Assessing the Expressivity of Formal Specification Languages *

Natalia López, Manuel Núñez and Ismael Rodríguez

Dept. Sistemas Informáticos y Programación
Universidad Complutense de Madrid, 28040 Madrid, Spain
e-mail:{natalia,mn,isrodrig}@sip.ucm.es

Abstract. Formal modelling languages are powerful tools to systematically represent and analyze the properties of systems. A myriad of new modelling languages, as well as extensions of existing ones, are proposed every year. We may consider that a modelling language is useful if it allows to represent the *critical* aspects of systems in an expressive way. In particular, we require that the modelling language allows to accurately discriminate between *correct* and *incorrect* behaviors concerning critical aspects of the model. In this paper we present a method to assess the suitability of a modelling language to define systems belonging to a specific domain. Basically, given a system, we consider alternative correct/incorrect systems and we study whether the representations provided by the studied modelling language keep the distinction between correct and incorrect as each alternative system does.

1 Introduction

Hundreds of languages have been proposed to formally describe any kind of systems. A lot of them differ only in some aspect concerning the way some features are represented or interpreted (e.g. timed automata [AD94] versus temporal process algebras [NS94] versus timed Petri nets [Zub80], generative versus reactive probabilistic systems [GSS95], Markovian [Hil96,BG98] versus non-markovian stochastic models [LN01,BG02], and so on). Since the number of possible ways to deal with each feature is high, each lineal combination of these possibilities eventually leads to a new language. Thus, a clear and well-defined criterion to asses the utility of a language to model systems belonging to some domain would be very useful. In this line, we could ask ourselves which characteristics we want in a given formal method [AR00]. Informally speaking, a language is good to model a class of systems if it allows to create models where critical features are suitably represented. In terms of correction, a model should be able to perform what the original system does, and should not do what the system does not. For instance, if a system must perform the action a only if the variable x is equal to 10 then a model specifying only that *a may be performed* would not be accurate

* Research partially supported by the Spanish MCYT project TIC2003-07848-C02-01 and the Junta de Castilla-La Mancha project PAC-03-001.

enough. Besides, if that action a must be performed only after 5 seconds then a model where that requirement is not included would not be suitable. Following these ideas, a modelling language is suitable to define a class of systems if it discriminates desirable and undesirable behavior (almost) as the corresponding modelled systems do. In order to check it, we will (semi-)automatically compare the behavior of systems and their models. In particular, we will compare the correct and incorrect behaviors each of them may expose.

There is a *testing technique* that can inspire the creation of a *new* methodology that actually fits into our purposes. *Mutation Testing* [Ham77,How82,BM99] allows to estimate the power of a test suite to assess the (in-)correctness of an implementation with respect to a specification. Basically, mutation testing consists in facing tests with several *mutants* of the specification, that is, alternative specifications where some aspect is modified. Mutants are created from the specification by introducing modifications that simulate typical programming errors. Then, the set of tests to be assessed is applied to each mutant and we observe the capacity of tests to *kill* mutants, that is, to detect erroneous behaviors in mutant specifications that are actually wrong. Let us note that a mutant could be *correct*, that is, equivalent to the original specification. Unfortunately, to check the correctness of a mutant is not decidable. So, this technique is, in general, semi-automatic. Our method, inspired in the previous idea, can be basically described as follows: Given a real system (defined in some language) we create some mutants (in the same language) that might behave incorrectly. Then, we apply the *modelling language under assessment* to create models of both the original system and their mutants. If the modelling language were not expressive enough, then several systems with different behaviors (taken from the mutant systems and/or the original system) could converge to a single model. If these systems were either all correct or all incorrect then it would not be a problem that the modelling language provides a single model for them. This is so because the conversion might not have lost relevant aspects. However, if some of the systems that converged to a single model were correct and others were not then the modelling language is losing relevant characteristics that delimit the difference between correct and incorrect. Let us note that, in general, no modelling language allows to express in a *natural* way exactly the same things than the language used for the system definition.

Our measure of the expressivity and suitability of a language to define a given system will be based on the relation between the correctness of the mutants and the correctness of their corresponding models. For example, let us consider 1000 mutants of a system. Let us suppose that 900 mutants are incorrect with respect to the system and let us consider the 900 models of these mutants. If 300 of them are also models of the original system then the modelling language is not very suitable for defining this class of systems because the modelling phase loses aspects that are critical to define the *border* between correct and incorrect. Other possible unsuitability criterion is the following: If 300 of these models of incorrect mutants are *equivalent* to the original system or its model (or *more/less* restrictive in a sense that can be considered valid) then the modelling language

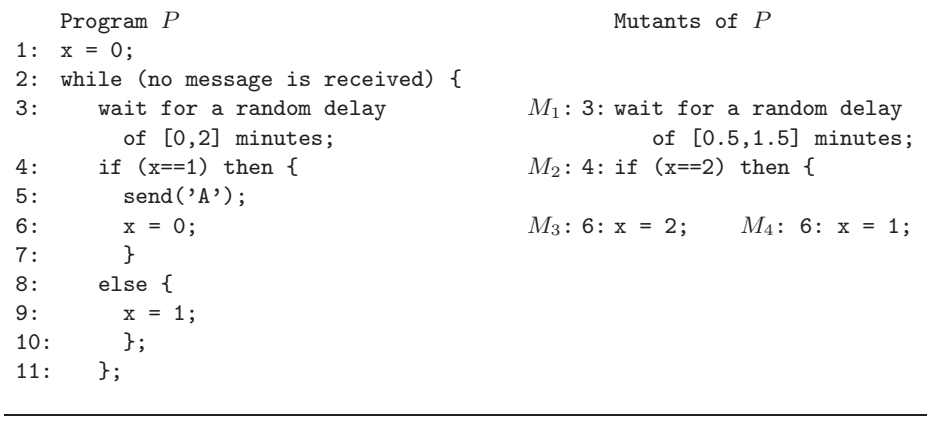

Fig. 1. Program example

is unsuitable. Since our methodology can be applied to other systems of the same domain to measure the suitability of the target language to describe them, it may help to check whether a new language makes a relevant contribution to express the critical aspects of systems belonging to a specific domain.

The rest of the paper is structured as follows. In the next section we introduce a simple example to motivate the definition of our methodology. In Section 3 we formally present the main concepts of our methodology. Next, in Section 4 we study some formal properties relating the concepts previously introduced. Finally, in Section 5 we present our conclusions and some lines for future work.

2 Motivating Example

In this section we illustrate our method with a running example. A program P, written in a given language L, is depicted in Figure 1. Until a message is received, it iteratively performs two operations: First, it waits for a delay between 0 and 2 minutes (we assume that the random variable denoting the delay is *uniformly* distributed); next, it sends the message `A` one out of two times. Let us note that, from an external observer's point of view, the behavior of P actually consists in iteratively sending the message `A` after random delays between 0 and 4 minutes, until a message is received from the environment.

In Figure 1 we also give four *mutants* of P. Each of them differs from P in a single code line. We will apply the following *correctness criterion*: A mutant M_i is correct with respect to P if its external behavior coincides with the one from P. We suppose that *external behavior* concerns only sent messages and delays between them. Since the maximal delay between two consecutive `A` messages in M_1 is 3, M_1 cannot produce some behaviors included in P. So, M_1 is incorrect. M_2 is incorrect as well: For instance, it can wait 10 minutes until a message is

received without sending any message. M_3 is correct because its behavior is not affected by the change. Finally, M_4 is incorrect: *All* the times the loop is taken, except the first one, we have that `X` is 1, so an `A` message is sent. Hence, the delay between messages `A` is equal to, at most, 2 minutes, while delays can take 4 minutes in P.

Let us consider three modelling languages L_1, L_2, and L_3. Each of them misses some aspect that is actually considered in the language L: Only *fix* temporal delays can be represented in L_1, L_2 does not use any *variables* to govern the behavior of systems, and L_3 cannot represent any *temporal delay* at all. We will study and compare the suitability of these languages to model the program P. In particular, we will study whether each of the languages properly captures the (subtle) aspects delimiting the border between correct and incorrect behaviors. In order to do it, each language will face the definition of each mutant of P, and we will study whether the (in-)correctness of each alternative properly remains in the models domain provided by each language.

The modelling language L_1 represents most program aspects exactly in the same way as L does. The only difference is that L_1 does not allow to represent *random* temporal delays. Instead, all temporal delays must denote a *deterministic* amount of time. In particular, the translation of a program from L to L_1 is done as follows: For any random delay we consider its *mean* expected delay. For example, line 3 in P is translated into "`wait for 1 minute`." As a result, the models of P and M_1 in language L_1 actually *coincide*. Since P is correct but M_1 is not, this collision denotes that L_1 does not properly represent this behavior. The collision of two different systems, being one of them correct and the other incorrect, into a single *model*, is called *collision mistake* in our framework (the formal definition is given in Section 3). Basically, it denotes that some critical details are lost in translation.

Let us consider the correctness criterion we use in the domain of L and let us apply it to the domain of L_1 models. Despite M_1 is an incorrect mutant, its model in L_1 produces exactly the same external behavior as the model of P. In fact, let us note that the *fix* delay placed before the `if` statement is 1 minute long in *both* models. That is, the model of M_1 is *correct* with respect to the model of P, but the system it comes from is not correct with respect to P. We denote the situation where an incorrect system leads to a correct model, or viceversa, by *model mistake* (see Section 3). Basically, it shows that the correctness criterion is not properly preserved in the model domain.

Regarding the modifications induced by M_2, M_3, and M_4, they are properly represented in L_1. This is because L_1 represents all lines but line 3 exactly as L does. In particular, these mutants do not produce any of the mistakes considered before. On the one hand, their models differ from each other and from the model of P, so there do not lead to any collision mistake. On the other hand, models of incorrect mutants are also incorrect in the domain of models, so they do not produce model mistakes. In particular, the model of M_2 does not produce any message, and the (fix) delay between messages in the model of M_4 is 1 minute. However, the fix delay in the model of P is 2 minutes. Finally, since both M_3

and its model are correct, M_3 does not yield a model mistake. Summarizing, in both approaches 3 out of 4 mutants do not produce mistakes and are correctly represented by L_1.

We will conclude the presentation of this example at the end of Section 3, once all the concepts underlying our methodology have been formally introduced. In particular, we will use L_2 and L_3 to create models of P and its mutants, and we will compare the suitability of each language to represent P.

3 Formal Framework

In this section we present the basic concepts of our methodology and show how they are applied to assess the suitability of a modelling language to describe a class of systems. First, we introduce the concept of *language*. In our framework a language is defined by a *syntax*, allowing to construct the appropriate words (i.e., programs), as well as a *semantics*, associating semantic values to syntactic expressions.

Definition 1. A *language* is a pair $\mathcal{L} = (\alpha, \beta)$, where α denotes the *syntax* of $\mathcal{L}$ and β denotes its *semantics*. Let $\texttt{Systems}(\mathcal{L})$ denote the set of all words conforming to α. Then, β is a total function $\beta : \texttt{Systems}(\mathcal{L}) \longrightarrow B$, where B is the *semantic domain* for $\mathcal{L}$. □

Next we present the concept of *correctness* of a system. Correctness is defined in the domain of semantic values and it is given in terms of a comparison between values. So, we may say that a semantic value b_1 is correct *with respect to* b_2. For example, this might mean that both semantic values represent *bisimilar* behaviors. The correctness relation is not necessarily symmetric. For example, it may define a *conformance* relation where the behavior of a system must be a subset of that exposed by another. Besides, it could be defined in terms of the set of semantic values where some required property holds. In general, if c is a *correctness criterion* then $b \in c(b')$ means that b is correct with respect to b'. Let us note that by using this criterion we implicitly define which aspects of a system will be considered *critical*. For example, if this criterion says that a system is correct regardless of its temporal performance, then temporal issues are not considered critical in our analysis. In the following definition, $\mathcal{P}(X)$ denotes the powerset of the set X.

Definition 2. Let $\mathcal{L} = (\alpha, \beta)$ be a language with $\beta : \texttt{Systems}(\mathcal{L}) \longrightarrow B$. A *correctness criterion* for the language $\mathcal{L}$ is a total function $c : B \longrightarrow \mathcal{P}(B)$.

If for all $b \in B$ we have $b \in c(b)$ then c is a *reflexive* criterion. If for all $b, b' \in B$, $b' \in c(b)$ implies $b \in c(b')$, then c is a *symmetric* criterion. If for all $b, b', b'' \in B$, $b' \in c(b)$ and $b'' \in c(b')$ imply $b'' \in c(b)$, then c is a *transitive* criterion. We say that a criterion c is an *equivalence criterion* if it is reflexive, symmetric, and transitive. □

According to our methodology, we will create *mutants* to check whether they behave as their corresponding models. Thus, we will be able to assess whether the

modelling language properly discriminates correct and incorrect behaviors. This will provide us with a measure of its suitability. In order to create mutants we introduce modifications in the original systems. These modifications *substitute* some subexpressions of the system by others. Given a string of symbols and a substitution, this function returns the set of strings resulting from the application of the substitution to the string at any point. In the next definition we formally present this concept. For any set of symbols Σ, we consider that Σ^* denotes the set of all finite strings of symbols belonging to Σ.

Definition 3. Let Σ be a set of symbols. Let $e = e_1 \cdots e_n$ and $e' = e'_1 \cdots e'_m$ be sequences in Σ^*. The *term substitution function* for e and e', denoted by $[e/e']$, is a function $[e/e'] : \Sigma^* \longrightarrow \mathcal{P}(\Sigma^*)$ where for any $w = w_1 \cdots w_p$ we have

$$[e/e'](w) = \left\{ w_1 \cdots w_k e'_1 \cdots e'_m w_{k+n+1} \cdots w_p \,\middle|\, \begin{array}{c} 1 \leq k \leq p \wedge \\ w = w_1 \cdots w_k e_1 \cdots e_n w_{k+n+1} \cdots w_p \end{array} \right\}$$

□

The creation of mutants will be defined in terms of a function. Given a system and a set of term substitution functions, this function generates a set of mutants by applying substitutions of the set to the system.

Definition 4. Let $\mathcal{L} = (\alpha, \beta)$ be a language and $\mathtt{S}$ be a set of term substitution functions. A *mutation function* for $\mathcal{L}$ and $\mathtt{S}$ is a total function $M : \mathtt{Systems}(\mathcal{L}) \longrightarrow \mathcal{P}(\mathtt{Systems}(\mathcal{L}))$, where for any $a \in \mathtt{Systems}(\mathcal{L})$ and $b \in M(a)$ we have $b \in \sigma(a)$ for some $\sigma \in \mathtt{S}$. □

From now on we will assume that for any mutation function M for $\mathcal{L}$ and any $a \in \mathtt{Systems}(\mathcal{L})$ we have $a \in M(a)$. This assumption will help to deal with a system and its mutants in a more compact way.

The *translation* of a system from the original system language into the modelling language under assessment will be also represented by means of a function. Let us note that this function, in general, will not be injective: Some systems with different behaviors could be represented by the same model. Hence, the modelling language may lose some details. If these details are not considered critical in the analysis, then losing them is not necessarily bad. On the contrary, eliminating irrelevant details in models may help to create more handleable models. However, if the translation into the modelling language loses critical details, then the modelling language may not be suitable. As we will see below, these situations will be detected within our framework.

Definition 5. Let $\mathcal{L}_1 = (\alpha_1, \beta_1), \mathcal{L}_2 = (\alpha_2, \beta_2)$ be two languages. A *translation function* from $\mathcal{L}_1$ to $\mathcal{L}_2$ is a total function $T : \mathtt{Systems}(\mathcal{L}_1) \longrightarrow \mathtt{Systems}(\mathcal{L}_2)$. □

Now we are provided with all the needed machinery to present the concepts used in our methodology. We propose two alternative criteria to check whether a modelling language is suitable to represent relevant details of systems. Next we present the first one: If a correct system and an incorrect one (taken from

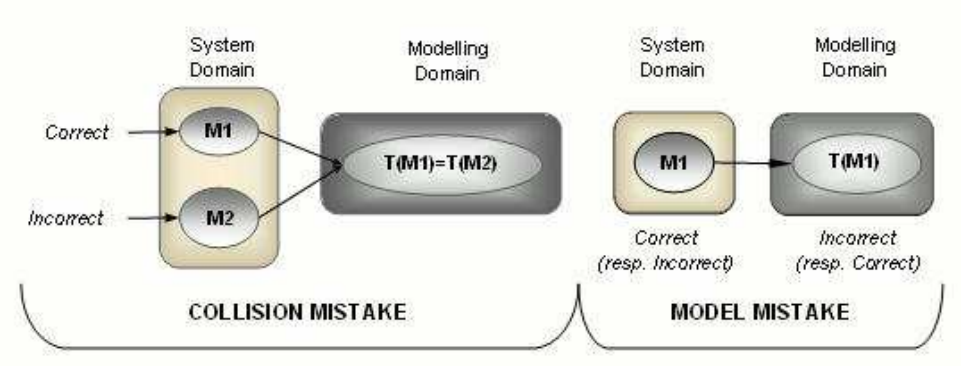

Fig. 2. Correctness Criteria.

the set of mutants and the original system) converge to a single model then relevant details concerning the correctness are lost. In particular, the translation misses details that make the difference between correct and incorrect. Hence, we say that it is a *collision mistake* in the representation of the system by the modelling language. This situation is graphically presented in Figure 2 (left). In the following definition the cardinal of a set X is denoted by $|X|$.

Definition 6. Let $\mathcal{L}_1 = (\alpha_1, \beta_1), \mathcal{L}_2 = (\alpha_2, \beta_2)$ be two languages, $\mathtt{S}$ be a set of term substitution functions, and M be a mutation function for $\mathcal{L}_1$ and $\mathtt{S}$. Let T be a translation function from $\mathcal{L}_1$ to $\mathcal{L}_2$. Let c_1 be a correctness criterion for $\mathcal{L}_1$ and $s \in \mathtt{Systems}(\mathcal{L}_1)$. We say that $m \in M(s)$ is a *collision mistake* for s, M, T, and c_1, denoted by $\mathtt{CMistake}^{s,M,T,c_1}(m)$, if there exists $m' \in M(s)$, with $T(m) = T(m')$, such that either

(a) $\beta_1(m) \in c_1(\beta_1(s))$ and $\beta_1(m') \notin c_1(\beta_1(s))$ or
(b) $\beta_1(m) \notin c_1(\beta_1(s))$ and $\beta_1(m') \in c_1(\beta_1(s))$.

In this case, we say that m' is a *collision pair* of m for s, M, T, and c_1, and we denote it by $m' \updownarrow^{s,M,T,c_1} m$. The *number of collision mistakes* of s for M, T, and c_1, denoted by $\mathtt{NumCM}^{M,T,c_1}(s)$, is defined as $|\{m \mid m \in M(s) \wedge \mathtt{CMistake}^{s,M,T,c_1}(m)\}|$. □

For the sake of notation simplicity, we will omit some parameters when we assume that they are unique in the context. Hence, we will sometimes simply write $\mathtt{CMistake}(m)$, $m' \updownarrow m$, and $\mathtt{NumCM}(s)$. Next we define our second mechanism to check whether a modelling language fails to express a system. Now, a correctness criterion will also be applied to the modelling language. For instance, an equivalence criterion may be used, that is, a reflexive, symmetric, and transitive criterion. We say that the modelling language produces a *model mistake* in the translation of the original system if the correctness criteria of

the original language and the modelling language map (in-)correct systems in a different way. In particular, if a *correct* (resp. *incorrect*) system is translated into a model that is considered *incorrect* (resp. *correct*), then we detect a failure in the representation of the system by the modelling language. This situation is graphically presented in Figure 2 (right).

Definition 7. Let $\mathcal{L}_1 = (\alpha_1, \beta_1), \mathcal{L}_2 = (\alpha_2, \beta_2)$ be two languages, $\mathtt{S}$ be a set of term substitution functions, M be a mutation function for $\mathcal{L}_1$ and $\mathtt{S}$, T be a translation function from $\mathcal{L}_1$ to $\mathcal{L}_2$, c_1 and c_2 be correctness criteria for $\mathcal{L}_1$ and $\mathcal{L}_2$, respectively, and $s \in \mathtt{Systems}(\mathcal{L}_1)$. We say that $m \in M(s)$ is a *model mistake* for s, M, t, c_1, and c_2, denoted by $\mathtt{MMistake}^{s,M,t,c_1,c_2}(m)$, if either

(a) $\beta_1(m) \in c_1(\beta_1(s))$ and $\beta_2(t(m)) \notin c_2(\beta_2(t(s)))$ or
(b) $\beta_1(m) \notin c_1(\beta_1(s))$ and $\beta_2(t(m)) \in c_2(\beta_2(t(s)))$.

The *number of model mistakes* of s for M, t, c_1, and c_2, denoted by $\mathtt{NumMM}^{M,t,c_1,c_2}(s)$, is given by $\mid \{m \mid m \in M(s) \ \wedge \ \mathtt{MMistake}^{s,M,t,c_1,c_2}(m)\} \mid$. □

Again, we will omit some parameters when they can be inferred from the context. Now we present mechanisms to find the mistakes in the representation of a system, regardless of the used criterion. In both cases, we define the *suitability* of a modelling language to represent a system as the ratio of mutants that are *correctly* translated.

Definition 8. Let $\mathcal{L}_1 = (\alpha_1, \beta_1), \mathcal{L}_2 = (\alpha_2, \beta_2)$ be languages, $\mathtt{S}$ be a set of term substitution functions, M be a mutation function for $\mathcal{L}_1$ and $\mathtt{S}$, T be a translation function from $\mathcal{L}_1$ to $\mathcal{L}_2$, c_1 and c_2 be correctness criteria for $\mathcal{L}_1$ and $\mathcal{L}_2$, respectively, and $s \in \mathtt{Systems}(\mathcal{L}_1)$. The *collision suitability* of $\mathcal{L}_2$ to define s under M, T, and c_1, denoted by $\mathtt{CSuit}(s, \mathcal{L}_2, M, T, c_1)$, is defined as

$$\mathtt{CSuit}(s, \mathcal{L}_2, M, T, c_1) = \frac{\mid M(s) \mid \ - \mathtt{NumCM}(s)}{\mid M(s) \mid}$$

The *model suitability* of $\mathcal{L}_2$ to define s under M, T, c_1, and c_2, denoted by $\mathtt{MSuit}(s, \mathcal{L}_2, M, T, c_1, c_2)$, is defined as

$$\mathtt{MSuit}(s, \mathcal{L}_2, M, T, c_1, c_2) = \frac{\mid M(s) \mid \ - \mathtt{NumMM}(s)}{\mid M(s) \mid}$$

□

Let us remark that in order to measure collision suitability no correctness criterion is applied to the modelling language $\mathcal{L}_2$, that is, c_2 is ignored. The previous concepts can be extended to deal with *sets* of systems belonging to some specific domain. Thus, our framework is not only useful to assess the suitability of a language with respect to a specific system but also with respect to a family of systems. For instance, we could assess the suitability of a modelling language to describe data link layer network protocols or distributed cryptographic protocols. A modelling language unable to represent temporal delays (resp. data encryption) would produce several collision and/or model mistakes and would

be unsuitable. In each case, a representative set of systems in the domain has to be chosen and applied. Depending on the relevance of each system and its fitness in the domain, it should have a different weight in the overall assessment. In the following definition we *overload* the functions presented in the previous definition to deal with *sets*.

Definition 9. Let $\mathcal{L}_1 = (\alpha_1, \beta_1)$ and $\mathcal{L}_2 = (\alpha_2, \beta_2)$ be two languages, $\mathtt{S}$ be a set of term substitution functions, and M be a mutation function for $\mathcal{L}_1$ and $\mathtt{S}$. Let T be a translation function from $\mathcal{L}_1$ to $\mathcal{L}_2$ and c_1, c_2 be correctness criteria for $\mathcal{L}_1$ and $\mathcal{L}_2$, respectively. Let $\mathcal{S} = \{(s_1, w_1), \ldots, (s_n, w_n)\} \in \mathcal{P}(\mathtt{Systems}(\mathcal{L}_1) \times (0..1])$ be a set of pairs of systems and *weights* for these systems, where for all $1 \leq i \leq n$ we have $w_i > 0$ and $\sum_{1 \leq i \leq n} w_i = 1$. We define the *collision suitability* of $\mathcal{L}_2$ to define $\mathcal{S}$ under M, T, and c_1 as

$$\mathtt{CSuit}(\mathcal{S}, \mathcal{L}_2, M, T, c_1) = \sum_{1 \leq i \leq n} w_i \cdot \mathtt{CSuit}(s_i, \mathcal{L}_2, M, T, c_1)$$

We define the *model suitability* of $\mathcal{L}_2$ to define $\mathcal{S}$ under M, T, c_1, and c_2 as

$$\mathtt{MSuit}(\mathcal{S}, \mathcal{L}_2, M, T, c_1, c_2) = \sum_{1 \leq i \leq n} w_i \cdot \mathtt{MSuit}(s_i, \mathcal{L}_2, M, T, c_1, c_2)$$

□

3.1 Concluding our motivating example

Let us consider the modelling language L_2. It represents random delays exactly as L does, but it considers neither program variables nor their effects. In particular, it converts any condition on variables into a *non-deterministic* choice. For instance, concerning the conditional sentence in line 4 of the program P presented in Figure 1, the model of P in L_2 just denotes that *either of the branches is non-deterministically chosen.* In fact, since variables are ignored in L_2, the choice just consists in choosing between sending `A` or not.

Let us note that the models of the programs P and M_1 in L_2 do not coincide because their different versions of line 3 stay in L_2. For the same reason, the model of M_1 is different to the model of the *correct* mutant M_3, so there is no collision mistake with M_1. Moreover, there is not a *model* mistake with M_1 either, but the reason is quite different. Let us note that the model of P in L_2 can produce *any* delay between `A` messages. Since the `if` choice is implemented as a non-deterministic choice, the `else` statement can be taken *any* number of times before the next message `A` is sent. Thus, any delay can be cumulated before sending each message. However, the model of M_1 cannot produce its *first* message until 0.5 units of time pass. That is, the model of M_1 is not equivalent to the model of P. Hence, in spite of the collateral effects of the absence of variables, the (wrong) temporal behavior of M_1 is properly detected.

Let us remark that the mutants of M_2, M_3, and M_4 only differ from P in the way they deal with variables. Hence, the models of M_2, M_3, and M_4 in L_2 coincide with the model of P. The collision with the original program is not a collision mistake for M_3, because M_3 is actually correct. However, it is so for

the incorrect mutants M_2 and M_4. Moreover, since these models coincide with the model of the correct mutant M_3, M_3 is also involved in a collision mistake. Regarding the correctness in the model domain, we find the same problem for the incorrect mutants M_2 and M_4: The models of M_2 and M_4 are *correct* with respect to the *model* of P (they have the same external behavior), but the respective systems M_2 and M_4 are incorrect with respect to P. On the other hand, since the model of M_3 is correct, M_3 does not produce any model mistake. So, the number of properly represented mutants with L_2 is 1 and 2, for collision and model mistakes, respectively.

Finally, we consider the modelling language L_3. This language does not represent temporal delays (neither deterministically nor probabilistically), though variables are represented and used exactly as in L. Since the line 3 is ignored by L_3, P and the incorrect mutant M_1 collide by producing the same model in L_3, which in turn is a collision mistake. Besides, since both models coincide, we have that the model of M_1 is equivalent to the model P (though M_1 is not equivalent to P). Hence, we also have a model mistake (as we will study in Section 4, a collision mistake between two models induces a model mistake in *at least* one of the involved systems).

Regarding M_2 and M_4, they lead to models that are different to both the model of P and the model of the correct mutant M_3. That is, M_2 and M_4 do not induce any collision mistake. The correct mutant M_3 does not produce a collision mistake either. Concerning model mistakes, M_2 does not produce them because its model cannot produce any message at all, which is not equivalent to the behavior of the model of P. However, the model of the incorrect M_4 is actually *equivalent* to the model of P. Let us note that, since temporal delays are not concerned by L_3, the external behavior of L_3 only reveals the number of times the message `A` is sent. Both the model of P and the model of M_4 can send `A` any number of times. So, they present the same external behavior and we have a model mistake. Summarizing, if we consider collision mistakes then 3 out of 4 mutants are free of them. However, if model mistakes are concerned then only 2 out of 4 mutants are properly represented.

In the next tables we compare the performance of each language to define the program P and its mutants. The first table shows the mistakes of each mutant as well as the mistakes of P. Obviously, P cannot yield a model mistake (its model is correct by definition). However, if the model of P coincides with the model of an incorrect mutant, then P is actually involved in a collision mistake. The second table, for each kind of mistake, divides the number of mistake-free systems by the total number of systems (both P and the mutants are considered).

	L_1 (*det. time*)		L_2 (*no vars*)		L_3 (*no time*)	
	collision m?	model m?	collision m?	model m?	collision m?	model m?
P	**with** M_1	no	**with** M_2, M_4	no	**with** M_1	no
M_1	**with** P	**yes**	no	no	**with** P	**yes**
M_2	no	no	**with** P, M_3	**yes**	no	no
M_3	no	no	**with** M_2, M_4	no	no	no
M_4	no	no	**with** P, M_3	**yes**	no	**yes**

	collision suitability	model suitability
L_1	0.6	0.8
L_2	0.2	0.6
L_3	0.6	0.6

If our micro-sample of *four* mutants were representative of all the kinds of errors that may appear in the system P, then we could conclude that the modelling language that provides the best representation accurateness is L_1. Similarly, we can also study sets of systems. If several systems within a given domain, instead of a single system, are considered then the suitability of a modelling language to specify some semantical aspects of these systems can be assessed. Let us note that, in the previous example, we could consider other correctness criterions (e.g., traces inclusion, bisimulation, conformance, etc). Since each criterion focuses on different semantical aspects of systems, each would lead to different suitability results. In the previous example we have considered a straightforward translation from L to each modelling language. Other more elaborated and precise translations could be considered. For example, temporal delays of *different* length could be simulated in L_1 by iterating a tiny delay any non-deterministic number of times. However, such a translation would produce too complex and artificial models, which does not fit into the purpose of a model. Hence, translations must be direct even though details are lost. Finally, let us remind that, depending on the framework, some operations of our methodology could require the participation of a human. For example, checking the equivalence of systems is not decidable if the language is Turing-powerful. The mutant generation function (which applies randomly generated mutations) and the language translation mechanism must be defined by a human, though they can be automatically applied once they are defined.

4 Properties of the Formal Framework

Next we will study the theoretical properties of the concepts presented in the previous section. In particular, we relate the suitability notions presented in Definitions 8 and 9 and we study sufficient conditions to guarantee them. The following results assume the notation criteria introduced in the beginning of Definition 6, that is, $\mathcal{L}_1 = (\alpha_1, \beta_1)$ and $\mathcal{L}_2 = (\alpha_2, \beta_2)$ are two languages, $\mathtt{S}$ is a set of term substitution functions, and M is a mutation function for $\mathcal{L}_1$ and $\mathtt{S}$. Besides, T is a translation function from $\mathcal{L}_1$ to $\mathcal{L}_2$, c_1 and c_2 are correctness criterions for $\mathcal{L}_1$ and $\mathcal{L}_2$, respectively, and $s \in \mathtt{Systems}(\mathcal{L}_1)$.

First, let us note that the relation of collision pairs (see Definition 6) $\updownarrow$ is a symmetric relation.

Lemma 1. For any $m_1, m_2 \in M(s)$ we have that $m_1 \updownarrow m_2$ implies $m_2 \updownarrow m_1$.

Proof. If $m_1 \updownarrow m_2$ then $\mathtt{CMistake}(m_2)$ holds, $m_1, m_2 \in M(s)$, and $T(m_1) = T(m_2)$. Besides, m_1 and m_2 fulfill either the case (a) or the case (b) of Definition 6. In the first case, $\beta_1(m_2) \in c_1(\beta_1(s))$ and $\beta_1(m_1) \notin c_1(\beta_1(s))$. Since

$m_1, m_2 \in M(s)$, $T(m_1) = T(m_2)$, and both $\beta_1(m_1) \notin c_1(\beta_1(s))$ and $\beta_1(m_2) \in c_1(\beta_1(s))$ hold, we conclude $m_2 \updownarrow m_1$ holds. The proof for the case where (b) holds is similar.

□

Nevertheless, the relation $\updownarrow$ is neither reflexive nor transitive. The next result relates collision mistakes, represented by the relation $\updownarrow$, and model mistakes. If two mutants produce a collision mistake then at least one of them is a model mistake.

Lemma 2. Let $m_1, m_2 \in M(s)$ such that $m_1 \updownarrow m_2$. Then, either $\texttt{MMistake}(m_1)$ or $\texttt{MMistake}(m_2)$.

Proof. If $m_1 \updownarrow m_2$ then $\texttt{CMistake}(m_2)$ holds, $m_1, m_2 \in M(s)$, and $T(m_1) = T(m_2)$. Besides, by Lemma 1, we have $\texttt{CMistake}(m_1)$.

Let us suppose that case (a) of Definition 6 holds (the proof is similar for the case where (b) holds). We have $\beta_1(m_2) \in c_1(\beta_1(s))$ and $\beta_1(m_1) \notin c_1(\beta_1(s))$. Let us show that if $\texttt{MMistake}(m_2)$ is false then we have that $\texttt{MMistake}(m_1)$ holds. If $\texttt{MMistake}(m_2)$ does not hold then, by the condition $\beta_1(m_2) \in c_1(\beta_1(s))$, we infer that $\beta_2(T(m_2)) \in c_2(\beta_2(T(s)))$ necessarily holds. Since $T(m_1) = T(m_2)$, we have that $\beta_2(T(m_1)) \in c_2(\beta_2(T(s)))$ is true as well. Besides, since we also have that $\beta_1(m_1) \notin c_1(\beta_1(s))$, we deduce that $\texttt{MMistake}(m_1)$ holds.

Similarly, we prove that if $\texttt{MMistake}(m_1)$ does not hold then $\texttt{MMistake}(m_2)$ is true. If $\texttt{MMistake}(m_1)$ is false then $\beta_2(T(m_1)) \notin c_2(\beta_2(T(s)))$ because we have $\beta_1(m_1) \notin c_1(\beta_1(s))$. Since $T(m_1) = T(m_2)$ and $\beta_1(m_2) \in c_1(\beta_1(s))$, we conclude $\texttt{MMistake}(m_2)$.

□

The previous result allows to relate incomplete collision suitability with incomplete model suitability. If a collision mistake is found while assessing the suitability of a modelling language to represent a set of systems, then the model suitability for this language and set is not full.

Proposition 1. Let $\mathcal{S} = \{(s_1, w_1), \ldots, (s_n, w_n)\} \in \mathcal{P}(\texttt{Systems}(\mathcal{L}_1) \times (0..1])$ with $\sum_{1 \leq i \leq n} w_i = 1$ and such that for all $1 \leq i \leq n$ we have $w_i > 0$. Then,

$$\texttt{CSuit}(\mathcal{S}, \mathcal{L}_2, M, T, c_1) < 1 \Rightarrow \texttt{MSuit}(\mathcal{S}, \mathcal{L}_2, M, T, c_1, c_2) < 1$$

Proof. Taking into account that $\sum_{1 \leq i \leq n} w_i = 1$ and that for all $1 \leq i \leq n$ we have $\texttt{CSuit}(s_i, \mathcal{L}_2, M, T, c_1) \leq 1$, $\texttt{CSuit}(\mathcal{S}, \mathcal{L}_2, M, T, c_1) < 1$ implies that there exists $1 \leq i \leq n$ such that $\texttt{CSuit}(s_i, \mathcal{L}_2, M, T, c_1) < 1$. This implies $\texttt{NumCM}^{M,T,c_1}(s_i) \geq 1$. Then, there exist m_1 and m_2 such that $m_1 \updownarrow^{s_i,M,T,c_1} m_2$.

By Lemma 2, either $\texttt{MMistake}^{s_i,M,T,c_1,c_2}(m_1)$ or $\texttt{MMistake}^{s_i,M,T,c_1,c_2}(m_2)$. In both cases we have $\texttt{NumMM}^{M,T,c_1,c_2}(s_i) \geq 1$. Thus, we also have that the condition $\texttt{MSuit}(s_i, \mathcal{L}_2, M, T, c_1, c_2) < 1$ holds. If we put this disequality together with the fact that for all $1 \leq j \leq n$ we have $\texttt{MSuit}(s_j, \mathcal{L}_2, M, T, c_1, c_2) \leq 1$ and $\sum_{1 \leq j \leq n} w_j = 1$, we infer $\texttt{MSuit}(\mathcal{S}, \mathcal{L}_2, M, T, c_1, c_2) < 1$.

□

Let us note that the previous implication holds in spite of c_2 not being considered in the left side, that is, it holds for any correctness criterion c_2. In the following result we use collision mistakes to find as many model mistakes as possible. Since each two mutants involved in a collision mistake induce at least one model mistake, we will group mutants into pairs of collision mistakes. In order to avoid that two pairs provide the same model mistake (this could happen if both pairs share a mutant), we will require that these pairs are *disjoint*. In this way, each pair will add up one model mistake. In the following result, # means *"number of."* Besides, graphs are denoted by pairs (V, E) where V is the set of *vertices* and E is a set such that each of its elements is a set that is comprised of exactly two (distinct) vertices. The elements of E are called *edges*.

Proposition 2. Let $G = (V, E)$ be a graph with $V = M(s)$ and $\{m, m'\} \in E$ iff $m \updownarrow m'$. Let us consider the set of graphs

$$\mathcal{G} = \{G' \mid G' = (V, E') \ \wedge \ E' \subseteq E \ \wedge \text{ no path in } G' \text{ traverses 3 different nodes}\}$$

Let $n = \max\{\#\text{ connected components with 2 nodes in } G' \mid G' \in \mathcal{G}\}$. We have that $\texttt{MSuit}(s, \mathcal{L}_2, M, T, c_1, c_2) \leq \frac{\mid M(s) \mid -n}{\mid M(s) \mid}$.

Proof. Let $G' = (V', E') \in \mathcal{G}$ be the graph where the number of connected components of 2 nodes is maximal. For any edge $\{m, m'\} \in G'$ we have $m \updownarrow m'$. Due to Lemma 2, we have that either $\texttt{MMistake}(m)$ or $\texttt{MMistake}(m')$. Hence, each connected component of 2 nodes in G' provides a model translation mistake. Since the sets of nodes of each connected component in G' are disjoint, mistakes provided by two components do not coincide. Thus, since $\texttt{NumMM}(s) \geq n$, we have $\texttt{MSuit}(s, \mathcal{L}_2, M, T, c_1, c_2) \leq \frac{\mid M(s) \mid -n}{\mid M(s) \mid}$.

□

Next we present a simple sufficient condition to guarantee total collision suitability. Intuitively, if the translation function is injective then two mutants cannot collapse into a single model, so the translation of a correct system and an incorrect system cannot produce correctness inconsistencies. Hence, the collision suitability of a modelling language to represent a set of systems is full.

Proposition 3. Let $\mathcal{S}$ be defined as in Proposition 1. If T is injective then we have $\texttt{CSuit}(\mathcal{S}, \mathcal{L}_2, M, T, c_1) = 1$.

Proof. If T is injective then there do not exist distinct $m, m' \in M(s)$ such that $T(m) = T(m')$. Thus, for all $1 \leq i \leq n$ we have that there does not exist $m \in M(s)$ with $\texttt{CMistake}^{s_i, M, T, c_1}(m)$. Thus, for all $1 \leq i \leq n$ we have $\texttt{NumCM}^{M, T, c_1}(s_i) = 0$, implying that $\texttt{CSuit}(s_i, \mathcal{L}_2, M, T, c_1) = 1$. Since $\sum_{1 \leq i \leq n} w_i = 1$, we conclude that $\texttt{CSuit}(\mathcal{S}, \mathcal{L}_2, M, T, c_1) = 1$.

□

The following condition allows to obtain full model suitability. If the correctness criterion of the modelling language assesses a model as correct if and only if this model comes from a correct mutant then we obtain *total* model suitability. Moreover, the reverse claim also holds.

Proposition 4. Let $\mathcal{S}$ be defined as in Proposition 1. For all $1 \leq i \leq n$ let $c_1(\beta_1(s_i)) = B_{1i}$ and let c_2 be such that $c_2(\beta_2(t(s_i))) = B_{2i}$, where

(1) If $b \in B_{1i}$ then for all $m \in M(s_i)$, with $\beta_1(m) = b$, $\beta_2(t(m)) \in B_{2i}$ holds and
(2) If $b \notin B_{1i}$ then for all $m \in M(s_i)$, with $\beta_1(m) = b$, $\beta_2(t(m)) \notin B_{2i}$ holds.

Then, $\texttt{MSuit}(\mathcal{S}, \mathcal{L}_2, M, t, c_1, c_2) = 1$. The reverse implication is also true.

Proof. We prove the left to right implication. First, let us show that if c_2 is defined as above then for all $1 \leq i \leq n$ we have $\texttt{NumMM}^{M,t,c_1,c_2}(s_i) = 0$, that is, there do not exist $1 \leq i \leq n$ and $m' \in M(s_i)$ such that we have $\texttt{MMistake}^{s_i,M,t,c_1,c_2}(m')$. By contrapositive, suppose that there exists $1 \leq i \leq n$ such that $\texttt{MMistake}^{s_i,M,t,c_1,c_2}(m)$ for some $m \in M(s_i)$. On the one hand, let us suppose that $\beta_1(m) \in c_1(\beta_1(s_i))$ and $\beta_2(t(m)) \notin c_2(\beta_2(t(s_i)))$. Then, $\beta_1(m) \in B_{1i}$ and $\beta_2(t(m)) \notin B_{2i}$, which makes a contradiction with the definition of c_2 (condition (1)). On the other hand, if we suppose $\beta_1(m) \notin c_1(\beta_1(s_i))$ and $\beta_2(t(m)) \in c_2(\beta_2(t(s_i)))$ then $\beta_1(m) \notin B_{1i}$ and $\beta_2(t(m)) \in B_{2i}$, which is contradictory with condition (2). Hence, $\texttt{MMistake}^{s_i,M,t,c_1,c_2}(m)$ is not possible for any $1 \leq i \leq n$, so $\texttt{NumMM}^{M,t,c_1,c_2}(s_i) = 0$ for all of them. This implies that for $1 \leq i \leq n$ we have $\texttt{MSuit}(s_i, \mathcal{L}_2, M, t, c_1, c_2) = 1$. Since $\sum_{1 \leq i \leq n} w_i = 1$, we conclude $\texttt{MSuit}(\mathcal{S}, \mathcal{L}_2, M, t, c_1, c_2) = 1$.

Next we prove the right to left implication. $\texttt{MSuit}(\mathcal{S}, \mathcal{L}_2, M, t, c_1, c_2) = 1$ implies that for all $1 \leq i \leq n$ we necessarily have $\texttt{MSuit}(s_i, \mathcal{L}_2, M, t, c_1, c_2) = 1$, since for all of them the condition $w_i > 0$ holds and we have $\sum_{1 \leq i \leq n} w_i = 1$. If $\texttt{MSuit}(s_i, \mathcal{L}_2, M, t, c_1, c_2) = 1$ then $\texttt{NumMM}^{M,t,c_1,c_2}(s_i) = 0$. Now we show that if $\texttt{NumMM}^{M,t,c_1,c_2}(s_i) = 0$ then for all $1 \leq i \leq n$ we have that c_2 follows the previous form. By contrapositive, let us suppose that c_2 does not. Then, for some $1 \leq i \leq n$ either (1) or (2) does not hold. On the one hand, let us suppose that there exist $b \in B_{1i}$ and $m \in M(s_i)$, with $\beta_1(m) = b$, such that $\beta_2(t(m)) \notin B_{2i}$. Then, $\beta_1(m) \in c_1(\beta_1(s_i))$ and $\beta_2(t(m)) \notin c_2(\beta_2(t(s_i)))$. So, $\texttt{MMistake}^{s_i,M,t,c_1,c_2}(m)$ and $\texttt{NumMM}^{M,t,c_1,c_2}(s_i) > 0$, which makes a contradiction. On the other hand, let us suppose that there exist $b \notin B_{1i}$ and $m \in M(s_i)$, with $\beta_1(m) = b$, such that $\beta_2(t(m)) \in B_{2i}$. Then, $\beta_1(m) \notin c_1(\beta_1(s_i))$ and $\beta_2(t(m)) \in c_2(\beta_2(t(s_i)))$. Again, $\texttt{MMistake}^{s_i,M,t,c_1,c_2}(m)$ and $\texttt{NumMM}^{M,t,c_1,c_2}(s_i) > 0$, so we also obtain a contradiction. Hence, for all $1 \leq i \leq n$ we have that c_2 fulfills the conditions (1) and (2).

□

Let us note that it is not always possible to construct such a correctness criterion c_2 as required in the previous result. In particular, if there exist $1 \leq i \leq n$ and $m \in M(s_i)$ such that $\texttt{CMistake}^{s_i,M,t,c_1}(m)$ then it will not be possible to fulfill both conditions. Besides, let us note that, even when it is possible, it is not desirable to create *on purpose* a correctness criterion c_2 so that complete suitability is met. On the contrary, the purpose of the correctness criterion is to assess the suitability, so it must be defined *prior to* stating the actual class of systems to be analyzed.

5 Conclusions and Future Work

In this paper we have presented a formal methodology to assess whether a formal modelling language is suitable to represent the *critical* aspects of a system or set of systems. It relies on the idea of creating several alternative systems and using them to exercise the capabilities of the modelling language to distinguish correct/incorrect behaviors. Then, by analyzing whether the modelling language maps correct and incorrect systems as the original language does, we assess the suitability of the modelling language to model these systems.

As future work we plan to apply our methodology to assess the suitability of some well-known modelling languages to represent systems belonging to specific domains. So, our methodology will provide us with an objective and systematic (while heuristic) criterion to compare modelling languages.

References

[AD94] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[AR00] E. Astesiano and G. Reggio. Formalism and method. *Theoretical Computer Science*, 236(1-2):3–34, 2000.

[BG98] M. Bernardo and R. Gorrieri. A tutorial on EMPA: A theory of concurrent processes with nondeterminism, priorities, probabilities and time. *Theoretical Computer Science*, 202:1–54, 1998.

[BG02] M. Bravetti and R. Gorrieri. The theory of interactive generalized semi-Markov processes. *Theoretical Computer Science*, 282(1):5–32, 2002.

[BM99] L. Bottaci and E.S. Mresa. Efficiency of mutation operators and selective mutation strategies: An empirical study. *Software Testing, Verification and Reliability*, 9:205–232, 1999.

[GSS95] R. van Glabbeek, S.A. Smolka, and B. Steffen. Reactive, generative and stratified models of probabilistic processes. *Information and Computation*, 121(1):59–80, 1995.

[Ham77] R.G. Hamlet. Testing programs with the aid of a compiler. *IEEE Transactions on Software Engineering*, 3:279–290, 1977.

[Hil96] J. Hillston. *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.

[How82] W.E. Howden. Weak mutation testing and completeness of test sets. *IEEE Transactions on Software Engineering*, 8:371–379, 1982.

[LN01] N. López and M. Núñez. A testing theory for generally distributed stochastic processes. In *CONCUR 2001, LNCS 2154*, pages 321–335. Springer, 2001.

[NS94] X. Nicollin and J. Sifakis. The algebra of timed process, ATP: Theory and application. *Information and Computation*, 114(1):131–178, 1994.

[Zub80] W.M. Zuberek. Timed Petri nets and preliminary performance evaluation. In *7th Annual Symposium on Computer Architecture*, pages 88–96. ACM Press, 1980.