Chapter 1

Mandatory and potential choice: Comparing Event-B and STAIRS

Atle Refsdal
SINTEF ICT, Norway

Ragnhild Kobro Runde
University of Oslo, Norway

Ketil Stølen
SINTEF ICT, Norway and University of Oslo, Norway

Abstract. In order to decide whether a software system fulfills a specification, or whether a detailed specification preserves the properties of a more abstract specification, we need an understanding of what it means for one specification to fulfill another specification. This is particularly important when the specification contains one or more operators for expressing choice. Operators for choice have been studied for more than three decades within the field of formal methods in general, and within methods for action-refinement in particular. As a more recent method within this tradition, in this paper we have chosen to focus on Event-B. Another kind of method is STAIRS, focusing on UML interactions and designed to provide the UML community with an understanding of refinement and fulfillment. To provide the required expressiveness, STAIRS distinguishes between potential and mandatory choice, where only the latter is required to by preserved by refinement. In this paper, we investigate the relationship between the operators for choice in Event-B and STAIRS.

1.1 Introduction

In order to decide whether a software system fulfills a specification, we need a clear understanding of the concept of fulfillment. Similarly, when a specification is developed further into a new more detailed (for example, platform-specific) specification, the essential properties captured by the original specification must still be present in the new specification. This requires an understanding of what it means for one specification to fulfill another specification.

STAIRS [HHRS05, RRS13] was designed to provide the UML community with this kind of understanding at a level of abstraction that is easily comprehensible for UML practitioners. STAIRS is inspired by formal methods and refinement theory. However, STAIRS is not really a formal method in the classical sense, as explained in the following. When formal methods are combined with more applied methods for software engineering, the resulting approaches may typically be classified according to whether:

- Artifacts of the applied method, typically specifications and models, are translated into the formal method and used for formal analysis.
- Artifacts of the applied method, again normally specifications and models, are annotated with formal expressions and used for formal analysis building on some unified underlying semantics.

STAIRS does not fit within this classification scheme since the emphasis of STAIRS is to provide a foundation for fulfillment within the conceptual universe of UML rather than supporting formal analysis of UML specifications and their relationships.

STAIRS addresses primarily sequence diagrams. Implicitely, STAIRS also defines the notion of fulfillment for the other UML notations for modeling dynamic behavior, where the behaviour may be captured by sets of sequence diagrams. In many respects, sequence diagrams are more general than other UML notations for dynamic behavior, e.g., state machines, because sequence diagrams may be used to describe examples of required behavior, rather than the complete allowed behavior. STAIRS provides this expressiveness by offering operators for potential as well as mandatory choice.

Operators for choice have been studied for more than three decades within the field of formal methods in general, and within methods for action-refinement in particular [BS91]. A prominent example of a method for action-refinement is Event-B [Abr10]. The objective of this paper is to investigate the relationship between the operators for choice in Event-B and STAIRS.

A large literature exists on Event-B. There is no fixed semantics for Event-B, instead the semantics is provided implicitly by proof obligations associated with a model [Hal11]. Nevertheless, several papers have suggested failure-divergences inspired semantics as a formal underpinning [STW11, But12,

Preserved by refinement	**CSP**		**STAIRS**
	environment	**system**	
No		Internal choice Demonic choice	Potential choice
Yes	External choice Angelic choice		Mandatory choice

TABLE 1.1: Choice types in CSP and STAIRS

SB12]. This paper builds on this approach. Failure-divergences semantics was originally developed for CSP [Hoa85]. In Section 1.2 we therefore start our investigation by comparing choice in CSP to choice in STAIRS. Then we conduct a comparison of Event-B and STAIRS; first at the syntactic level in Section 1.3; then at the semantic level in Section 1.4. Finally, Section 1.5 provides a summary and draws conclusions.

1.2 Kinds of choice

In this section, we relate the kinds of choice offered by CSP [Hoa85] and STAIRS [HHRS05]. We also classify the kinds of properties that may or may not be captured depending on the available choice operators.

In order to understand the choice operators in CSP we need to understand some underlying assumptions about the involved entities, as well as the communication model. As explained by [Ros98, p. 13], in CSP a system[1] is completely described by the way it can communicate with its environment. Hence, CSP assumes a black-box view where internal communication within the system itself, is hidden. Communication is synchronous (also known as handshake communication), meaning that “events only happen when both sides agree” [Ros98, p. 9]. This can be understood as follows: At any given point the system offers a set of events to the environment. If the environment accepts one of these events then the system moves on, otherwise a deadlock occurs.

Choices made by the environment between available alternatives are called *external* choices and represented by the $\Box$ operator in CSP, while choices made by the system are called *internal*[2] and represented by the $\sqcap$ operator. If one of the alternatives offered to the environment is removed, a deadlock will be introduced if the environment is willing to synchronize only on the removed alternative. Refinement in CSP therefore requires preservation of ex-

[1]The CSP literature typically uses the term “process”.

[2]Hoare uses the term “nondeterministic or” or just “nondeterminism” for internal choice.

ternal choice. Internal choice, on the other hand, represents underspecification and may be reduced in a valid refinement step, as motivated by the following quote [Hoa85, p. 101–102]:

> Sometimes a process has a range of possible behaviours, but the environment of the process does not have any ability to influence or even observe the selection between the alternatives [. . .] The choice is made, as it were internally, by the machine itself, in an arbitrary or nondeterministic fashion [. . .]
>
> There is nothing mysterious about this kind of nondeterminism: it arises from a deliberate decision to ignore the factor which influence the selection [. . .] Thus nondeterminism is useful for maintaining a high level of abstraction in descriptions of the behaviour of physical systems and machines [. . .]
>
> A process specified as $(P \sqcap Q)$ can be implemented either by building P or by building Q. The choice can be made in advance by the implementor on grounds not relevant (and deliberately ignored) in the specification [. . .]

The term *angelic* choice (or angelic nondeterminism) is sometimes used to describe a choice that will always be made so that an undesirable result (a deadlock) is avoided if possible. Hoare explains this in terms of an implementation that, when choosing between P and Q, "minimises the risk of deadlock by delaying the choice until the environment makes it, and then selecting whichever of P and Q does *not* deadlock" [Hoa85, p. 105]. Similarly, Roscoe explains angelic choice in terms of an operator that "keeps on giving the environment the choice of action of P and Q as long as the environment picks an event they both offer" [Ros10, p. 219]. Hence, angelic choice is a special kind of external choice. Conversely, although not used in the above references, the term *demonic choice* can be used to describe an internal choice that will (or at least can) be made so that a deadlock will occur, if possible. In [MMSS96], Morgan et al. use the terms demonic choice and internal choice interchangeably.

The two middle columns of Table 1.1 summarize the above discussion. The system column represents choices resolved by the specified system, while the environment column represents choices resolved by its environment. The column furthest to the left indicates whether choices are preserved by refinement.

According to [HHRS05], STAIRS is an approach to compositional development of UML interactions that assigns a precise interpretation to the various steps in incremental system development based on an approach to refinement known from the field of formal methods. There are a couple of ways in which STAIRS differs from CSP of immediate relevance for our discussion of choice here. First, STAIRS assumes an asynchronous communication model with infinite buffering, and is therefore not concerned with deadlock. Second, in STAIRS there is no implicit hiding of internal communication when composing specifications.

STAIRS offers two different choice operators: one for *potential* choice and one for *mandatory* choice. Along the same lines as internal choice in CSP, potential choice is motivated by the need for abstraction. This is explained by the following requirement to STAIRS stated in [HHRS05]:

> Should allow specification of potential behavior. Underspecification is a well-known feature of abstraction. In the context of interactions, "under-specification" means specifying several behaviors, each representing a potential alternative serving the same purpose, and that fulfilling only some of them (more than zero but not all) is acceptable for an implementation to be correct.

Mandatory choice, on the other hand, is motivated as follows:

> Should allow specification of mandatory behavior [...] Sometimes [...] it is essential to retain non-determinism in the implementation reflecting choice. For example, in a lottery, it is critical that every lottery ticket has the possibility to win the prizes [...] As a consequence, we need to distinguish explicit non-determinism capturing mandatory behavior from non-determinism expressing potential behavior.

Since potential choice facilitates underspecification by offering alternatives serving the same purpose, STAIRS allows potential choice to be reduced or removed by refinement. A mandatory choice, on the other hand, needs to be preserved in order to ensure that all intended behavior will be implemented. This applies regardless of whether the choice is made by the system or by the environment. The distinction between potential and mandatory choice in STAIRS is summarized in the right-hand column of Table 1.1.

Another kind of choice is probabilistic choice, meaning that each alternative should be selected according to a given probability. Probabilistic choice is beyond the scope of this paper and has therefore not been included in Table 1.1. However, mandatory choice (as understood in STAIRS) can be understood as probabilistic choice where all of the probabilities should be higher than 0, but where nothing more is known/specificed about the probabilities.

The constructs for expressing choice offered by a specification language and its notion of refinement restrict the kinds of properties that can be captured. System properties are typically analyzed on the basis of system traces, each of which characterizes a possible run or execution. Properties can then be classified according to their means of falsification. Properties that can be falsified by a tester on the basis of a single trace are called *trace properties*, while properties that can be falsified on the basis of trace sets are called *trace set* properties [McL94]. The former include safety and liveness as originally investigated by Alpern and Schneider [AS85, Sch00]. The latter include information security flow properties and are what McLean referred to as possibilistic properties [McL94].

As an example, assume we want to specify a simulator to simulate user behavior for automatic testing of vending machines offering tea and coffee. The simulator should then be able to choose both alternatives, and the choice should be made internally by the simulator (thus reflecting a user's preference) rather than by its environment. Before using the simulator to automatically test a vending machine, we need to test the simulator itself. When testing the simulator, if we observe a single trace yielding tea, we cannot deduce that the simulator is not able to choose coffee or vice versa; such falsifications can only be made by considering all traces of the system.

Specification approaches allowing all choices made internally by the specified system (as opposed to its environment) to be reduced by refinement, have no means to ensure that trace set properties are preserved. This is referred to as the refinement paradox in [JÖ1]. In the following, we discuss the syntax and semantics of choice in Event-B and STAIRS in the light of trace properties and trace set properties.

1.3 Comparing Event-B and STAIRS at the syntactic level

The essence of an Event-B specification is a set of guarded events, where an event is enabled and may be chosen to occur when its guard is true. More than one event may be enabled at the same time, and the choice between enabled events is an external choice made by the environment. Internal choice made by the system itself is modeled more indirectly, using nondeterministic assignment to internal variables in order to influence the enabledness of other events.

As an example of how choice is treated in Event-B, in Fig. 1.1 we look at the two vending machine specifications given by Butler in [But12]. An Event-B specification consists of a specification name, a declaration and initialization of variables and a set of named events. Each event is on the form **when** *guard* **then** *body* **end**, where the guard is a boolean statement over the variables and the body is a (possibly non-deterministic) variable assignment. An event is said to be enabled if its guard evaluates to true, otherwise it is disabled.

The difference between the two specifications in Fig. 1.1 is that in $VM1$, the internal variable $m1$ is set to *vend* after the `Coin` event has been executed, thus enabling both the `Tea` and the `Coffee` event, while in $VM2$, the internal variable $m2$ is set to either *tea* or *coffee*, thus enabling only one of `Tea` and `Coffee`. This means that in $VM1$, the choice between `Tea` and `Coffee` is to be made by the environment, and is thus an example of external choice. In [But12], it is argued that from a customer's point of view, this external choice should be preserved by refinement, meaning that $VM2$ should not be

```
machine VM1
variables m1 ∈ {idle,vend}
initialisation
        m1 := idle
events
 Coin ≙        when
                 m1 = idle
               then
                 m1 := vend
               end
 Tea ≙         when
                 m1 = vend
               then
                 m1 := idle
               end
 Coffee ≙      when
                 m1 = vend
               then
                 m1 := idle
               end
```

```
machine VM2
variables m2 ∈ {idle,tea,coffee}
initialisation
        m2 := idle
events
 Coin ≙        when
                 m2 = idle
               then
                 m2 :∈ {tea,coffee}
               end
 Tea ≙         when
                 m2 = tea
               then
                 m2 := idle
               end
 Coffee ≙      when
                 m2 = coffee
               then
                 m2 := idle
               end
```

FIGURE 1.1: Two Event-B vending machines as specified by [But12]

a valid refinement of $VM1$. This could be achieved for instance by requiring that a refinement should preserve the enabledness of individual events.

In $VM2$, the choice between `Tea` and `Coffee` is made by the machine itself, and this is an example of internal choice. An internal choice may be refined by an external choice, as this ensures that all events enabled in the original machine will also be enabled in the refined one. Consequently, $VM1$ should be a valid refinement of $VM2$.

As an example of potential choice in STAIRS, Fig. 1.2 gives a sequence diagram specification of a vending machine with messages that correspond to the events in $VM1$ and $VM2$ from Fig. 1.1. The main ingredients of a sequence diagram are a set of lifelines (depicted as vertical lines) and a number of messages (arrows) between the lifelines. In Fig. 1.2, the choice operator `alt` is used to signify that this diagram specifies two example scenarios, both starting with the vending machine receiving a coin from the environment, followed by the vending machine providing tea in one scenario, coffee in the other. As `alt` is used to model potential choice, a sequence diagram where only one of these scenarios is positive, and the other one is specified as negative, would be a valid refinement of `Vending Machine 1`.

In STAIRS, there is no fundamental distinction between internal and external choice. The choice between sending `Tea` or `Coffee` in Fig. 1.2 is an internal choice when seen from the sending lifeline `VM`, and an external choice when seen from the receiving lifeline `Env`. For a real vending machine, the choice between tea and coffee would be made by the user. In STAIRS, this

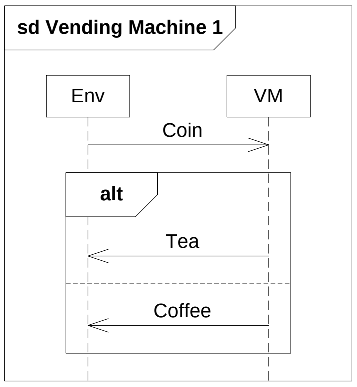

FIGURE 1.2: A simple vending machine with potential choice in STAIRS

may be modeled for instance by selection messages from `Env` to `VM` as seen in Fig. 1.3. Note, however, that the choice between the two alternatives is still specified using alt, meaning that in a valid implementation, only one of the tea and coffee scenarios may be present.

Back to Event-B, Butler [But12] argues that experience with Event-B modeling has demonstrated the need to be able to model both internal and external choice between enabled events more directly, in particular in situations where the guards are equal only as a result of abstraction. In reality, such a choice is not really external, but rather internal due to some condition not included in the abstract specification.

For instance, the choice between `Tea` and `Coffee` in $VM1$ in Fig. 1.1 should in some cases be seen as internal due to some condition abstracted away in $VM1$. A refinement may for instance add internal variables and guards so that coffee is always served in the morning, while tea is always served in the afternoon.

In [But12], the main goal is to allow both external and internal choice to be represented directly. This is achieved by letting the specifier divide the events into groups. The intuitive interpretation is that a choice between groups of events is external, while a choice between events within a group is internal. For $VM1$ in Fig. 1.1, the specifier may state that the choice between `Tea` and `Coffee` is internal by grouping them together, giving the following event groups for $VM1$: $G1 = \{\texttt{Coin}\}$, $G2 = \{\texttt{Tea}, \texttt{Coffee}\}$.

In [But12], the refinement relation is modified so that preservation of enabledness is preserved for event groups rather than for single events, thus ensuring that external choices are preserved (or increased) through refinement while at the same time allowing the amount of internal choice to be reduced. With the event groups $G1$ and $G2$ given above, this would mean that a valid refinement of $VM1$ may choose to offer only `Coffee` (or `Tea`).

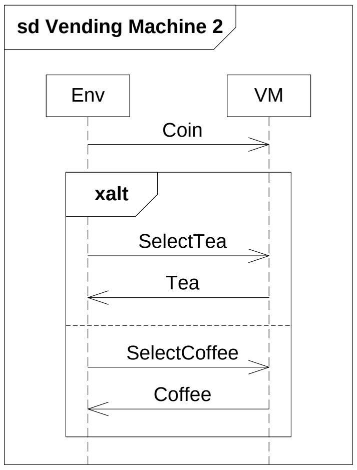

FIGURE 1.3: A simple vending machine with external choice in STAIRS

Mandatory choice is not discussed in [But12], and the introduction of event groups is not sufficient to capture system choices that must be preserved by refinement, i.e., trace set properties. Assume, for illustration purposes, that the specifier wants to model a machine which arbitrarily chooses between tea and coffee at run-time (similar to the user simulator described in Section 1.2). As putting `Tea` and `Coffee` in the same event group might lead to an implementation offering only one of them as seen above, the only other possibility is to put them in separate event groups. However, the semantics of such a specification would allow `Tea` (and similarly, `Coffee`) to be refused only when its guard is false, meaning that neither `Tea` nor `Coffee` could be refused after `Coin`, so that the choice between the two remains external and not internal.

A vending machine where the internal choice is made arbitrarily as described above, may be modeled in STAIRS by using the mandatory choice operator xalt as shown in Fig. 1.4. To simplify the main diagram `Vending Machine 3`, the diagram refers to two sub-diagrams `Provide tea` (also provided in Fig. 1.4) and `Provide coffee` (not shown, but symmetrical to `Provide tea`). The refuse operator is used to model that a specific alternative should be considered negative, e.g., in `Provide tea` the vending machine should serve tea and not coffee. The main diagram `Vending Machine 3` then requires the vending machine to have two mandatory behaviors, one with tea and not coffee, and one with coffee and not tea. Neither of these can be removed by refinement. Also, the mandatory choice in `Vending Machine 3` in Fig. 1.4 is a valid refinement of the potential choice in `Vending Machine 1` in Fig. 1.2, as should be expected. Further refinements may increase the mandatory behavior required by adding more xalt-operands, e.g., a third alternative providing chocolate but not tea or coffee.

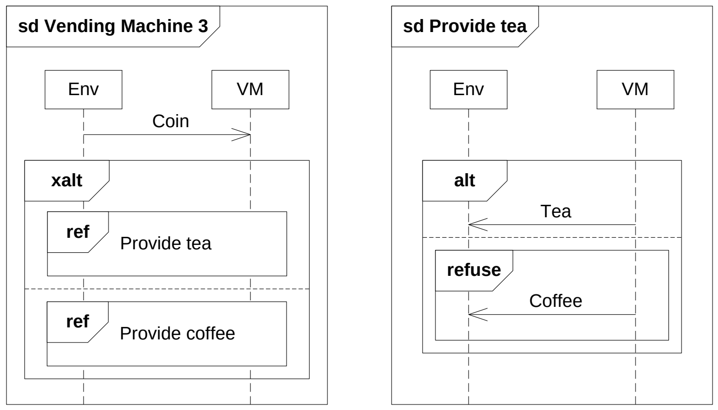

FIGURE 1.4: Mandatory choice in STAIRS

1.4 Interaction-obligations versus failure-divergences

In the previous section, we compared Event-B and STAIRS at the syntactic level. Semantically, an Event-B specification may be represented by a failure-divergences pair while a sequence diagram in STAIRS corresponds to a set of interaction-obligations. If the sequence diagram does not contain mandatory choice, a single interaction-obligation is sufficient. In the following, we outline the intuition behind interaction-obligations and failure-divergences and how they are related. We also explain how mandatory choice is represented semantically and discuss the relationship to external choice.

1.4.1 Interaction-obligations

A trace in STAIRS is a finite or infinite sequence of events where an event is either the sending or the reception of a message. A trace is required to fulfill certain well-formedness conditions [RHS05]. Informally, a trace is well-formed if, for each message, the send event is ordered before the corresponding receive event.

Let $\mathcal{H}$ denote the set of all well-formed traces. An interaction-obligation is a pair (p, n) of trace-sets which classifies the elements of $\mathcal{H}$ into three categories: the positive traces p, the negative traces n, and the inconclusive traces $\mathcal{H} \setminus (p \cup n)$. The inconclusive traces are those traces that are neither specified as positive nor as negative by the sequence diagram in question.

A pre-post specification $(pre, post)$ in Hoare-logic may be used as a first approximation of the intuition behind an interaction-obligation (p, n). Roughly speaking:

- A positive trace (in p) corresponds to an execution initiated in a state fulfilling *pre* that if it terminates, does so in state fulfilling *post* (given Hoare-logic for partial correctness).
- An inconclusive trace (in $\mathcal{H}\backslash(p\cup n)$) corresponds to an execution initiated in a state fulfilling $\neg pre$.
- A negative trace (in n) corresponds to an execution initiated in a state fulfilling *pre* that terminates in state fulfilling $\neg post$.

It is worth noticing that while the inconclusive behavior corresponding to a pre-post specification is chaotic, meaning that anything is allowed, the inconclusive behavior of an interaction-obligation is not necessarily so. For example, if the finite trace t is inconclusive then the result $t \frown t'$ of extending t with t' is not necessarily inconclusive; $t \frown t'$ may be positive ($t \notin p \cup n \wedge t \frown t' \in p$) and another extension t'' may be negative ($t \notin p \cup n \wedge t \frown t'' \in n$). In fact, an interaction-obligation may for the same environment behavior allow inconclusive, positive as well as negative behavior. A pre-post specification on the other hand, classifies executions initiated in a state (in a pre-post setting, representing the environment behavior) as either positive or negative if it fulfills *pre* and as inconclusive, otherwise.

It is also worth mentioning that for interaction-obligations as for pre-post specifications, there may be environment behaviors for which no behavior is allowed. An example of a pre-post specification of this kind is

$$(\ \mathit{true}\ ,\ x = 0 \Rightarrow \mathit{false}\ )$$

It disallows any behavior for the initial state $x = 0$. In classical Hoare-logic, such a specification is not implementable because any real program has some kind of behavior whatever the environment does. In other words, no real program is partial. Hence, any implementable pre-post specification is total; it allows at least one system behavior for each possible initial state. The same is not true for interaction-obligations because sequence diagrams only specify example runs and not the full behavior of a real program. Hence, a sequence diagram only considering some input behaviors is unproblematic from a methodological point of view.

In Hoare-logic, refinement corresponds to weakening the pre-condition and strengthening the post-condition. Refinement of an interaction-obligation corresponds to reducing inconclusive behavior and redefining positive behavior as negative. Formally:

Definition 1 *An interaction-obligation* (p', n') *refines an interaction-obligation* (p, n) *if* $p \subseteq p' \cup n'$ *and* $n \subseteq n'$.

Given the mapping to pre-post specifications outlined above, reducing inconclusive behavior may be understood as weakening the pre-condition; redefining positive behavior as negative may be understood as strengthening the post-condition. Hence, refinement of interaction-obligations reflects very well refinement of pre-post specifications.

1.4.2 Failure-divergences

In the setting of Event-B, a specification may be described by a pair (f, d) of a set of failures f and a set of divergences d. A *failure* is a pair (t, X) of a finite trace t and a set of events X that the specified system may refuse after having engaged in the external interaction corresponding to t. In other words, the specified system may deadlock after having engaged in t if offered only X or a subset of X by the environment. A *divergence* is a trace t after which the systems may diverge, meaning that it performs an infinite unbroken sequence of internal (and hence invisible) actions without any external communication happening at all, also referred to as livelock. Well-formedness constraints [Hoa85, p. 130] are imposed on failure-divergence pairs that imply that any such pair is total; it allows some behavior (possibly consisting of doing nothing) whatever the environment does.

Failures-divergences refinement corresponds to removing failures and divergences. This means set inclusion with respect to the failures and the divergences. Formally:

Definition 2 *A failures-divergences pair (f', d') refines a failures-divergences pair (f, d) iff $f' \subseteq f$ and $d' \subseteq d$.*

External choice cannot be reduced by refinement, as this would imply adding new failures in order to allow the specified system to refuse some of the events offered to the environment according to the more abstract specification.

1.4.3 Relating the two models

In the case of total correctness the relationship between an Event-B specification captured by (f, d) and a sequence diagram captured by the interaction-obligation (p, n) may be characterized as follows:

- The positive behavior p corresponds to $e \setminus d$, where $e = \{t \mid (t, \emptyset) \in f\}$.
- The inconclusive behavior $\mathcal{H} \setminus (p \cup n)$ corresponds to d.
- The negative behavior n corresponds to $\mathcal{H} \setminus (e \cup d)$.

Given the mapping above, reducing inconclusive behavior in STAIRS may be understood as reducing the set of divergences, while redefine positive behavior as negative in STAIRS may be understood as reducing the set of traces that are not divergences. This mapping is not information preserving since semantically different failure-divergences are mapped to the same interaction-obligation. In particular, the semantic difference between external and internal choice disappears.

Contrary to a failure-divergences pair, an interaction-obligation may be partial in the sense that there may exist environment behavior for which no positive system behavior is defined. It may be argued that a refinement should not impose additional constraints on the environment behavior and thereby increase partiality. Although this constraint may easily be imposed, it is not enforced by STAIRS because there are situations where this is not very practical. We may for example use the operator for potential choice to specify two different protocols for interaction with the environment and then leave it to the implementor to select which one to use. This choice will also restrict (or impose additional assumptions about) the behavior of the environment because the specified system will only work properly if the environment sticks to the selected protocol.

1.4.4 Sets of interaction-obligations

Sequence diagrams with mandatory choice is in STAIRS represented by a set of interaction-obligations. Informally speaking, each interaction-obligation represents an alternative that must be reflected in any correct implementation. In the most general case, refinement corresponds to:

Definition 3 *A set of interaction-obligations o' refines a set of interaction-obligations o if for each interaction-obligation $(p, n) \in o$ there is an interaction-obligation $(p', n') \in o'$ such that (p', n') refines (p, n).*

Hence, each interaction-obligation at the more abstract level must be refined by at least one interaction-obligation at the more concrete level. On the other hand, o' may have interaction-obligations that do not refine any of those is o. In the STAIRS literature this notion of refinement is called general refinement. A more restrictive version is limited refinement which also requires each of the more concrete interaction-obligations to be a refinement of at least one abstract one at the more abstract level.

In the mapping from failure-divergences to interaction-obligations defined in the sub-section above, we lost the distinction between external and internal choice. When mapping failure-divergences to sets of interaction-obligations we have the expressiveness required to keep this distinction. Roughly speaking, each external choice alternative corresponds to a separate interaction-obligation as outlined by the example in Figure 1.5. The assert, which is a standard UML 2.x operator, makes any inconclusive trace in its body negative. From the perspective of the System lifeline, the diagram captures an external choice between receiving either a or b. If neither behavior-if-a nor behavior-if-b contain xalt-operators, the semantics of the diagram is a pair of two interaction-obligations; one corresponding to receiving a and one corresponding to receiving b.

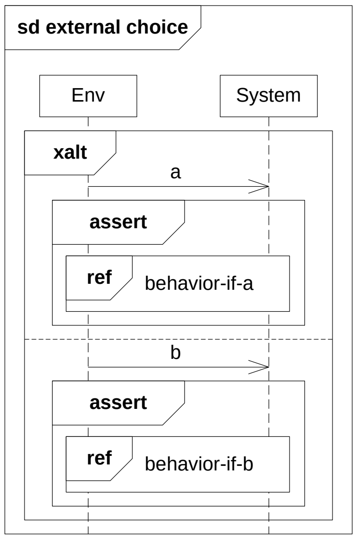

FIGURE 1.5: External choice represented by a sequence diagram in STAIRS

1.5 Conclusion

This paper compares Event-B (with a failure-divergences semantics) and STAIRS with particular focus on mandatory and potential choice. While the failure-divergences semantics gives a pure black-box interpretation of the specified system, STAIRS offers a white-box interpretation in terms of interaction-obligations and sets of interaction-obligations. Sets of interaction-obligations are required to capture mandatory choice while a single interaction-obligation is sufficient to model potential choice.

The main inspiration for writing this paper was Butler's proposal to capture external and internal choice directly by letting the specifier divide the events into groups. The approach seemed to resemble our proposal to capture mandatory and potential choice by sets of interaction-obligations.

Our conclusion is that it does. The expressivity offered by Butler's proposal is also provided by STAIRS. In the same sense as a single event group captures internal choice, single interaction-obligations captures potential choice. When potential choice is restricted to the specified system, internal and potential choice is the same – both represent underspecification. Moreover, in the same sense as sets of event groups capture external choice, sets of interaction-obligations capture mandatory choice. When mandatory choice is restricted to the environment, external and mandatory choice is the same – both represent nondeterminism that must be preserved by refinement.

Acknowledgments. This work has been conducted as a part of the DIAMONDS (201579) project and the AGRA (236657) project, both funded by the Research Council of Norway, and the CONCERTO (232059) project funded by the Research Council of Norway and by ARTEMIS Joint Undertaking - a public private partnership in the field of embedded systems supported under the Seventh Framework Programme of the European Commission.

Bibliography

[Abr10] J.-R. Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, 2010.

[AS85] B. Alpern and F. B. Schneider. Defining liveness. *Information Processing Letters*, 21(4):181 – 185, 1985.

[BS91] R.-J. Back and K. Sere. Stepwise refinement of action systems. *Structured Programming*, 12(1):17–30, 1991.

[But12] M. Butler. External and internal choice with event groups in Event-B. *Formal Aspects of Computing*, 24(4-6):555–567, 2012.

[Hal11] S. Hallerstede. On the purpose of Event-B proof obligations. *Formal Aspects of Computing*, 23(1):133–150, 2011.

[HHRS05] Ø. Haugen, K. E. Husa, R. K. Runde, and K. Stølen. STAIRS towards formal design with sequence diagrams. *Journal of Software and Systems Modeling*, 4:355–367, 2005.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[JÖ1] J. Jürjens. Secrecy-preserving refinement. In *Proceedings of Formal Methods Europe (FME'01)*, volume 2021 of *Lecture Notes in Computer Science*, pages 135–152. Springer, 2001.

[McL94] J. McLean. A general theory of composition for trace sets closed under selective interleaving functions. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 79–93. IEEE Computer Society, 1994.

[MMSS96] C. Morgan, A. McIver, K. Seidel, and J.W. Sanders. Refinement-oriented probability for CSP. *Formal Aspects of Computing*, 8(6):617–647, 1996.

[RHS05] R. K. Runde, Ø. Haugen, and K. Stølen. Refining UML interactions with explicit and implicit nondeterminism. *Nordic Journal of Computing*, 12:157–158, 2005.

[Ros98] A. W. Roscoe. *The Theory and Practice of Concurrency*. Prentice Hall, 1998.

[Ros10] A. W. Roscoe. *Understanding Concurrent Systems.* Springer, 2010.

[RRS13] R. K. Runde, A. Refsdal, and K. Stølen. Relating computer systems to sequence diagrams: the impact of underspecification and inherent nondeterminism. *Formal Aspects of Computing*, 25(2):159–187, 2013.

[SB12] R. Silva and M. Butler. Shared event composition/decomposition in Event-B. In B. K. Aichernig, F. S. de Boer, and M. M. Bonsangue, editors, *Formal Methods for Components and Objects*, volume 6957 of *Lecture Notes in Computer Science*, pages 122–141. Springer, 2012.

[Sch00] F. B. Schneider. Enforceable security policies. *ACM Transactions on Information and Systems Security*, 3(1):30–50, February 2000.

[STW11] S. Schneider, H. Treharne, and H. Wehrheim. A CSP account of Event-B refinement. In *Proceedings 15th International Refinement Workshop, Refine 2011, Limerick, Ireland, 20th June 2011.*, pages 139–154, 2011.