observed above for model $\mathcal{N}$, here in the INIT event the initialization of the variable r is rather an external initialization.

external_a_2_b
any x **where**
$x \in \mathbb{N}$
$r = s$
then
$b := x$
$r := 1 - r$
end

b_2_c
when
$r \neq s$
$p = \text{FALSE}$
then
$c := b$
$s := 1 - s$
$p := \text{TRUE}$
end

out_c
when
$p = \text{TRUE}$
then
$p := \text{FALSE}$
end

6.4 Refinements

We now refine $\mathcal{N}$ to $\mathcal{NR}$. The refinement consists of adding a new variable d. The value of the variable a is gradually moved to d by incrementing d. For this, we introduce a new convergent event named inc_d. Next are the events of $\mathcal{NR}$. Notice that the event external_b_2_c is not modified).

in_a **when** $m = \text{FALSE}$ **then** $a :\in \mathbb{N}$ $m := \text{TRUE}$ **end**	inc_d **status** convergent **when** $m = \text{TRUE}$ $d \neq a$ **then** $d := d + 1$ **end**	a_2_b **when** $m = \text{TRUE}$ $r = s$ $d = a$ **then** $b := d$ $m := \text{FALSE}$ $r := 1 - r$ $d := 0$ **end**	external_b_2_c **when** $r \neq s$ **then** $s := 1 - s$ **end**

In order to prove the convergence of the event inc_d, we have to exhibit a variant, which is clearly the following:

variant_NR: $a - d$

Now proving that this variant is a natural number requires that d is always less than or equal to a. In turn, proving this invariant requires an additional invariant as shown below:

inv_NR_1: $d \leq a$

inv_NR_2: $m = \text{FALSE} \Rightarrow d = 0$

We now refine similarly $\mathcal{P}$ to $\mathcal{PR}$. The refinement consists of adding a new variable e. The value of the variable b is gradually moved to e by incrementing e. For this, we introduce a new convergent event inc_e. Next are the events of $\mathcal{PR}$. Notice that the event external_a_2_b is not modified.

external_a_2_b **any** x **where** $x \in \mathbb{N}$ $r = s$ **then** $b := x$ $r := 1 - r$ **end**	inc_e **status** convergent **when** $n = \text{TRUE}$ $e \neq b$ **then** $e := e + 1$ **end**	b_2_c **when** $r \neq s$ $p = \text{FALSE}$ $e = b$ **then** $c := e$ $s := 1 - s$ $p := \text{TRUE}$ $e := 0$ **end**	out_c **when** $p = \text{TRUE}$ **then** $p := \text{FALSE}$ **end**

Similar considerations as done for the previous refinement lead to the following variant and invariants:

variant_PR: $b - e$

inv_PR_1: $e \leq b$

inv_PR_2: $n = \text{FALSE} \Rightarrow e = 0$

6.5 Summary of the Development

The development of this short example can be summarized in the following diagram. We first did a refinement of our initial model $\mathcal{M}$ to $\mathcal{M}1$ (thus preparing the interface), then we decomposed $\mathcal{M}1$ into sub-models $\mathcal{N}$ and $\mathcal{P}$, and finally we refined our two sub-models to $\mathcal{NR}$ and $\mathcal{PR}$ respectively:

$$
\begin{array}{ccccc}
 & & \mathcal{M} & & \\
 & & \downarrow & & \\
 & & \mathcal{M}1 & & \\
 & \swarrow & & \searrow & \\
\mathcal{N} & & & & \mathcal{P} \\
\downarrow & & & & \downarrow \\
\mathcal{NR} & & & & \mathcal{PR}
\end{array}
$$

7 Another Example

The example presented in this section is a little less simple than the one presented in previous section. However, it is still clearly a bit artificial: we want to show how we can proceed in order to be able to refine the "natural" channel between two components. Contrarily to the previous example, we perform several refinements on the non-decomposed model before eventually decomposing it. The goal of these refinements is to construct carefully the interface between the future decomposed sub-models.

7.1 Initial Model

In this first model, we define a context dealing with two abstract sets: $QUESTION$ and $RESPONSE$. We also have a constant, $answer$, linking $QUESTION$ to $RESPONSE$:

sets: $QUESTION$
$RESPONSE$

constants: $answer$

axm_0_1: $answer \in QUESTION \to RESPONSE$

The model itself is made of two variables: $question$ and $response\_0$. These variables denote the set of questions and responses so far encountered. An obvious invariant, **inv0_2**, relates both variables. We have a single event question_response which does the job in one shot.

variables: $question$
$response\_0$

inv0_1: $question \subseteq QUESTION$

inv0_2: $response\_0 = answer[question]$

question_response
any q **where**
$q \notin question$
then
$question := question \cup \{q\}$
$response\_0 := response\_0 \cup \{answer(q)\}$
end

7.2 First Refinement

In this refinement, we introduce a channel, $channel\_0$, between the sets $question$ and $response$. The single event of previous abstract model is split into two events named prepare_question and produce_response. The former refines the abstract event question_response, while the latter is a new convergent event (with variant the finite set $channel\_0$). Notice that abstract variable $response\_0$ is data-refined to variable $response$ with an obvious gluing invariant as shown in **inv1_2**.

variables: $question$
$response$
$channel\_0$

inv1_1: $channel\_0 \subseteq question$

inv1_2: $response\_0 = response \cup answer[channel\_0]$

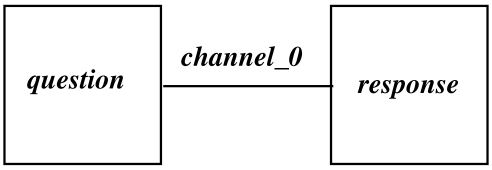
inv1_3: $\text{finite}(channel\_0)$

prepare_question
refines
question_response
any q **where**
$q \notin question$
then
$question := question \cup \{q\}$
$channel\_0 := channel\_0 \cup \{q\}$
end

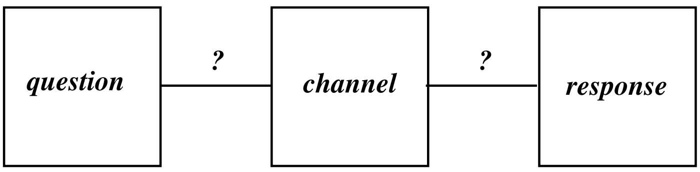
produce_response
status
convergent
any q **where**
$q \in channel\_0$
then
$channel\_0 := channel\_0 \setminus \{q\}$
$response := response \cup \{answer(q)\}$
end

variant1: $channel\_0$

At this point, it might be tempting to decompose this refinement into two sub-models with the shared variable $channel\_0$ in between them:



We shall not do that however because our intention is to later consider that $channel\_0$ is a simple abstraction for a far more complicated *middleware component*. In other words, we want to be able to later refine $channel\_0$. The idea is to encapsulate $channel\_0$ in its own component so that the decomposition will be as shown in the following figure:



The purpose of the next refinements is to prepare the yet unknown interfaces between these components.

7.3 Second Refinement

In this refinement, we introduce a buffer, $buffer\_2$ on the right hand side of the channel $channel\_0$, which is data-refined to $channel\_1$ (with gluing invariants **inv2_1** and **inv2_2**). The buffer is controlled by the boolean variable $bool\_2$.

variables: $question$
$response$
$channel\_1$
$buffer\_2$
$bool\_2$

inv2_1: $bool\_2 = \text{TRUE} \Rightarrow channel\_0 = channel\_1$

inv2_2: $bool\_2 = \text{FALSE} \Rightarrow channel\_0 = channel\_1 \cup \{buffer\_2\}$

inv2_3: $bool\_2 = \text{FALSE} \Rightarrow buffer\_2 \notin channel\_1$

```
prepare_question
  any q where
    q ∉ question
  then
    question := question ∪ {q}
    channel_1 := channel_1 ∪ {q}
  end
```

```
produce_response
  when
    bool_2 = TRUE
  with
    q = buffer_2
  then
    bool_2 := FALSE
    response := response ∪ {answer(buffer_2)}
  end
```

A new convergent event read_question is introduced (with variant the finite set $\{bool\_2, \text{TRUE}\}$).

```
read_question
  status
    convergent
  any q where
    q ∈ channel_1
    bool_2 = FALSE
  then
    bool_2 := TRUE
    channel_1 := channel_1 \ {q}
    buffer_2 := q
  end
```

variant2: $\{bool\_2, \text{TRUE}\}$

7.4 Third Refinement

Similarly to what has been done in the previous refinement, we introduce now a buffer on the left hand side of the channel.

variables: $question$
$response$
$channel$
$buffer\_1$
$bool\_1$
$buffer\_2$
$bool\_2$

inv3_1: $bool\_1 = \mathrm{TRUE} \Rightarrow channel\_1 = channel$

inv3_2: $bool\_1 = \mathrm{FALSE} \Rightarrow channel\_1 = channel \cup \{buffer\_1\}$

inv3_3: $bool\_1 = \mathrm{FALSE} \Rightarrow buffer\_1 \notin channel$

prepare_question
any q **where**
$q \notin question$
$bool\_1 = \mathrm{FALSE}$
then
$question := question \cup \{q\}$
$bool\_1 := \mathrm{TRUE}$
$buffer\_1 := q$
end

write_question
status
convergent
when
$bool\_1 = \mathrm{TRUE}$
then
$bool\_1 := \mathrm{FALSE}$
$channel := channel \cup \{buffer\_1\}$
end

read_question
any q **where**
$q \in channel$
$bool\_2 = \mathrm{FALSE}$
then
$bool\_2 := \mathrm{TRUE}$
$channel := channel \setminus \{q\}$
$buffer\_2 := q$
end

produce_response
when
$bool\_2 = \mathrm{TRUE}$
then
$bool\_2 := \mathrm{FALSE}$
$response := response \cup \{answer(buffer\_2)\}$
end

variant3: $\{bool\_1, \mathrm{FALSE}\}$

7.5 Fourth Refinement

In this refinement, we introduce alternating bits as we did in the previous example.

variables: $question$, $response$, $channel$, $buffer\_1$, $buffer\_2$, bit_11, bit_12, bit_21, bit_22

inv4_1: $bit\_11 \in \{0,1\}$

inv4_2: $bit\_12 \in \{0,1\}$

inv4_3: $bit\_11 = bit\_12 \Leftrightarrow bool\_1 = \text{FALSE}$

inv4_4: $bit\_21 \in \{0,1\}$

inv4_5: $bit\_22 \in \{0,1\}$

inv4_6: $bit\_21 = bit\_22 \Leftrightarrow bool\_2 = \text{FALSE}$

prepare_question
any q **where**
$q \notin question$
$bit\_11 = bit\_12$
then
$question := question \cup \{q\}$
$bit\_11 := 1 - bit\_11$
$buffer\_1 := q$
end

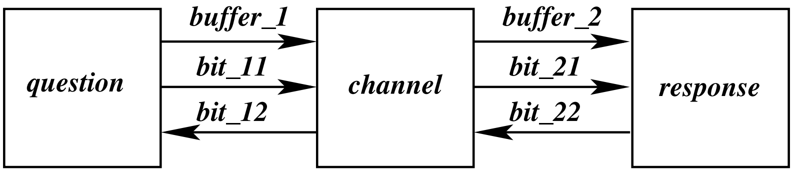
write_question
when
$bit\_11 \neq bit\_12$
then
$bit\_12 := 1 - bit\_12$
$channel := channel \cup \{buffer\_1\}$
end

read_question
any q **where**
$q \in channel$
$bit\_21 = bit\_22$
then
$bit\_21 := 1 - bit\_21$
$channel := channel \setminus \{q\}$
$buffer\_2 := q$
end

produce_response
when
$bit\_21 \neq bit\_22$
then
$bit\_22 := 1 - bit\_22$
$response := response \cup \{answer(buffer\_2)\}$
end

7.6 Decomposition

At this point, we are now ready to perform our decomposition according to the following schemata:



Left Component.

variables: $question$
shared: $buffer\_1$
bit_11
bit_12

invL_1: $question \subseteq QUESTION$

invL_2: $buffer\_1 \in QUESTION$

invL_3: $bit\_11 \in \{0, 1\}$

invL_4: $bit\_12 \in \{0, 1\}$

prepare_question
any q **where**
$q \notin question$
$bit\_11 = bit\_12$
then
$question := question \cup \{q\}$
$bit\_11 := 1 - bit\_11$
$buffer\_1 := q$
end

external_write_question
when
$bit\_11 \neq bit\_12$
then
$bit\_12 := 1 - bit\_12$
end

Right Component.

variables: $response$
shared: $buffer\_2$
bit_21
bit_22

invR_1: $response \subseteq RESPONSE$

invR_2: $buffer\_2 \in QUESTION$

invR_3: $bit\_21 \in \{0, 1\}$

invR_4: $bit\_22 \in \{0, 1\}$

external_read_question
any q **where**
$q \in QUESTION$
$bit\_21 = bit\_22$
then
$bit\_21 := 1 - bit\_21$
$buffer\_2 := q$
end

produce_response
when
$bit\_21 \neq bit\_22$
then
$bit\_22 := 1 - bit\_22$
$response := response \cup \{answer(buffer\_2)\}$
end

Middle Component.

variables: $channel$
shared: $buffer\_1$
bit_11
bit_12
$buffer\_2$
bit_21
bit_22

invM_1: $channel \subseteq QUESTION$

invM_1: $buffer\_1 \in QUESTION$

invM_2: $bit\_11 \in \{0, 1\}$

invM_3: $bit\_12 \in \{0, 1\}$

invM_4: $buffer\_2 \in QUESTION$

invM_5: $bit\_21 \in \{0, 1\}$

invM_6: $bit\_22 \in \{0, 1\}$

write_question
when
$bit\_11 \neq bit\_12$
then
$bit\_12 := 1 - bit\_12$
$channel := channel \cup \{buffer\_1\}$
end

read_question
any q **where**
$q \in channel$
$bit\_21 = bit\_22$
then
$bit\_21 := 1 - bit\_21$
$channel := channel \setminus \{q\}$
$buffer\_2 := q$
end

external_prepare_question
any q **where**
$q \in QUESTION$
$bit\_11 = bit\_12$
then
$bit\_11 := 1 - bit\_11$
$buffer\_1 := q$
end

external_produce_response
when
$bit\_21 \neq bit\_22$
then
$bit\_22 := 1 - bit\_22$
end

Appendix

As announced in section 5.4, the proof that the re-composition is indeed a refinement of the original non-decomposed model is now proposed here. We first present the mathematical framework, then we state what we have to prove, and finally we prove it. Note that the proofs presented in this Appendix have been performed automatically on the Rodin Platform.

We suppose that the initial non-decomposed model $\mathcal{M}$ has a state constructed on a set S. We are concerned with an event e mathematically represented by a binary relation from S to S:

$$e \in S \leftrightarrow S$$

For simplifying things without loss of generality, we envisage the decomposition of model $\mathcal{M}$ into two sub-models $\mathcal{N}$ and $\mathcal{P}$ only. These two sub-models have states constructed on two sets T and U respectively.

The decomposition is materialized by means of two projections functions p and q from S to T and S to U respectively. These functions are total surjections:

$$p \in S \twoheadrightarrow T$$

$$q \in S \twoheadrightarrow U$$

The reason for these functions to be total surjections is that in a more concrete setting p and q are compositions of proper Cartesian product projections prj_1 and prj_2, which are both total surjections.

The event e of model $\mathcal{M}$ is projected on the two sub-models. This results in two events mathematically represented by two binary relations a and b:

$$a \in T \leftrightarrow T$$

$$b \in U \leftrightarrow U$$

These relations are the projections of relation e. They are formally defined as follows:

$$a \ \widehat{=}\ p^{-1} \,;\, e \,;\, p$$

$$b \ \widehat{=}\ q^{-1} \,;\, e \,;\, q$$

The sub-model $\mathcal{N}$ is refined to a model $\mathcal{NR}$ with a state constructed on a set X. The refinement is done by means of a total binary relation l from X to T. Likewise, the sub-model $\mathcal{P}$ is refined to a model $\mathcal{PR}$ with a state constructed on a set Y. The refinement is done by means of a total binary relation m from Y to U:

$$l \in X \leftarrowtail\!\!\!\rightarrow T$$

$$m \in Y \leftarrowtail\!\!\!\rightarrow U$$

These relations are assumed to be total: it is a fundamental assumption used in the refinement theory as presented in chapter 14 of the book on Event-B.

Events a and b are refined to events ar and br. The forward refinement of these events is mathematically represented by the following predicates:

$$l^{-1} \,;\, ar \ \subseteq\ a \,;\, l^{-1}$$

$$m^{-1} \,;\, br \ \subseteq\ b \,;\, l^{-1}$$

Again, these refinement conditions come from chapter 14 of the book on Event-B (forward refinement).

The recomposition of refinements $\mathcal{NR}$ and $\mathcal{PR}$ results in a model $\mathcal{MR}$ whose state is constructed on a set Z. This set is projected on the two sets X and Y by means of two total surjective projections functions u and v:

$$u \in Z \twoheadrightarrow X$$

$$v \in Z \twoheadrightarrow Y$$

We now re-compose an event er on model $\mathcal{MR}$

$$er \in Z \leftrightarrow Z$$

by means of the two events ar and br. The event er is formally defined as follows:

$$er \ \widehat{=}\ (u \,;\, ar \,;\, u^{-1}) \cap (v \,;\, br \,;\, v^{-1})$$

Here we construct the re-composed event er out of the existing refinement events ar and br. This is the reason why we used an intersection.

Putting together the two refinement relations l and m, we get a refinement relation n:

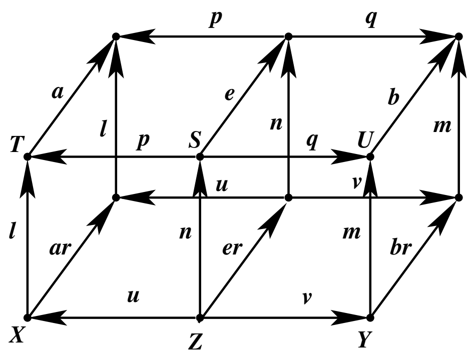
$$n \in Z \leftrightarrow S$$

which is defined as follows:

$$n \;\widehat{=}\; (u\,;l\,;p^{-1}) \cap (v\,;m\,;q^{-1})$$

Relation n is indeed a total relation as both relations $u\,;l\,;p^{-1}$ and $v\,;m\,;q^{-1}$ are total. For example, $u\,;l\,;p^{-1}$ is total because u is a total function, l is total relation, and p is a surjective function (hence p^{-1} is total relation).

All this can be illustrated on the following diagram:



Now, we have to prove the following theorem stating that er is a refinement of e, that is:

$$\boxed{n^{-1}\,;er \;\subseteq\; e\,;n^{-1}}$$

The proof is made of two parts. First, we prove:

$$n^{-1}\,;er \;\subseteq\; r$$

and then we prove:

$$r \;\subseteq\; e\,;n^{-1}$$

where r is a relation from S to Z (as are both relations $n^{-1}\,;er$ and $e\,;n^{-1}$):

$$r \in S \rightarrow Z$$

defined as follows:

$$r = (p\,;p^{-1}\,;e\,;p\,;l^{-1}\,;u^{-1}) \;\cap\; (q\,;q^{-1}\,;e\,;q\,;m^{-1}\,;v^{-1})$$

Proof of the First Part. The proof of the first part can be conducted in a completely algebraic form.

PROOF

$$
\begin{array}{ll}
n^{-1} ; er & \\
= & \text{definition of } er \\
n^{-1} ; ((u ; ar ; u^{-1}) \ \cap \ (v ; br ; v^{-1})) & \\
\subseteq & \text{set theory} \\
(\underline{n^{-1} ; u} ; ar ; u^{-1}) \ \cap \ (\underline{n^{-1} ; v} ; br ; v^{-1}) & \\
\subseteq & \text{lemmas} \\
(\underline{p ; l^{-1}} ; ar ; u^{-1}) \ \cap \ (\underline{q ; m^{-1}} ; br ; v^{-1}) & \\
\subseteq & \text{refinements of } a \text{ and } b \\
(p ; a ; l^{-1} ; u^{-1}) \ \cap \ (q ; b ; m^{-1} ; v^{-1}) & \\
= & \text{definitions of } a \text{ and } b \\
(p ; p^{-1} ; e ; p ; l^{-1} ; u^{-1}) \ \cap \ (q ; q^{-1} ; e ; q ; m^{-1} ; v^{-1}) &
\end{array}
$$

The previous proof relies on two lemmas. Here is the proof of the first one (the second one is similar).

PROOF of lemma: $n^{-1} ; u \ \subseteq \ p ; l^{-1}$

$$
\begin{array}{ll}
n^{-1} ; u & \\
= & \text{definition of } n \\
((p ; l^{-1} ; u^{-1}) \cap (q ; m^{-1} ; v^{-1})) ; u & \\
\subseteq & \text{set theory} \\
(p ; l^{-1} ; u^{-1} ; u) \cap (q ; m^{-1} ; v^{-1} ; u) & \\
= & u \text{ is a total surjection} \\
(p ; l^{-1}) \cap (q ; m^{-1} ; v^{-1} ; u) & \\
\subseteq & \text{set theory} \\
p ; l^{-1} &
\end{array}
$$

Proof of the Second Part. We have to prove now the second part, namely:

$$(p ; p^{-1} ; e ; p ; l^{-1} ; u^{-1}) \ \cap \ (q ; q^{-1} ; e ; q ; m^{-1} ; v^{-1}) \ \subseteq \ e ; n^{-1}$$

This proof cannot be conducted in an algebraic form like we did for the first part in the previous section. We have first to expand it as follows:

$$
\begin{array}{l}
\forall s, z' \cdot (\exists s1, s1' \cdot p(s) = p(s1) \ \wedge \ s1 \mapsto s1' \in e \ \wedge \ u(z') \mapsto p(s1') \in l) \ \wedge \\
\qquad (\exists s2, s2' \cdot q(s) = q(s2) \ \wedge \ s2 \mapsto s2' \in e \ \wedge \ v(z') \mapsto q(s2') \in m) \\
\qquad \Rightarrow \\
\qquad (\exists s' \cdot s \mapsto s' \in e \ \wedge \ u(z') \mapsto p(s') \in l \ \wedge \ v(z') \mapsto q(s') \in m)
\end{array}
$$

We have now to be more precise about the sets S, T, U, X, Y, and Z. This can be done by means of the following definitions:

$$
\begin{aligned}
S \ &\widehat{=} \ S1 \times S2 \times S3 \\
T \ &\widehat{=} \ S1 \times S2 \\
U \ &\widehat{=} \ \phantom{S1 \times {}} S2 \times S3 \\
X \ &\widehat{=} \ Z1 \times S2 \\
Y \ &\widehat{=} \ \phantom{Z1 \times {}} S2 \times Z3 \\
Z \ &\widehat{=} \ Z1 \times S2 \times Z3
\end{aligned}
$$

As can be seen, the interface between the two sub-models $\mathcal{N}$ and $\mathcal{P}$ is played by the set $S2$. We can also see that $S2$ *is not refined* by looking at the definitions of sets X and Y.

The projections p, q, u, v are instantiated accordingly:

$$p(s1 \mapsto s2 \mapsto s3) = s1 \mapsto s2$$

$$q(s1 \mapsto s2 \mapsto s3) = s2 \mapsto s3$$

$$u(z1 \mapsto s2 \mapsto z3) = z1 \mapsto s2$$

$$v(z1 \mapsto s2 \mapsto z3) = s2 \mapsto z3$$

As a consequence, the predicates $p(s) = p(s1)$ and $q(s) = q(s2)$ (seen in the expansion of the statement to prove above) become $p(s1 \mapsto s2 \mapsto s3) = p(s11 \mapsto s12 \mapsto s13)$ and $q(s1 \mapsto s2 \mapsto s3) = p(s21 \mapsto s22 \mapsto s23)$ respectively. Now these predicates can be simplified to $s1 = s11 \wedge s2 = s12$ and $s2 = s22 \wedge s3 = s23$ respectively.

For instantiating the refinement relations l, m, and n, we introduce two relations λ and μ:

$$\lambda \in Z1 \times S2 \leftrightarrow S1$$

$$\mu \in S2 \times Z3 \leftrightarrow S3$$

Next are now the instantiations of the refinement relations l, m and n:

$$(z1 \mapsto z2) \mapsto (s1 \mapsto s2) \in l \quad \widehat{=} \quad (z1 \mapsto z2) \mapsto s1 \in \lambda \ \wedge \ z2 = s2$$

$$(z2 \mapsto z3) \mapsto (s2 \mapsto s3) \in m \quad \widehat{=} \quad (z2 \mapsto z3) \mapsto s3 \in \mu \ \wedge \ z2 = s2$$

$$(z1 \mapsto z2 \mapsto z3) \mapsto (s1 \mapsto s2 \mapsto s3) \in n \quad \widehat{=} \quad (z1 \mapsto z2) \mapsto s1 \in \lambda \ \wedge \ (z2 \mapsto z3) \mapsto s3 \in \mu \ \wedge \ z2 = s2$$

Notice the various predicates $z2 = s2$. This takes account that the interface $S2$ is not refined.

It remains now for us to specialize the event e in two different ways.

First Specialization. First we consider a relation $e2$ leaving its component on set $S3$ unchanged, it is however modifying its components on sets $S1$ and $S2$. For this, we introduce a relation ϵ typed as follows:

$$\epsilon \in S1 \times S2 \leftrightarrow S1 \times S2$$

$$(s1 \mapsto s2 \mapsto s3) \mapsto (s1' \mapsto s2' \mapsto s3') \in e2 \quad \widehat{=} \quad (s1 \mapsto s2) \mapsto (s1' \mapsto s2') \in \epsilon \ \wedge \ s3 = s3'$$

This event $e2$ is typically the event a_2_b used in the first example. Here is again what we had to prove before the instantiations (we removed the external universal quantification):

$$\begin{array}{l}(\exists s1, s1' \cdot p(s) = p(s1) \ \wedge \ s1 \mapsto s1' \in e \ \wedge \ u(z') \mapsto p(s1') \in l) \ \wedge \\ (\exists s2, s2' \cdot q(s) = q(s2) \ \wedge \ s2 \mapsto s2' \in e \ \wedge \ v(z') \mapsto q(s2') \in m) \\ \Rightarrow \\ (\exists s' \cdot s \mapsto s' \in e \ \wedge \ u(z') \mapsto p(s') \in l \ \wedge \ v(z') \mapsto q(s') \in m)\end{array}$$

Let us instantiate these three predicates in turn. The instantiation of:

$$\exists s1, s1' \cdot p(s) = p(s1) \ \wedge \ s1 \mapsto s1' \in e \ \wedge \ u(z') \mapsto p(s1') \in l$$

is

$$\begin{array}{l} \exists s11, s12, s13, s11', s12', s13' \cdot \\ \quad\quad s1 = s11 \ \wedge \ s2 = s12 \ \wedge \\ \quad\quad (s11 \mapsto s12) \mapsto (s11' \mapsto s12') \in \epsilon \ \wedge \ s13 = s13' \ \wedge \\ \quad\quad (z1' \mapsto z2') \mapsto s11' \in \lambda \ \wedge \ z2' = s12' \end{array}$$

That is:

$$\underline{\exists s11' \cdot (s1 \mapsto s2) \mapsto (s11' \mapsto z2') \in \epsilon \ \wedge \ (z1' \mapsto z2') \mapsto s11' \in \lambda}$$

The instantiation of:

$$\exists s2, s2' \cdot q(s) = q(s2) \ \wedge \ s2 \mapsto s2' \in e \ \wedge \ v(z') \mapsto q(s2') \in m$$

is

$$\begin{array}{l} \exists s21, s22, s23, s21', s22', s23' \cdot \\ \quad\quad s2 = s22 \ \wedge \ s3 = s23 \ \wedge \\ \quad\quad (s21 \mapsto s22) \mapsto (s21' \mapsto s22') \in \epsilon \ \wedge \ s23 = s23' \ \wedge \\ \quad\quad (z2' \mapsto z3') \mapsto s23' \in \mu \ \wedge \ z2' = s22' \end{array}$$

that is

$$\underline{\exists s21, s21' \cdot (s21 \mapsto s2) \mapsto (s21' \mapsto z2') \in \epsilon \ \wedge \ (z2' \mapsto z3') \mapsto s3 \in \mu}$$

Finally, the instantiation of:

$$\exists s' \cdot s \mapsto s' \in e \ \wedge \ u(z') \mapsto p(s') \in l \ \wedge \ v(z') \mapsto q(s') \in m$$

is

$$\begin{array}{l} \exists s1', s2', s3' \cdot \\ \quad\quad (s1 \mapsto s2) \mapsto (s1' \mapsto s2') \in \epsilon \ \wedge \ s3 = s3' \ \wedge \\ \quad\quad (z1' \mapsto z2') \mapsto s1' \in \lambda \ \wedge \ z2' = s2 \\ \quad\quad (z2' \mapsto z3') \mapsto s3' \in \mu \ \wedge \ z2' = s2 \end{array}$$

That is:

$$\underline{\exists s1' \cdot (s1 \mapsto s2) \mapsto (s1' \mapsto z2') \in \epsilon \ \wedge \ (z1' \mapsto z2') \mapsto s1' \in \lambda \ \wedge \ (z2' \mapsto z3') \mapsto s3 \in \mu}$$

So that we have to prove the following:

$$\begin{array}{l} (\exists s11' \cdot (s1 \mapsto s2) \mapsto (s11' \mapsto z2') \in \epsilon \ \wedge \ (z1' \mapsto z2') \mapsto s11' \in \lambda \ \wedge \\ (\exists s21, s21' \cdot (s21 \mapsto s2) \mapsto (s21' \mapsto z2') \in \epsilon \ \wedge \ (z2' \mapsto z3') \mapsto s3 \in \mu) \\ \Rightarrow \\ (\exists s1' \cdot (s1 \mapsto s2) \mapsto (s1' \mapsto z2') \in \epsilon \ \wedge \ (z1' \mapsto z2') \mapsto s1' \in \lambda \ \wedge \ (z2' \mapsto z3') \mapsto s3 \in \mu) \end{array}$$

yielding:

$$\begin{array}{l} (s1 \mapsto s2) \mapsto (s11' \mapsto z2') \in \epsilon \ \wedge \ (z1' \mapsto z2') \mapsto s11' \in \lambda) \ \wedge \\ (s21 \mapsto s2) \mapsto (s21' \mapsto z2') \in \epsilon \ \wedge \ (z2' \mapsto z3') \mapsto s3 \in \mu) \\ \Rightarrow \\ (\exists s1' \cdot (s1 \mapsto s2) \mapsto (s1' \mapsto z2') \in \epsilon \ \wedge \ (z1' \mapsto z2') \mapsto s1' \in \lambda \ \wedge \ (z2' \mapsto z3') \mapsto s3 \in \mu) \end{array}$$

The result is obtained by instantiating $s1'$ with $s11'$.

Second Specialization. Now we specialize the event e by considering a relation $e1$ leaving its component on sets $S2$ and $S3$ unchanged: this event is just modifying its component on set $S1$:

$$(s1 \mapsto s2 \mapsto s3) \mapsto (s1' \mapsto s2' \mapsto s3') \in e1 \quad \widehat{=} \quad s1 \mapsto s1' \in \epsilon \ \wedge \ s2 = s2' \ \wedge \ s3 = s3'$$

This event $e1$ is typically the event in_a used in the first example. A treatment similar to the one we have done for $e2$ leads to the following to prove:

$$\begin{array}{l}
(\exists s11' \cdot s1 \mapsto s11' \in \epsilon \ \wedge \ (z1' \mapsto z2') \mapsto s11' \in \lambda \ \ \wedge s2 = z2') \ \wedge \\
(\exists s21, s21' \cdot s21 \mapsto s21' \in \epsilon \ \wedge \ (z2' \mapsto z3') \mapsto s3 \in \mu \ \wedge \ s2 = z2') \\
\Rightarrow \\
(\exists s1' \cdot s1 \mapsto s1' \in \epsilon \ \wedge \ (z1' \mapsto z2') \mapsto s1' \in \lambda \ \wedge \ (z2' \mapsto z3') \mapsto s3 \in \mu \ \wedge \ s2 = z2')
\end{array}$$

yielding:

$$\begin{array}{l}
s1 \mapsto s11' \in \epsilon \ \wedge \ (z1' \mapsto z2') \mapsto s11' \in \lambda \ \ \wedge s2 = z2' \ \wedge \\
s21 \mapsto s21' \in \epsilon \ \wedge \ (z2' \mapsto z3') \mapsto s3 \in \mu \ \wedge \ s2 = z2' \\
\Rightarrow \\
\exists s1' \cdot s1 \mapsto s1' \in \epsilon \ \wedge \ (z1' \mapsto z2') \mapsto s1' \in \lambda \ \wedge \ (z2' \mapsto z3') \mapsto s3 \in \mu \wedge s2 = z2'
\end{array}$$

The result is obtained by instantiating $s1'$ with $s11'$.