Underspecification, Inherent Nondeterminism and Probability in Sequence Diagrams

Atle Refsdal[1,2], Ragnhild Kobro Runde[1], and Ketil Stølen[1,2]

[1] Department of Informatics, University of Oslo, Norway
[2] SINTEF ICT, Norway

Abstract. Nondeterminism in specifications may be used for at least two different purposes. One is to express underspecification, which means that the specifier for the same environment behavior allows several alternative behaviors of the specified component and leaves the choice between these to those responsible for implementing the specification. In this case a valid implementation will need to implement at least one, but not necessarily all, alternatives. The other purpose is to express inherent nondeterminism, which means that a valid implementation needs to reflect all alternatives. STAIRS is an approach to the compositional and incremental development of sequence diagrams supporting underspecification as well as inherent nondeterminism. Probabilistic STAIRS builds on STAIRS and allows probabilities to be included in the specifications. Underspecification with respect to probabilities is also allowed. This paper investigates the use of underspecification, inherent nondeterminism and probability in sequence diagrams, the relationships between these concepts, and how these are expressed in STAIRS and probabilistic STAIRS.

1 Introduction

Nondeterminism in specifications may be used for expressing underspecification as well as inherent nondeterminism. Underspecification means that the specifier leaves some freedom of choice to those who will implement or further refine the specification. This is useful when different design alternatives fulfill a function equally well from the specifier's point of view. For example, when specifying an automatic teller machine we need to ensure that money is delivered and the card is returned at the end of the transaction. But whether the card is returned before or after the money is not important, and we may leave the choice to those responsible for making the teller machine.

Inherent nondeterminism, on the other hand, means that all alternatives must be reflected also in the final implementation. For example, when specifying a program to simulate a coin flip it is essential that both heads and tails are possible outcomes. An inherently nondeterministic choice can be seen as an abstraction of a probabilistic choice where the probabilities are greater than 0 but otherwise unknown.

The difference between underspecification and inherent nondeterminism is related to refinement. In an implementation, which is not supposed to be refined and has no underspecification, the distinction is not relevant.

R. Gorrieri and H. Wehrheim (Eds.): FMOODS 2006, LNCS 4037, pp. 138–155, 2006.

STAIRS ([HHRS05b],[RHS05c]) is a method for the compositional development of interactions, such as sequence diagrams and interaction overview diagrams. STAIRS employs two different choice operators to distinguish between underspecification and inherent nondeterminism; alt represents underspecification and xalt represents inherent nondeterminism. Probabilistic STAIRS ([RHS05a]) replaces xalt with the palt operator that also allows specification of probabilities on its operands.

STAIRS includes all the main composition operators of UML 2.0 interactions, such as seq and par for specifying sequential and parallel composition respectively. As these operators are not important for the discussion in this paper, we refer to [HHRS05b] for formal definitions and examples using these operators.

This paper summarizes insights gained during our work with formalization of various forms of nondeterminism in STAIRS and probabilistic STAIRS by investigating the different roles of nondeterminism in interactions. In particular we

- demonstrate the usefulness of underspecification, inherent nondeterminism and probability in specifications,
- show that these concepts are adequately expressed in STAIRS and probabilistic STAIRS by the operators alt, xalt and palt,
- explore the properties of these operators, in particular with respect to refinement,
- provide simple examples that give a thorough understanding of the use of these operators, both separately and combined.

The paper is organized as follows: Section 2 discusses underspecification and its representation in a simplified version of STAIRS. In Section 3 we motivate the need for inherent nondeterminism and show how this is incorporated in full STAIRS. Section 4 introduces probabilistic STAIRS. We discuss related work in Section 5 before concluding in Section 6.

2 Underspecification

2.1 Motivation

Often, it is useful to write specifications where certain aspects of the behavior of the system are left open. This is known as underspecification. In many cases, underspecification will be an implicit consequence of using abstraction when describing the important features of a system. Many specification languages also include some kind of 'or' operator for explicitly specifying the alternatives the implementer may choose between. In STAIRS, this is the alt operator.

In our setting of interactions, the alt operator may be used to describe scenarios that are different, but still seen as alternative means to achieve the same purpose in some sense. The alt operator is also called potential choice, as the alternatives represent choices that the implementation may choose between in order to satisfy the specification. As an everyday example, consider the action of making a u-turn when walking. This may be achieved by turning either 180 degrees left or 180 degrees right. Which alternative you choose is usually insignificant.

2.2 Semantic Representation

In STAIRS the semantics of an interaction is defined by denotational trace semantics, where a trace is a sequence of events representing a system run. We denote the semantics of an interaction d by $[\![d]\!]$. For the subset of STAIRS presented so far, containing only underspecification (and not inherent nondeterminism) the semantics of an interaction is represented by an *interaction obligation* (p, n). Here, p is a set of positive traces, representing desired or acceptable behavior, while n is a set of negative traces, representing undesired or unacceptable behavior.

An interaction is a partial specification in the sense that it does not in general define all the behavior of the system. Traces that are neither positive nor negative are called inconclusive, meaning that these are traces that the specifier has not yet considered. Letting $\mathcal{H}$ denote the universe of all traces, the traces $\mathcal{H} \setminus (p \cup n)$ are inconclusive in the obligation (p, n).

From an implementation point of view, there is no distinction between inconclusive and positive traces, as they all represent possible behaviors of the system. However, conceptually there is an important difference between behaviors that are explicitly described and behaviors that are not. Also, positive and inconclusive traces are treated differently by composition operators such as seq (sequential composition) and par (parallel composition), see [HHRS05b].

Underspecification by means of alt corresponds to taking the pairwise union of the positive and negative trace-sets of the operands. Formally:

$$[\![d_1 \ \mathsf{alt} \ d_2]\!] \stackrel{\mathsf{def}}{=} [\![d_1]\!] \uplus [\![d_2]\!] \tag{1}$$

where

$$(p_1, n_1) \uplus (p_2, n_2) \stackrel{\mathsf{def}}{=} (p_1 \cup p_2, n_1 \cup n_2) \tag{2}$$

From this definition it is clear that the alt operator can be used not only to introduce underspecification in the form of alternative ways of fulfilling a task (i.e. new positive traces), but also to introduce more restrictions by adding new negative traces. By taking the union also of the negative traces, the alt operator can be used to merge alternatives that are considered to be similar, both at the positive and the negative level. In addition, the above definition ensures monotonicity of refinement with respect to alt, which will be clear from the following sections.

2.3 Refinement

Refinement of a specification means to reduce underspecification by adding information so that the specification becomes closer to an implementation. As in [HHRS05b], we distinguish between two special cases of refinement, called narrowing and supplementing. Narrowing reduces the set of positive traces to capture new design decisions or to match the problem more accurately. Supplementing categorizes (to this point) inconclusive behavior as either positive

or negative. Formally, an interaction obligation (p', n') is a refinement of an interaction obligation (p, n), written $(p, n) \rightsquigarrow (p', n')$, iff

$$n \subseteq n' \wedge p \subseteq p' \cup n' \tag{3}$$

Intuitively, supplementing means that it is possible to add new positive or negative traces to those already specified. Specifying more alternative traces is usually achieved by using the alt operator, meaning that we want d_1 alt d_2 to be a valid refinement of d_1 (and of d_2). As negative traces must remain negative in a refinement, this means that d_1 alt d_2 must include the negative traces of both d_1 and of d_2, as in equation 2 above.

2.4 Simple Example

We now give a simple example of underspecification and refinement. Figure 1 specifies the game of tic-tac-toe between a player and the system. Either the player or the system may make the first move, and this is specified using alt. The player and the system then alternate making moves until the game is over. The opt operator is a shorthand for an alt with an empty second operand, while loop(2,3) may be interpreted as an alt between performing the contents of the loop two and three times. For formal definitions of opt and loop, see [RHS05c]. The game is finished after minimum five and maximum eight moves, depending on how many times the loop is executed, and whether the move inside opt is performed or not. (A ninth move is never really necessary, as the result of the game will be given at latest after the eight move.) We have omitted the details describing the exact positions taken in each move.

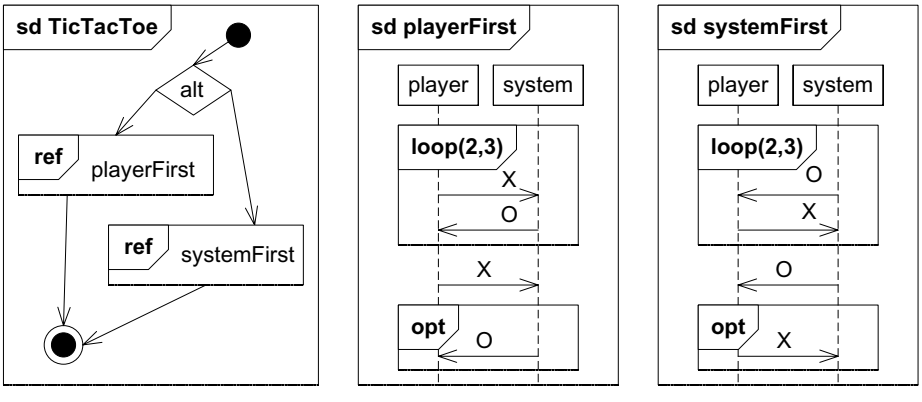

Fig. 1. Playing tic-tac-toe

In TicTacToe, the choice of who gets the first move is an example of underspecification. A possible refinement could be to use narrowing in order to remove this underspecification, as in TicTacToe2 where the player always moves first:

TicTacToe2 = (playerFirst) alt (refuse systemFirst)

where the operator refuse intuitively means that all traces defined by its argument should be considered negative. (For a formal definition of refuse, se Section 3.2.) A further refinement could be to add behavior to the specification by e.g. defining that traces where the system makes a second move before the player gets to do his/her move, are negative. These behaviors were inconclusive in TicTacToe2 (and TicTacToe), making this an example of supplementing.

2.5 Properties of alt and Refinement

As can be expected, the operator alt is

- associative: d_1 alt (d_2 alt d_3) = (d_1 alt d_2) alt d_3
- commutative: d_1 alt $d_2 = d_2$ alt d_1

This follows trivially from the associativity and commutativity of $\cup$.

As proved in [HHRS05a], we also have that the refinement operator $\rightsquigarrow$ is

- reflexive: $(p, n) \rightsquigarrow (p, n)$
- transitive: $(p, n) \rightsquigarrow (p', n') \wedge (p', n') \rightsquigarrow (p'', n'') \Rightarrow (p, n) \rightsquigarrow (p'', n'')$
- monotonic with respect to alt:
  $[\![d_1]\!] \rightsquigarrow [\![d_1']\!] \wedge [\![d_2]\!] \rightsquigarrow [\![d_2']\!] \Rightarrow [\![d_1 \text{ alt } d_2]\!] \rightsquigarrow [\![d_1' \text{ alt } d_2']\!]$

3 Inherent Nondeterminism

3.1 Motivation

Underspecification gives rise to nondeterminism, as the system behavior is not completely determined by the specification. Still, nondeterminism in the sense of underspecification does not require that the implementation itself should be nondeterministic. Sometimes, however, it is desirable to specify nondeterminism that *must* be present also in the implementation. We call this *inherent nondeterminism*. The throwing of a dice is an example of a process we would specify as inherently nondeterministic. Another example is a password generator, that should select passwords nondeterministically, at least from the perspective of the user (and the attacker). Inherent nondeterminism is in fact also essential in the domain of (information) security, see [Ros95].

As inherent nondeterminism and underspecification impose different requirements on an implementation, they should be described differently both in the syntax and the semantics of interactions. In STAIRS, inherent nondeterminism is specified by the use of the operator xalt. The xalt operator is also called mandatory choice, as the implementation must be able to perform (i.e. choose) any one of the given alternatives.

3.2 Semantic Representation

In Section 2.2 we represented the semantics of a STAIRS specification with underspecification as an interaction obligation (p, n). With this simple semantics,

it is not possible to express cases where *all* alternatives need to be present in an implementation, as traces could be moved from positive to negative by means of refinement. For STAIRS specifications with both underspecification and inherent nondeterminism, we therefore extend the semantics to be a *set* of interaction obligations. The interpretation is that for each interaction obligation (p_i, n_i) a valid implementation needs to be able to produce at least one trace allowed by (p_i, n_i). Intuitively, each interaction obligation (p_i, n_i) defines an inherently nondeterministic alternative that needs to be implemented, but exactly how this should be achieved is underspecified, since $\mathcal{H} \setminus n_i$ is a set. This leads us to the following formal definition of xalt:

$$[\![\ d_1 \ \mathsf{xalt}\ d_2\ ]\!] \stackrel{\mathsf{def}}{=} [\![\ d_1\ ]\!] \cup [\![\ d_2\ ]\!] \tag{4}$$

We now define the operator refuse, informally explained in Section 2.4, and generalize the definition of alt to cover operands with several interaction obligations:

$$[\![\ \mathsf{refuse}\ d\ ]\!] \stackrel{\mathsf{def}}{=} \{(\varnothing, p \cup n) \mid (p, n) \in [\![\ d\ ]\!]\} \tag{5}$$

$$[\![\ d_1\ \mathsf{alt}\ d_2\ ]\!] \stackrel{\mathsf{def}}{=} \{(p_1 \cup p_2, n_1 \cup n_2) \mid (p_1, n_1) \in [\![\ d_1\ ]\!] \wedge (p_2, n_2) \in [\![\ d_2\ ]\!]\} \tag{6}$$

3.3 Refinement Revisited

The whole point of inherent nondeterminism in a specification is to ensure that the alternatives are preserved during refinement. Since each interaction obligation represents an inherently nondeterministic alternative, we need to ensure that each interaction obligation from the abstract specification will be represented also in the more concrete specification. Formally, a specification d' is a refinement of a specification d, written $d \rightsquigarrow d'$, iff

$$\forall\, o \in [\![\ d\ ]\!] : \exists\, o' \in [\![\ d'\ ]\!] : o \rightsquigarrow o' \tag{7}$$

where $o \rightsquigarrow o'$ is refinement of interaction obligations as given by definition 3.

3.4 Simple Example

As an example, we consider a so-called 'randomizer' that should provide nondeterministic output selected randomly. Figure 2 gives a specification where the randomizer simulates the flipping of a coin, where both heads and tails should be possible outcomes.

Textually, we may write the Coin specification and its semantics as:

$$\begin{array}{lcl} \text{Coin} & = & (\text{heads } \mathsf{alt}\ (\mathsf{refuse}\ \text{tails})) \ \mathsf{xalt}\ (\text{tails } \mathsf{alt}\ (\mathsf{refuse}\ \text{heads})) \\ [\![\ \text{Coin}\ ]\!] & = & \{\ (\{h\}, \{t\}), (\{t\}, \{h\})\ \} \end{array}$$

where h denotes the trace(s) where the outcome is heads and t denotes the trace(s) where the outcome is tails. This semantics is illustrated in the bottom

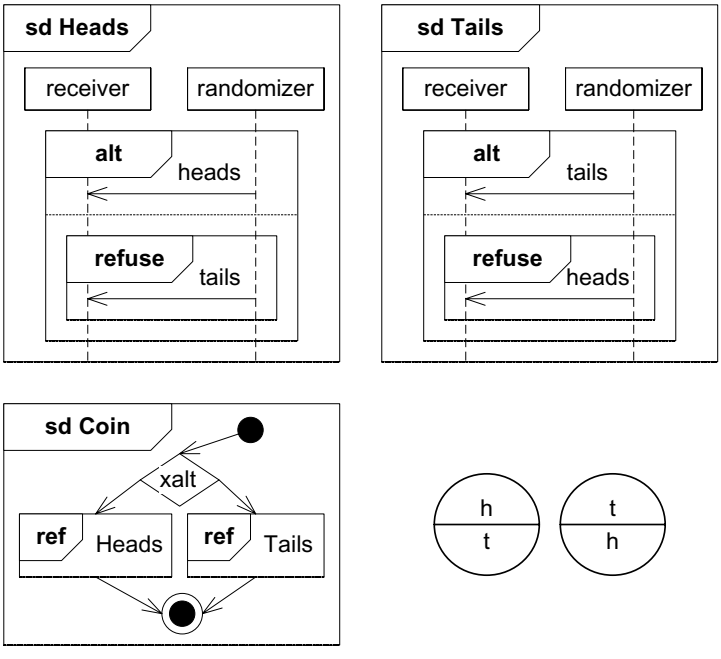

Fig. 2. The coin specification. Semantic representation to the bottom right.

right of Figure 2, where each circle represents an interaction obligation with the positive traces in the upper half and the negative traces in the lower half.

As another example, we specify how throwing a dice may simulate the flipping of a coin. One way of doing this is to let odd numbers represent heads, and even numbers represent tails. This is expressed by the specification

$$\text{DiceCoin} = \text{Throw135 } \mathsf{xalt} \text{ Throw246}$$

where Throw135 specifies a throw resulting in an odd number and Throw246 specifies a throw resulting in an even number:

$$\text{Throw135} = (1 \text{ alt } 3 \text{ alt } 5) \text{ alt } (\text{refuse } (2 \text{ alt } 4 \text{ alt } 6))$$
$$\text{Throw246} = (2 \text{ alt } 4 \text{ alt } 6) \text{ alt } (\text{refuse } (1 \text{ alt } 3 \text{ alt } 5))$$

Using the given definitions of alt and xalt, we thereby get:

$$[\![\text{ DiceCoin }]\!] = \{ \ (\{1,3,5\},\{2,4,6\}),(\{2,4,6\},\{1,3,5\}) \ \}$$

As should be expected, this semantics tells us that when using a dice to simulate a coin, the dice should at least be able to produce one of the numbers 1, 3 and 5 (representing heads) and one of the numbers 2, 4 and 6. However,

it is not significant that all numbers may be produced, and DiceCoin may be implemented by the unfair dice DiceCoin2 giving only the numbers 1 and 6:

$$[\![\ \text{DiceCoin2}\ ]\!] = \{\ (\{1\},\{2,3,4,5,6\}),(\{6\},\{1,2,3,4,5\})\ \}$$

We see that DiceCoin2 is a valid refinement of DiceCoin, as each obligation of DiceCoin is refined into an obligation of DiceCoin2 where some of the positive behaviors have been redefined as negative (i.e. narrowed).

3.5 Relating xalt to alt

It is interesting to investigate what kinds of specifications we get by combining the operators for underspecification (i.e. alt) and inherent nondeterminism (i.e. xalt). We have already seen examples of alt within xalt in DiceCoin and DiceCoin2 in the previous section. It remains to investigate the use of xalt within one or both of the operands of alt.

A possible refinement of the Coin specification in Figure 2, is to strengthen the specification by stating that the coin should never land on the side. As landing on the side is negative both in the heads and the tails alternative, this behavior may be added by using alt as the top-level operator as in Coin2:

$$\begin{aligned} \text{Coin2} &= \text{Coin alt (refuse side)} \\ [\![\ \text{Coin2}\ ]\!] &= \{\ (\{h\},\{t,s\}),(\{t\},\{h,s\})\ \} \end{aligned}$$

where s denotes the trace(s) where the coin lands on the side. As the example demonstrates, alt may in general be used to add (i.e. supplement) the same positive and/or negative traces to *all* interaction obligations specified by xalt.

It remains to consider the case where we have xalt in both operands of alt. Consider again the flipping of a coin as given in Figure 2. Another specification where the randomizer simulates the rolling of a three-sided dice is given by:

$$\begin{aligned} \text{Dice} = {} & (1 \text{ alt } (\text{refuse } (2 \text{ alt } 3))) \text{ xalt } (2 \text{ alt } (\text{refuse } (1 \text{ alt } 3))) \text{ xalt} \\ & (3 \text{ alt } (\text{refuse } (1 \text{ alt } 2))) \end{aligned}$$

In Figure 3 the specifications Coin and Dice are merged by the alt operator. Observe that Coin/Dice is a refinement of both the Coin and the Dice specifications. Each interaction obligation defined by Coin has three refined obligations in Coin/Dice (one would have been sufficient), as the earlier inconclusive traces related to Dice have been supplemented as positive or negative. Similarly, each of the three interaction obligations defined by Dice is refined by two interaction obligations in Coin/Dice. In this sense we may say that the specification of Coin/Dice represents both the Coin and the Dice specifications.

On the other hand, neither Coin nor Dice are valid refinements of Coin/Dice, since the traces $1, 2, 3$ are inconclusive in the interaction obligations of Coin and the traces h and t are inconclusive in the interaction obligations of Dice. However, the specifications (Coin alt (refuse Dice)) and ((refuse Coin) alt Dice) are

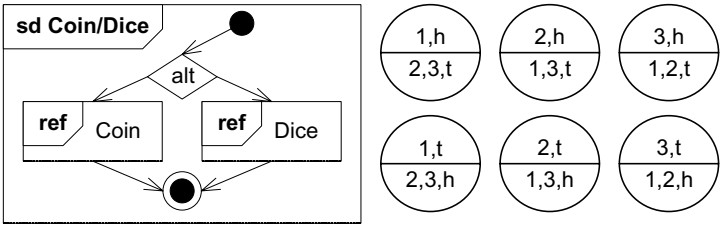

Fig. 3. The Coin and Dice specifications combined by alt. Semantic representation to the right.

both valid refinements of Coin/Dice, since these specifications ensure that none of the traces from the Coin/Dice specification are inconclusive. Intuitively, these specifications mean that traces from the Dice (or Coin) alternative should not be produced, which means that the opposite alternative is chosen. In general, for any specifications d_1 and d_2 the set of valid refinements (and therefore implementations) of d_1 alt d_2 will be both a subset of the valid refinements of d_1 and a subset of the valid refinements of d_2.

A valid refinement of the specification in Figure 3 would be to move the trace h to the negative sets in all interaction obligations, without doing the same with the trace t. The possible outcomes of a single run would then be 1, 2, 3 or t – so we know that if a coin trace is produced, it will be t (assuming 1, 2, 3, h and t are the only relevant traces).

The alt operator should be interpreted as underspecification w.r.t. traces and not w.r.t. interaction obligations. As demonstrated by the examples in this section it is not sufficient for an implementation to consider only one of the alt operands. In general, the alt characterizes the intersection of its operands, meaning that d_1 alt d_2 is a refinement of both d_1 and d_2. If we restrict refinement to narrowing, using alt between two specifications with xalt may be interpreted as 'the implementation must include the inherent nondeterminism specified by at least one of the alternatives'.

3.6 Properties of xalt and Refinement

As for alt, xalt is

- associative: d_1 xalt $(d_2$ xalt $d_3) = (d_1$ xalt $d_2)$ xalt d_3
- commutative: d_1 xalt $d_2 = d_2$ xalt d_1

This follows trivially from the associativity and commutativity of $\cup$.

With respect to xalt, alt is

- right distributive: $(d_1$ xalt $d_2)$ alt $d_3 = (d_1$ alt $d_3)$ xalt $(d_2$ alt $d_3)$
- left distributive: d_1 alt $(d_2$ xalt $d_3) = (d_1$ alt $d_2)$ xalt $(d_1$ alt $d_3)$

meaning that a specification with arbitrary nesting of alt and xalt may always be rewritten as a specification with xalt as the top-level operator. This is proved in [RRS06].

As in the simple case, we have that the refinement operator $\rightsquigarrow$ is:

- reflexive: $d \rightsquigarrow d$
- transitive: $d \rightsquigarrow d' \wedge d' \rightsquigarrow d'' \;\Rightarrow\; d \rightsquigarrow d''$
- monotonic with respect to alt and xalt:
  $d_1 \rightsquigarrow d_1' \wedge d_2 \rightsquigarrow d_2' \;\Rightarrow\; d_1 \text{ alt } d_2 \rightsquigarrow d_1' \text{ alt } d_2' \;\wedge\; d_1 \text{ xalt } d_2 \rightsquigarrow d_1' \text{ xalt } d_2'$

These results are proved in [HHRS05a].

4 Probability

4.1 Motivation

Being able to specify probabilities add useful expressiveness to the specifications. One typical example is in the specification of a coin or a dice, where the alternatives must occur with the same probability. Another example is a gambling machine, where the winning alternatives should occur, but less often than the losing ones.

Interactions are mainly used for specifying communication scenarios. Probabilities are equally relevant in this setting, for instance to specify the probability that a message will never be received when sent over an unreliable communication channel. Another example is when specifying soft real-time constraints such as 'the user of the system will receive an answer within 10 seconds at least 90% of the time' (for more details, see [RHS05a]). As this example demonstrates, we are not only interested in assigning exact probabilities to all alternatives specified by an xalt, but also to specify a possible range for the probabilities, i.e. to allow underspecification with respect to probabilities as well as behaviors. In STAIRS, this is achieved by generalizing xalt to the palt operator.

4.2 Semantic Representation

Semantically, a probabilistic STAIRS specification is represented by a set of *probability obligations* (also called *p-obligations*). A p-obligation $((p, n), Q)$ consists of an interaction obligation (p, n) and a set of probabilities Q. In any valid implementation the p-obligation $((p, n), Q)$ should be selected with a probability in Q. The fact that Q is a set and not a single probability allows us to represent underspecification w.r.t. probabilities.

If a specification includes the p-obligation $((\{t_1, t_2\}, \mathcal{H} \setminus \{t_1, t_2\}), \{0.6\})$, this does not necessarily mean that the probability of getting either t_1 or t_2 is 0.6; it may be greater if there is another p-obligation $((p, n), Q)$ such that $\{t_1, t_2\} \not\subseteq n$. On the other hand, if a specification contains a p-obligation $((p, n), \{0.6\})$ such that $\{t_3, t_4\} \subseteq n$, then we know that the probability of getting a trace in $\{t_3, t_4\}$ is at most 0.4.

The palt construct expresses probabilistic choice. Use of the palt operator is the only way to assign probabilities different from 1. Before defining the semantics of the palt, we introduce the notion of probability decoration, used to specify the probabilities associated with the operands of a palt. It is defined by

$$[\![\ d;Q'\ ]\!] \stackrel{\text{def}}{=} \{(o, Q * Q') \mid (o, Q) \in [\![\ d\ ]\!]\} \tag{8}$$

where multiplication of probability sets is defined by

$$Q_1 * Q_2 \stackrel{\text{def}}{=} \{q_1 * q_2 \mid q_1 \in Q_1 \wedge q_2 \in Q_2\} \tag{9}$$

We also define the summation of n probability sets:

$$\sum_{i=1}^{n} Q_i \stackrel{\text{def}}{=} \{\mathsf{min}(\sum_{i=1}^{n} q_i, 1) \mid \forall\, i \leq n : q_i \in Q_i\} \tag{10}$$

The palt operator describes the probabilistic choice between two or more alternative operands whose joint probability should add up to one. Formally, the palt is defined by

$$[\![\ \mathsf{palt}(d_1;Q_1,\ldots,d_n;Q_n)\ ]\!] \stackrel{\text{def}}{=} \tag{11}$$

$$\{(\oplus \bigcup_{i\in N} \{po_i\}, \sum_{i\in N} \pi_2.po_i) \mid$$

$$N \subseteq \{1,\ldots,n\} \wedge N \neq \varnothing \wedge \forall\, i \in N : po_i \in [\![\ d_i;Q_i\ ]\!]\} \tag{a}$$

$$\cup\ \{(\oplus \bigcup_{i=1}^{n} [\![\ d_i;Q_i\ ]\!], \{1\} \cap \sum_{i=1}^{n} Q_i)\} \tag{b}$$

where $\pi_2.po$ returns the probability set of the p-obligation po and $\oplus$ is an operator for combining the interaction obligations of a set S of p-obligations into a single interaction obligation, defined as

$$\oplus S \stackrel{\text{def}}{=} ((\bigcup_{((p,n),Q)\in S} p) \cap (\bigcap_{((p,n),Q)\in S} p \cup n), \bigcap_{((p,n),Q)\in S} n) \tag{12}$$

We now explain definition 11 in detail. We first look at 11a. If we restricted each N to be a singleton set then this part of the definition could be written equivalently as $\bigcup_{i=1}^{n} [\![\ d_i;Q_i\ ]\!]$. This would correspond to the definition of xalt and means simply that each probabilistic alternative should be reflected in a valid implementation.

By including also the cases where N is any non-empty subset of $\{1,\ldots,n\}$ we are able to define the semantics as a set of p-obligations instead of as a multiset. The operator $\oplus$ characterizes the traces allowed by all the p-obligations in its argument set: A trace t is positive if it is positive according to at least one p-obligation and not inconclusive according to any; t is negative only if it is negative according to all p-obligations; traces that are inconclusive according to at least one p-obligation remain inconclusive. So if a p-obligation $((p, n), Q)$ occurs for example in two operands of the palt, then the resulting semantics will contain a p-obligation $((p, n), Q + Q)$.

The single p-obligation in 11b requires the probabilities of the operands to add up to one. If it is impossible to choose one probability from each Q_i so that

the sum is 1, then the probability set will be empty and the specification is not implementable.

We also redefine the refuse and alt operators to take probabilities into account. Redefining positive traces as negative does not influence probabilities, so refuse is defined simply by

$$[\![\text{ refuse } d]\!] \stackrel{\text{def}}{=} \{((\varnothing, p \cup n), Q) \mid ((p, n), Q) \in [\![d]\!]\} \tag{13}$$

The alt construct captures underspecification with respect to traces. Two sets of p-obligations are combined by taking the pairwise combination of p-obligations from each set. As before, interaction obligations are combined by taking the union of the positive traces and the union of the negative traces. In Section 3.5 we showed that the resulting interaction obligation is a refinement of both the original ones, and therefore represents both of these interaction obligations. Since the two p-obligations from the different operands are chosen independently from each other, probabilities are multiplied. Formally:

$$[\![d_1 \text{ alt } d_2]\!] \stackrel{\text{def}}{=} \{(o_1 \uplus o_2, Q_1 * Q_2) \mid (o_1, Q_1) \in [\![d_1]\!] \wedge (o_2, Q_2) \in [\![d_2]\!]\} \tag{14}$$

4.3 Refinement Revisited

A p-obligation is refined by either refining its interaction obligation, or by reducing its set of probabilities. Formally, a p-obligation $((p', n'), Q')$ is a refinement of a p-obligation $((p, n), Q)$, written $((p, n), Q) \rightsquigarrow ((p', n'), Q')$, iff

$$(p, n) \rightsquigarrow (p', n') \wedge Q' \subseteq Q \tag{15}$$

All abstract p-obligations must be represented by a p-obligation also at the refined level, unless it has 0 as an acceptable probability, which means that it does not need to be implemented. Formally, a specification d' is a refinement of a specification d, written $d \rightsquigarrow d'$, iff

$$\forall\, po \in [\![d]\!] : (0 \notin \pi_2.po \Rightarrow \exists\, po' \in [\![d']\!] : po \rightsquigarrow po') \tag{16}$$

We now explain further why also the cases where N is any non-singular subset of $\{1, \ldots, n\}$ is included in definition 11a. Firstly, we want to avoid a situation where two p-obligations (o_1, Q_1) and (o_2, Q_2) coming from different operands of a palt are represented *only* by a single p-obligation (o, Q) that is a refinement of both (o_1, Q_1) and (o_2, Q_2) at the concrete level. We avoid this since also the p-obligation $(\oplus\{(o_1, Q_1), (o_2, Q_2)\}, Q_1 + Q_2)$ is included in the semantics and hence needs to be represented at the concrete level.

Secondly, it should be possible to let a single p-obligation at the abstract level be represented by a combination of p-obligations at the concrete level, as long as each of these p-obligations are valid refinements of the original p-obligation w.r.t. interaction obligations and their probability sets add up to a subset of the original probability set. The inclusion of the combined p-obligations (resulting

from N sets with more than one element) in the palt semantics makes this possible.

Our definition of refinement also explains why we have chosen to assign sets of acceptable probabilities to the operands, and not simply lower bounds. Consider the following specifications:

$$d_a = \mathsf{palt}(d_1;[\tfrac{1}{5}\dots 1], d_2;[\tfrac{1}{5}\dots 1], d_3;[\tfrac{1}{5}\dots 1])$$
$$d_b = \mathsf{palt}(d_1;[\tfrac{1}{5}\dots \tfrac{1}{2}], d_2;[\tfrac{1}{5}\dots \tfrac{1}{2}], d_3;[\tfrac{1}{5}\dots \tfrac{1}{2}])$$
$$d_c = \mathsf{palt}(d_1;\{\tfrac{1}{5}\}, d_2;\{\tfrac{1}{5}\}, d_3;\{\tfrac{3}{5}\})$$

Then d_c is a refinement of d_a, but not of d_b. So by using only lower bounds we would have less expressive power.

4.4 Simple Example

We now demonstrate a simple refinement in probabilistic STAIRS, building on the DiceCoin/DiceCoin2 example from Section 3.4. Let pDiceCoin be a probabilistic version of DiceCoin where the probabilities of odd and even numbers are the same, represented syntactically and semantically by

$$\begin{aligned} \text{pDiceCoin} &= \mathsf{palt}(\text{Throw135};\{\tfrac{1}{2}\}, \text{Throw246};\{\tfrac{1}{2}\}) \\ [\![\ \text{pDiceCoin}\ ]\!] &= \{\ ((\{1,3,5\},\{2,4,6\}),\{\tfrac{1}{2}\}),((\{2,4,6\},\{1,3,5\}),\{\tfrac{1}{2}\}), \\ &\qquad ((\{1,2,3,4,5,6\},\varnothing),\{1\})\ \} \end{aligned}$$

The semantic representation tells us that the dice should be able to produce at least one number in {1,3,5}, and the probability for this alternative should be $\frac{1}{2}$. Similarly, the dice should be able to produce at least one number in {2,4,6}, with probability $\frac{1}{2}$. Obviously, the probability of producing a number in {1,2,3,4,5,6} should then be 1.

Suppose now that we require that the dice should be fair w.r.t. the odd numbers, give equal chances of odd and even number, and not produce any even number different from 6. We first let $[\![\ \text{Throw1}\ ]\!] = \{\ ((\{1\},\{2,3,4,5,6\}),\{1\})\ \}$ and similarly for the other numbers. We then refine Throw135 by Throw135Fairly:

$$\text{Throw135Fairly} = \mathsf{palt}(\text{Throw1};\{\tfrac{1}{3}\}, \text{Throw3};\{\tfrac{1}{3}\}, \text{Throw5};\{\tfrac{1}{3}\})$$
$$\begin{aligned} &[\![\ \text{Throw135Fairly}\ ]\!] = \{\ ((\{1\},\{2,3,4,5,6\}),\{\tfrac{1}{3}\})\ , \\ &((\{3\},\{1,2,4,5,6\}),\{\tfrac{1}{3}\})\ ,((\{5\},\{1,2,3,4,6\}),\{\tfrac{1}{3}\})\ , \\ &((\{1,3\},\{2,4,5,6\}),\{\tfrac{2}{3}\})\ ,((\{1,5\},\{2,3,4,6\}),\{\tfrac{2}{3}\})\ , \\ &((\{3,5\},\{1,2,4,6\}),\{\tfrac{2}{3}\})\ ,((\{1,3,5\},\{2,4,6\}),\{1\})\ \} \end{aligned}$$

As Throw135 has the semantics $\{((\{1,3,5\},\{2,4,6\}),\{1\})\}$, we see that this is indeed a valid refinement, since the only p-obligation in $[\![\ \text{Throw135}\ ]\!]$ is identical to one of the p-obligations in $[\![\ \text{Throw135Fairly}\ ]\!]$. A dice that is fair w.r.t. the odd numbers, has equal chances of odd and even numbers, and does not produce any even number different from 6 can now be expressed by

$$\text{pDiceCoin2} = \mathsf{palt}(\text{Throw135Fairly};\{\tfrac{1}{2}\}, \text{Throw6};\{\tfrac{1}{2}\})$$
$$\begin{aligned}
[\![\ \text{pDiceCoin2}\ ]\!] = \{\ & (\{\{1\},\{2,3,4,5,6\}\},\{\tfrac{1}{6}\})\ , \\
& (\{\{3\},\{1,2,4,5,6\}\},\{\tfrac{1}{6}\})\ ,\ (\{\{5\},\{1,2,3,4,6\}\},\{\tfrac{1}{6}\})\ , \\
& (\{\{1,3\},\{2,4,5,6\}\},\{\tfrac{1}{3}\})\ ,\ (\{\{1,5\},\{2,3,4,6\}\},\{\tfrac{1}{3}\})\ , \\
& (\{\{3,5\},\{1,2,4,6\}\},\{\tfrac{1}{3}\})\ ,\ (\{\{1,6\},\{2,3,4,5\}\},\{\tfrac{2}{3}\})\ , \\
& (\{\{3,6\},\{1,2,4,5\}\},\{\tfrac{2}{3}\})\ ,\ (\{\{5,6\},\{1,2,3,4\}\},\{\tfrac{2}{3}\})\ , \\
& (\{\{1,3,6\},\{2,4,5\}\},\{\tfrac{5}{6}\})\ ,\ (\{\{1,5,6\},\{2,3,4\}\},\{\tfrac{5}{6}\})\ , \\
& (\{\{3,5,6\},\{1,2,4\}\},\{\tfrac{5}{6}\})\ ,\ (\{\{1,3,5\},\{2,4,6\}\},\{\tfrac{1}{2}\})\ , \\
& (\{\{6\},\{1,2,3,4,5\}\},\{\tfrac{1}{2}\})\ ,\ (\{\{1,3,5,6\},\{2,4\}\},\{1\})\ \}
\end{aligned}$$

Each p-obligation in $[\![$ pDiceCoin $]\!]$ has a refining p-obligation in $[\![$ pDiceCoin2 $]\!]$, so pDiceCoin $\rightsquigarrow$ pDiceCoin2 holds.

4.5 Relating palt to xalt and alt

In STAIRS, every xalt-operand represents an alternative that must be reflected in the implementation. Its probability should be greater than 0, but is otherwise unknown. In probabilistic STAIRS, a specification $\mathsf{xalt}(d_1, \ldots, d_n)$ is therefore interpreted as $\mathsf{palt}(d_1;Q, \ldots, d_n;Q)$ where $Q = \langle 0, \ldots, 1]$.

We now discuss what it means to have probabilistic STAIRS specifications that combine the use of the alt and palt operators. We hope the meaning of underspecification within probabilistic alternative is intuitively clear, and do not go further into this. Instead we show a probabilistic version of the previous examples. pCoin specifies a coin, while pDice specifies a 3-sided dice:

$$\begin{aligned}
\text{pCoin} &= \mathsf{palt}(\text{Heads};\{\tfrac{1}{2}\}, \text{Tails};\{\tfrac{1}{2}\}) \\
\text{pDice} &= \mathsf{palt}(\text{One};\{\tfrac{1}{3}\}, \text{Two};\{\tfrac{1}{3}\}, \text{Three};\{\tfrac{1}{3}\}) \\
[\![\ \text{pCoin}\ ]\!] &= \{(\{\{h\},\{t\}\},\{\tfrac{1}{2}\}),(\{\{t\},\{h\}\},\{\tfrac{1}{2}\}),(\{\{h,t\},\varnothing\},\{1\})\} \\
[\![\ \text{pDice}\ ]\!] &= \{(\{\{1\},\{2,3\}\},\{\tfrac{1}{3}\}),(\{\{2\},\{1,3\}\},\{\tfrac{1}{3}\}),(\{\{3\},\{1,2\}\},\{\tfrac{1}{3}\}), \\
&\quad (\{\{1,2\},\{3\}\},\{\tfrac{2}{3}\}),(\{\{1,3\},\{2\}\},\{\tfrac{2}{3}\}),(\{\{2,3\},\{1\}\},\{\tfrac{2}{3}\}), \\
&\quad (\{\{1,2,3\},\varnothing\},\{1\})\}
\end{aligned}$$

These examples use only a single probability in each probability set (there is no underspecification w.r.t. probabilities). Figure 4 shows the semantics of

$$\text{pCoin/Dice} = \text{pCoin}\ \mathsf{alt}\ \text{pDice}$$

We see that the interaction obligation of each p-obligation in pCoin/Dice refines the interaction obligation of a p-obligation for both pCoin and pDice. For example, the interaction obligation of the leftmost, uppermost p-obligation in Figure 4 represent the first p-obligation of both $[\![$ pCoin $]\!]$ and $[\![$ pDice $]\!]$. Since these represent two independent probabilistic choices it is reasonable to multiply their probabilities. This also gives the nice result that if we consider

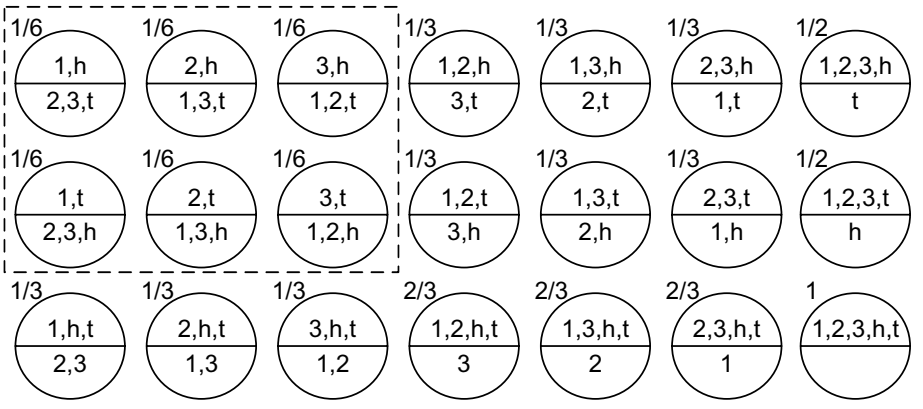

Fig. 4. The semantics of (pCoin alt pDice) in probabilistic STAIRS

only 'pure' p-obligations (those we get from definition 11a in the cases where N is a singleton set), then their probabilities add up to 1. In Figure 4 these 'pure' p-obligations are enclosed by the dotted line.

4.6 Properties of alt, palt and Refinement

For alt, the revised definition 14 is still associative and commutative.

In contrast to xalt, palt is *not* associative. The order in which obligations are combined according to 11b is significant, since this determines which probabilities must add up to 1. Remember that the requirement that probabilities for the operands add up to 1 applies to each occurrence of a palt operator, independently of the nesting level. For similar reasons, alt is not distributive with respect to palt. Consider the following specifications:

$$d_a = (\mathsf{palt}(d_1;Q_1, d_2;Q_2))\ \mathsf{alt}\ (\mathsf{palt}(d_3;Q_3, d_4;Q_4))$$
$$d_b = \mathsf{palt}((\mathsf{palt}(d_1;Q_1, d_2;Q_2)\ \mathsf{alt}\ d_3);Q_3, (\mathsf{palt}(d_1;Q_1, d_2;Q_2)\ \mathsf{alt}\ d_4);Q_4)$$

In d_b we are free to choose different probabilities from the sets Q_1 and Q_2 in the two operands of the outermost palt. In d_a there is no such freedom, so in this respect d_a is more restrictive than d_b.

However, we do have commutativity of palt:

$$\forall\, i, j \in [1, n] : \mathsf{palt}(\ldots, d_i;Q_i, \ldots, d_j;Q_j, \ldots) = \mathsf{palt}(\ldots, d_j;Q_j, \ldots, d_i;Q_i, \ldots)$$

This follows trivially from the commutativity of $\cup$.

For probabilistic STAIRS, the refinement operator $\rightsquigarrow$ is:

- reflexive, transitive, and monotonic with respect to alt
- restricted monotonic with respect to palt:
  $(\forall\, i \in [1:n] : d_i \rightsquigarrow d_i' \wedge Q_i' \subseteq Q_i \wedge \oplus[\![\ d_i\ ]\!] \rightsquigarrow \oplus[\![\ d_i'\ ]\!]) \Rightarrow$
  $\mathsf{palt}(d_1;Q_1, \ldots, d_n;Q_n) \rightsquigarrow \mathsf{palt}(d_1';Q_1', \ldots, d_n';Q_n')$

This is proved in [RHS05b], which also motivates the last requirement in the monotonicity for palt.

The interpretation given for xalt in probabilistic STAIRS is reasonable, as $\mathsf{xalt}(d_1, \ldots, d_n)$ and $\mathsf{palt}(d_1;\langle 0 \ldots 1], \ldots d_n;\langle 0 \ldots 1])$ are refinements of each other when abstracting away the probabilities. This is proved in [RRS06].

5 Related Work

Most specification languages do not distinguish between underspecification and inherent nondeterminism the way it is done in STAIRS. The most well known dialects of interactions are UML [OMG04] and MSC [ITU99]. Neither of these have two different operators corresponding to alt and xalt. In practice, the alt operator of UML is probably used by different groups to describe both inherent nondeterminism and underspecification.

Live Sequence Charts [DH01] and [HM03] is a dialect of MSC where a (part) of an interaction may be designated as either universal (mandatory) or existential (optional). Explicit criteria in the form of precharts decide when the chart applies; whenever the communication behavior described by the prechart occurs, behavior described by the chart *must* follow (in the case of universal locations) or *may* follow (in the case of existential locations). Universal charts specify all allowed traces. This is therefore not the same as inherently nondeterministic alternatives in STAIRS, since the latter only specifies some of the traces that must be present in an implementation.

CSP [Hoa85] defines two different operators for nondeterministic choice. Their difference, however, is explained in terms of internal versus external choice. This is not the same distinction as the one between underspecification and inherent nondeterminism. As an example, let ? denote an input event, ! denote an output event, and seq be the operator for sequential composition in the STAIRS specification $(?a \ \mathsf{seq} \ (!b \ \mathsf{xalt} \ !c)) \ \mathsf{alt} \ ((?b \ \mathsf{seq} \ !d))$. Here, the environment may choose between the two alt-operands, corresponding to external choice in CSP. However, the choice between !b and !c should be inherently nondeterministic, a requirement that may not be expressed using the CSP operators, while replacing xalt with alt, would correspond to internal choice in CSP.

[SBDB97] extends the process algebraic language LOTOS [ISO89] with a disjunction operator for specifying implementation freedom (i.e. underspecification), leaving the LOTOS choice operator to be used for inherent nondeterminism. The disjunction operator is similar to our alt operator, and the choice operator corresponds to xalt. An important difference between disjunction and alt is that an implementation will have to select exactly one of the disjunction operands, while it may include several of the traces specified by alt.

Probabilistic automata [Seg95] includes both nondeterminism and probabilistic choice. Underspecification with respect to probabilities is represented by nondeterministic choices between distributions. As for automata in general, specifications are complete in the sense that there is no notion of inconclusive behavior.

In [MM99] a probabilistic extension of Dijkstra's Guarded Command Language *GCL* [Dij76] called *pGCL* is presented. The language includes both an

operator $\sqcap$ for 'demonic' choice and an operator ${}_p\oplus$ for probabilistic choice. The following intuitive explanation is given for the meaning of the construct $this \sqcap that$: 'The customer will be happy with either *this* or *that*; and the implementer may choose between them according to his own concerns.' This indicates that the role of the $\sqcap$ operator in a *pGCL* specification is to express underspecification, similar to the role of the alt operator in (probabilistic) STAIRS. By specifying probabilistic choices the role of the ${}_p\oplus$ operator in *pGCL* corresponds to the role of palt in probabilistic STAIRS. There is no notion of inconclusive behavior in *pGCL*.

[Heh04] shows how probabilistic reasoning can be applied to predicative programs and specifications. Nondeterminism is disjunction, and equivalent to a deterministic choice in which the determining expression is a variable of unknown value (probability). Nondeterminism gives freedom to the implementer; it can be refined by a deterministic or a probabilistic choice. Since the implementer is not forced to produce both alternatives, the nondeterminism in [Heh04] corresponds to underspecification in STAIRS. Cases where both alternatives need to be possible are expressed by a probabilistic choice, as in probabilistic STAIRS.

6 Conclusion

This article has shown the need for underspecification, inherent nondeterminism and probability in specifications. We have demonstrated that these phenomena are adequately expressed in STAIRS and probabilistic STAIRS by the operators alt, xalt and palt. New insight has been gained into the interplay between these operators through studies of simple examples. The focus of this paper has been on the theoretical understanding of how underspecification and inherent nondeterminism is expressed in specifications and represented semantically. The simplicity of the specifications has allowed us to properly explain their semantic representations. For more examples related to communication see [HHRS05b], [RHS05c] and [RHS05a]. We firmly believe that STAIRS and probabilistic STAIRS offer a suitable expressiveness for practical specifications, and intend to show this in the future through studies of real-life specifications.

The research on which this paper reports has been partly carried out within the context of the IKT-2010 project SARDAS (15295/431) and the IKT SOS project ENFORCE (164382/V30), both funded by the Research Council of Norway. We thank Roberto Segala and the other members of the SARDAS project for useful discussions related to this work. We also thank the anonymous reviewers for constructive feedback.

References

[DH01] W. Damm and D. Harel. LSCs: Breathing life into message sequence charts. *Formal Methods in System Design*, 19(1):45–80, 2001.

[Dij76] E. W. Dijkstra. *A Discipline of Programming.* Prentice-Hall, 1976.

[Heh04] E. C. R. Hehner. Probabilistic predicative programming. In Dexter Kozen and Carron Shankland, editors, *Mathematics of Program Construction, 7th International Conference*, number 3125 in Lecture Notes in Computer Science, pages 169–185. Springer, 2004.

[HHRS05a] Ø. Haugen, K. E. Husa, R. K. Runde, and K. Stølen. Why timed sequence diagrams require three-event semantics. Technical Report 309, Department of Informatics, University of Oslo, 2005.

[HHRS05b] Ø. Haugen, K.E. Husa, R.K. Runde, and K. Stølen. STAIRS towards formal design with sequence diagrams. *Software and System Modeling*, 4(4):349–458, 2005.

[HM03] D. Harel and R. Marelly. *Come, Let's Play: Scenario-Based Programming Using LSC's and the Play-Engine*. Springer, 2003.

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

[ISO89] International Standards Organization. *Information Processing Systems – Open Systems Interconnection - Lotos – a Formal Description Technique Based on the Temporal Ordering of Observational Behaviour – ISO 8807*, 1989.

[ITU99] International Telecommunication Union. *Recommendation Z.120 — Message Sequence Chart (MSC)*, 1999.

[MM99] C. Morgan and A. McIver. pGCL: Formal reasoning for random algorithms. *South African Computer Journal*, 22:14–27, 1999.

[OMG04] Object Management Group. *UML 2.0 Superstructure Specification*, ptc/04-10-02 edition, 2004.

[RHS05a] A. Refsdal, K. E. Husa, and K. Stølen. Specification and refinement of soft real-time requirements using sequence diagrams. In P. Pettersson and W. Yi, editors, *Proc. Formal Modeling and Analysis of Timed Systems: Third International Conference, FORMATS, 2005*, number 3829 in Lecture Notes in Computer Science, pages 32–48. Springer, 2005.

[RHS05b] A. Refsdal, K. E. Husa, and K. Stølen. Specification and refinement of soft real-time requirements using sequence diagrams. Technical Report 323, Department of Informatics, University of Oslo, 2005.

[RHS05c] R.K. Runde, Ø. Haugen, and K. Stølen. Refining UML interactions with underspecification and nondeterminism. *Nordic Journal of Computing*, 12(2):157–188, 2005.

[Ros95] A. W. Roscoe. CSP and determinism in security modelling. In *Proc. IEEE Symposium on Security and Privacy*, pages 114–127. IEEE Press, 1995.

[RRS06] A. Refsdal, R. K. Runde, and K. Stølen. Underspecification, inherent nondeterminism and probability in sequence diagrams. Technical Report 335, Department of Informatics, University of Oslo, 2006.

[SBDB97] M.W.A. Steen, H. Bowman, J. Derrick, and E.A. Boiten. Disjunction of LOTOS specifications. In T. Mizuno, N. Shiratori, T. Higashino, and A. Togashi, editors, *Formal Description Techniques and Protocol Specification, Testing and Verification: FORTE X / PSTV XVII '97*, pages 177–192. Chapman & Hall, 1997.

[Seg95] R. Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems*. PhD thesis, Massachusetts Institute of Technology, 1995.