A Simple Language for Expressing Properties of Telecommunication Services and Features*

C.A. Middelburg**
Dept. of Network & Service Control, PTT Research
and
Dept. of Philosophy, Utrecht University

October 1994

Abstract

This paper reports on a quest for a language for expressing properties of telecommunication services and features, which may play a part in feature interaction detection. A language is sought with a restricted, but practically sufficient, expressive power, which can be complemented with computer-based tools for verification of models described in SDL with respect to properties expressed in the language. A language is suggested which allows the observer technique to be used for checking whether properties expressed in the language are satisfied by models described in SDL. This language can be viewed as a restricted version of the full branching-time temporal logic ACTL*.

1 Introduction

New features are added in telecommunication systems to provide new telecommunication facilities. However these facilities may be somehow in conflict with the existing ones. Feature interaction is the general name for this phenomenon. An extreme kind of feature interactions occurs when characteristic properties of a new feature are inconsistent with properties of the core service or additional features that are already provided. For example, with the Terminating Call Screening feature subscriber C may enforce that he will not receive calls from subscriber A, but subscriber B may use the Call Forwarding Unconditional feature to enforce that subscriber C will receive all his incoming calls – including calls from subscriber A – which is impossible. There are, however, many milder kinds of feature interactions conceivable; from unforeseen, undesirable ones to intended, desirable ones. An undesirable feature interaction occurs,

*This work was partially supported by the RACE project no. 2017, SCORE. It represents the view of the author.

**e-mail: `C.A.Middelburg@research.ptt.nl` for example, if the Call Forwarding Unconditional feature can be used to circumvent the blocking intended when using the Originating Call Screening feature. This is possible if the number being forwarded to is not considered to be a dialled number. However, although it is undesirable, this interaction does not give rise to an inconsistency.[1] A comprehensive survey of current features can be found in [5], which also suggests a categorization of feature interactions.

New features are usually viewed as functionality extensions to the existing services. There are, of course, no feature interactions present if the existing functionality remains unaffected by the newly created functionality, but that is rather the exception than the rule. It is appropriate to call the feature concerned a conservative extension if this favourable situation occurs. However, the situation is unlikely to occur, because new facilities tend to add to the special cases that have to be taken into account by the existing ones. For example, all three above-mentioned features affect dialling such that certain properties of the existing services concerning dialling will no longer hold in the general case. There is at least empirical evidence that useful features are almost inevitably non-conservative extensions. This means among other things that the interactions caused by them are at least partly intended. Furthermore, unintended interactions may be unforeseen but desirable ones. However, this does not mean that there are no undesirable interactions and even ones that should be absolutely excluded.

This paper reports on work done concerning feature interaction detection. It is about a language for expressing properties of telecommunication services and features. The quest is for a "property language" with a restricted, but practically sufficient, expressive power, which can be complemented with computer-based tools for verification of models described in SDL [2] with respect to properties expressed in the language. The underlying idea is that a property language for SDL together with a suitable "model checker" may play a part in feature interaction detection before new features are actually added (see also [3]).

The current section already touched upon the fact that feature interactions are generally inescapable and not necessarily undesirable. Section 2 explains that the way in which a property language for SDL, together with an accompanying model checker, may play a part in feature interaction detection is currently still very limited. Section 3 sketches the approach followed to find a suitable property language and Section 4 presents the results. In Section 5, the observer technique is proposed to check whether properties expressed in the language concerned are satisfied by models described in SDL. Pre-defined functions and predicates that are needed in this language are mentioned in Section 6. Finally, some closing remarks are made in Section 7.

2 Connections with feature interaction detection

In order to minimize the danger of having modelled services and features different from the intended ones, it is important to check whether the models concerned satisfy

[1]Because either is described in isolation, the properties of Call Forwarding Unconditional and Originating Call Screening, as expressed in Section 4, are inconsistent.

anticipated properties. Because SDL is the language generally used in the telecommunication world for modeling, this means that it is useful to have a property language for SDL and an accompanying model checker – to check whether models described in SDL satisfy properties formulated in the property language. But which part can they play in a systematic way to detect feature interactions?

Let us assume that the existing services have been modelled using SDL, that their characteristic properties have been given in the property language, and that it has been checked that the model satisfies these properties. An example of such properties is the following crucial property of the Terminating Call Screening feature: subscriber B should not receive calls from subscriber A when A is on B's screening list. It must be considered to be a major problem in itself to reach the situation sketched above, but we suppose that this problem can be solved. In [16], the following desirable, but not very realistic scenario for the addition of a new feature is mentioned.

The characteristic properties of the new feature are identified and they are expressed in the property language. Next, a model of the new feature is described in SDL and it is checked, using the model checker, whether this model satisfies the characteristic properties. Finally, the model of the existing services and the model of the new feature are combined and it is checked whether the combined model satisfies the union of all the characteristic properties concerned (it is not made precise in [16] what is meant by combining models).

Because new features are frequently non-conservative extensions, there are several shortcomings of this scenario. If the final step does not succeed, this does not have to mean that a case of inconsistent properties has been detected;[2] it does not even have to mean that there is an undesirable feature interaction. It might indicate that the model of the existing services is not suited to extension without changes – the question remains how this must be dealt with. It is moreover likely that the characteristic properties of the existing services have to be adapted – to take new special cases into account – and that the model described in SDL has to be changed accordingly. This has to be accomplished before the final step can be performed.

Suppose, for the sake of simplicity, that only the Basic Call service exists and that the Call Forwarding Unconditional feature has to be added. The latter feature changes the effect of dialling such that the characteristic properties of the Basic Call service concerning dialling will no longer hold. The original characteristic properties would now, for example, lead to such impossible situations as getting at the same time a ring-back tone from the subscriber that a call is intended for as well as a busy tone from the subscriber that the call is forwarded to. So clearly, they must be adapted and so must be the model of the Basic Call service.

The adaptations needed in cases such as the one described above are the result of interactions introduced by the new feature. They are needed to turn the undesirable interactions into acceptable ones. But how do we detect these undesirable interactions? They must have been detected before the step meant for the detection of undesirable feature interactions can be performed! In the example just given, we should have detected beforehand that the addition of the Call Forwarding Uncondi-

[2]Some generic model checkers, e.g. SMV [4], can be used for checking the consistency of properties.

tional feature leads to situations in which a subscriber gets a ring-back tone as well as a busy tone.

All this indicates that the systematic approach to add new features, which incorporates detection of undesirable interactions with existing ones, suggested by the scenario given above, is not to be expected soon. We are, for example, far away from having identified the kinds of feature interactions that are likely to be undesirable; the empirical results needed to identify them require a lot of experiments in adding new features to existing ones and in detecting feature interactions. However, as explained at the beginning of this section, it remains useful to check whether a new model, obtained by adapting an existing one to the needs of a new feature, satisfies all relevant properties.

3 Finding a suitable property language

In this section the approach followed to find a suitable property language is described.

The following approach has been followed in order to find a property language with a restricted, but practically sufficient, expressive power, which can be complemented with a model checker for verifying models described in SDL with respect to properties expressed in the language.[3]

Characteristic properties of features have been expressed, from the subscriber's point of view, in a highly expressive language, viz. a first-order version of ACTL* [15]. But it has been further investigated whether there are common forms of formulae of this logic, which suffice for expressing these properties.

As in [10], we focussed on the subscriber's point of view. This means that we dealt with properties that can be expressed in terms of:

- the events that can be produced by subscribers at their telephones, such as taking off-hook, putting on-hook and dialling a number;
- the observable states of the subscriber's telephones, such as being idle, emitting a dial-tone, etc.;
- the phases of a call that are recognizable through the observable states, such as the ready phase, the calling phase, etc.;
- the features that subscribers have activated.

Observable states and phases are global states, i.e. they comprise a succession of internal states of the telecommunication system. They are viewed as (basic and derived, respectively) predicates on the internal states. The properties on which we focus in this manner are the ones that are really relevant to telephone subscribers. This also means that, in an explanation of features meant for their (potential) users, other properties do not matter.

[3]An alternative to the official semantics of SDL [17], defining the meaning of SDL specifications as labelled transition systems like in [11], is assumed.

The connection with models described in SDL is clear. Events correspond to signals from the environment. Each observable state may encompass many consecutive SDL states satisfying a common predicate explicitly definable in terms of the values of certain variables, the contents of the input port queue of certain process instances, etc.; and so does each phase (see also Section 6). Note, however, that by giving the characteristic properties these predicates are only specified up to the point that is relevant from the subscriber's point of view. Unlike this is the way in which characteristic properties of features are expressed in [7]. A specialization of linear-time temporal logic for SDL is used there to express the properties directly in terms of the specifics of a given model of the services and features involved.

In order to make analysis practically feasible, a suitable abstraction is made. For example, an event produced by a subscriber is regarded to happen at the moment that the telecommunication system starts to handle the event. Thus, some exceptional cases are not covered; but otherwise even the Basic Call service appears to be a complete chaos.

ACTL*, for Action-based CTL*, can be viewed as CTL* extended with relativized next operators.[4] This highly expressive logic is a sublogic of the logic used as the general SPECS property language (SPECS PL) described in [18]; it is essentially the SPECS PL without its fixed-point operator, but with an until operator – which can be introduced as an abbreviation in the SPECS PL using the fixed-point operator.

ACTL* has, in addition to the usual logical operators – $\top$ (true), $\neg$ (not), $\wedge$ (and), $\forall$ (for all) – of classical first-order logic, the following temporal operators: $\boldsymbol{X}$ (nexttime), $\boldsymbol{X}_\alpha$ for each transition label α (relativized nexttime), $\boldsymbol{U}$ (until) and $\boldsymbol{A}$ (for all paths). A transition label α is either an element a from a set of actions A or the special label τ (silent action). The intuition behind these operators is as follows:

- $\boldsymbol{X}\ \varphi$ means that φ will be true after the next transition,
- $\boldsymbol{X}_\alpha\ \varphi$ means that the next transition will be an α transition and φ will be true after this transition,
- $\varphi\ \boldsymbol{U}\ \psi$ means that ψ will eventually be true and until then φ will be true,
- $\boldsymbol{A}\,\varphi$ means that φ will be true for all paths starting from the current state.

τ transitions are used to model transitions where the action involved is hidden from the environment, e.g. the internal steps of a system. The first-order version of ACTL* is precisely defined in Appendix A.

Some well-known temporal operators that can be introduced as abbreviations are $\boldsymbol{F}$ (finally or sometime), $\boldsymbol{G}$ (globally or always), and $[a]$ (inevitably after a):

- $\boldsymbol{F}\,\varphi$ stands for $\top\ \boldsymbol{U}\ \varphi$,
- $\boldsymbol{G}\,\varphi$ stands for $\neg\ \boldsymbol{F}\,\neg\ \varphi$,
- $[a]\,\varphi$ stands for $\boldsymbol{A}\,\neg\ \boldsymbol{X}_a\ \neg\,((\boldsymbol{X}_\tau\ \top)\ \boldsymbol{U}\ \varphi)$.

$[a]\,\varphi$ means that for all paths from the current state with an a transition as its first transition, after this a transition and zero or more directly following τ transitions,

[4]CTL* [9] is the standard full branching-time temporal logic.

φ will be true. So the operator $[a]$ is slightly different from the one in the standard Hennessy-Milner Logic of [12], where it is allowed to have τ transitions directly preceding the a transition as well. We also use the abbreviation $[a_1, \ldots, a_n]\,\varphi$ for $[a_1]\,\varphi \land \ldots \land [a_n]\,\varphi$.

Before we proceed with our quest for a suitable property language, it is worth recalling that we are not looking here for a language which allows us to formulate any property as elegantly or naturally as possible; the emphasis is on the practical feasibility of model checking. We are also not looking for a language which allows us to express the purpose of a new feature as described by the service provider who wants to provide the feature; it is the functionality extension agreed with the service provider to reach this purpose what matters to feature interaction detection. The purpose is elusive; what it means to reach the purpose is usually not even considered. For example, the purpose of the Terminating Call Screening feature is: not to be disturbed by telephone calls from certain people. This gives us little clue about the interactions this feature may cause; what is offered to prevent such disturbance determines these interactions. Of course, the purpose of the feature suggests potential interactions to experts. This means that it is suitable for guesswork, but it is not amenable to rigorous analysis.

4 Common forms of formulae

In practice, the crucial properties of most features can be expressed by formulae of the following general forms:

1. $\boldsymbol{A}\,\boldsymbol{G}\,\varphi$
2. $\boldsymbol{A}\,\boldsymbol{G}(\varphi \;\Rightarrow\; [a]\,\psi)$

where φ and ψ are formulae without temporal operators, i.e. formulae of classical first-order logic. They are mainly built from atomic formulae concerning the observable states of the subscriber's telephones, the phases of a call that are recognizable through the observable states, and the features that subscribers have activated. a is an action label corresponding to an event that can be produced by subscribers at their telephones.

In principle, we might also need formulae of the following general form:[5]

3. $\boldsymbol{A}\,\boldsymbol{G}(\varphi \;\Rightarrow\; \neg\,[a]\,\psi)$

Indeed, we actually need a few formulae of this additional form to formulate some general response properties of the system. For example, the following formula is needed to express that if the telephone of a subscriber is ready for dialling, it is possible for him or her to dial another subscriber:

$$\boldsymbol{A}\,\boldsymbol{G}(A \neq B \land \mathit{ready}(A) \;\Rightarrow\; \neg\,[\mathit{dial}(A,B)]\,\bot)$$

[5]The conjecture is that the additional form allows for any degree of non-bisimilarity to be distinguished (see [14]).

None of the properties of this kind is specific to a certain feature. The technique to check whether properties are satisfied by a model described in SDL, which is explained in Section 5, can be adapted to properties expressed by formulae of this additional form. But, unlike the original technique, the use of the adapted one is not supported by any commercially available SDL-toolset.

A formula of the form 1 expresses a state invariance property, i.e. property that will hold at all states along all possible paths. A formula of the form 2 is a transition rule; it expresses a property that will hold for all state transitions along all possible paths. For example, the crucial properties of the Originating Call Screening (OCS) feature can be expressed by the following formulae if we assume that only the Basic Call service exists:

$$\boldsymbol{A\,G}(A \neq B \land \mathit{OCS}(A,B) \;\Rightarrow\; \neg\; \mathit{calling}(A,B))$$
$$\boldsymbol{A\,G}(A \neq B \land \mathit{OCS}(A,B) \land \mathit{ready}(A) \land \mathit{idle}(B) \;\Rightarrow\; [\mathit{dial}(A,B)]\; \mathit{rejecting}(A))$$
$$\boldsymbol{A\,G}(A \neq B \land \neg\; \mathit{OCS}(A,B) \land \mathit{ready}(A) \land \mathit{idle}(B) \;\Rightarrow\; [\mathit{dial}(A,B)]\; \mathit{calling}(A,B))$$

The first formula is of the form 1. It expresses that the phase where subscriber A is calling subscriber B will never occur when B is on A's screening list. The last two formulae are of the form 2. The first of them expresses that, if A's telephone is ready for dialling and B's telephone is idle, but B is on A's screening list, the phase where A's call attempt is rejected will occur after A dials B. The second of them expresses that the calling phase will occur after A dials B, as previously being usual, if it is instead not the case that B is on A's screening list. For clearness' sake, we mention here that $\mathit{calling}(A, B)$ indicates the phase during which B's telephone is ringing and A gets a ring-back tone and that $\mathit{rejecting}(A)$ indicates the phase during which A gets a busy tone.

Note that the last two formulae change the property of the Basic Call service expressed by the following formula:

$$\boldsymbol{A\,G}(A \neq B \land \mathit{ready}(A) \land \mathit{idle}(B) \;\Rightarrow\; [\mathit{dial}(A,B)]\; \mathit{calling}(A,B))$$

This adaptation is needed because the Originating Call Screening feature affects dialling.

The crucial properties of many other features can be expressed in the same vein, e.g. Terminating Call Screening, Call Forwarding Unconditional and Call Forwarding on Busy/No Answer. Call Forwarding Unconditional (CFU), for example, can be described as follows if we again assume that only the Basic Call service exists:

$$\boldsymbol{A\,G}(A \neq B \land A \neq C \land \mathit{CFU}(A,B) \land \mathit{ready}(C) \land \mathit{idle}(B) \;\Rightarrow\; [\mathit{dial}(C,A)]\; \mathit{calling}(C,B))$$
$$\boldsymbol{A\,G}(A \neq B \land A \neq C \land \mathit{CFU}(A,B) \land \mathit{ready}(C) \land \mathit{busy}(B) \;\Rightarrow\; [\mathit{dial}(C,A)]\; \mathit{rejecting}(C))$$
$$\boldsymbol{A\,G}(A \neq C \land \neg\,(\exists B \cdot \mathit{CFU}(A,B)) \land \mathit{ready}(C) \land \mathit{idle}(A) \;\Rightarrow\; [\mathit{dial}(C,A)]\; \mathit{calling}(C,A))$$
$$\boldsymbol{A\,G}(A \neq C \land \neg\,(\exists B \cdot \mathit{CFU}(A,B)) \land \mathit{ready}(C) \land \mathit{busy}(A) \;\Rightarrow\; [\mathit{dial}(C,A)]\; \mathit{rejecting}(C))$$

The first two formulae express how dialling is affected if Call Forwarding Unconditional is activated by the called party and the last two formulae express that dialling is not affected if it is not activated. Note that if we instead assume that both the Basic Call service and the Originating Call Screening feature exist, several additional formulae are needed to cover the combinations of features as well. For the sake of simplicity, we will also assume in the remaining examples that only the Basic Call service exists.

The Abbreviated Dialling (ABD) feature can also be described naturally in the same way:

$$\boldsymbol{A}\,\boldsymbol{G}(A \neq B \wedge ABD(A,B,N) \wedge \mathit{ready}(A) \wedge \mathit{idle}(B) \Rightarrow [\mathit{dial}(A, \mathit{abbr}(N))]\, \mathit{calling}(A,B))$$
$$\boldsymbol{A}\,\boldsymbol{G}(A \neq B \wedge ABD(A,B,N) \wedge \mathit{ready}(A) \wedge \mathit{busy}(B) \Rightarrow [\mathit{dial}(A, \mathit{abbr}(N))]\, \mathit{rejecting}(A))$$

Here $\mathit{abbr}(N)$ is used to tag N as an abbreviated number – to make it distinguishable from a subscriber number.

Most features need mainly formulae of the form 2, but some features only need formulae of the form 1, e.g. Calling Number Delivery (CND) and Unlisted Number (UN):

$$\boldsymbol{A}\,\boldsymbol{G}(A \neq B \wedge CND(A) \wedge \mathit{calling}(B,A) \Rightarrow \mathit{delivering}(B,A))$$
$$\boldsymbol{A}\,\boldsymbol{G}(UN(B) \Rightarrow (\forall C \cdot \neg\, \mathit{delivering}(B,C)))$$

Note that these two features are trivially inconsistent. If subscriber B is calling subscriber A while A has activated CND, A should have B's number delivered during the calling phase. But if additionally B has activated UN, this is in contradiction to B's demand that his or her number should never be delivered to anybody.

The Automatic Call Back (ACB) feature can be described as well, but at the cost of introducing an auxiliary predicate *acbsubscr* indicating which subscriber is called if the Automatic Call Back code is dialled:

$$\boldsymbol{A}\,\boldsymbol{G}(A \neq B \wedge ACB(A) \Rightarrow [\mathit{dial}(B,A)]\, \mathit{acbsubscr}(A,B))$$
$$\boldsymbol{A}\,\boldsymbol{G}(A \neq B \wedge \mathit{acbsubscr}(A,B) \wedge \mathit{ready}(A) \wedge \mathit{idle}(B) \Rightarrow [\mathit{dial}(A, \mathit{acbcode})]\, \mathit{calling}(A,B))$$
$$\boldsymbol{A}\,\boldsymbol{G}(A \neq B \wedge \mathit{acbsubscr}(A,B) \wedge \mathit{ready}(A) \wedge \mathit{busy}(B) \Rightarrow [\mathit{dial}(A, \mathit{acbcode})]\, \mathit{rejecting}(A))$$
$$\boldsymbol{A}\,\boldsymbol{G}(A \neq B \wedge \mathit{acbsubscr}(A,B) \Rightarrow [\mathit{dial}(A, \mathit{acbcode})]\, \mathit{acbsubscr}(A,B))$$
$$\boldsymbol{A}\,\boldsymbol{G}(A \neq B \wedge \mathit{acbsubscr}(A,B) \Rightarrow [\mathit{offhook}(C), \mathit{onhook}(C)]\, \mathit{acbsubscr}(A,B))$$
$$\boldsymbol{A}\,\boldsymbol{G}(A \neq B \wedge C \neq D \wedge A \neq D \wedge \mathit{acbsubscr}(A,B) \Rightarrow [\mathit{dial}(C,D)]\, \mathit{acbsubscr}(A,B))$$
$$\boldsymbol{A}\,\boldsymbol{G}(ACB(A) \wedge \mathit{acbsubscr}(A,B) \wedge \mathit{acbsubscr}(A,C) \Rightarrow B = C)$$

The first formula expresses that, if subscriber A has activated ACB, subscriber B becomes the subscriber to be automatically called back immediately after B has dialled A.[6] The second and third formula express that such an automatic call back

[6]There are also descriptions of this feature where B would become the subscriber to be automatically called back immediately after a connection with A had been established.

will take place immediately after A dials the ACB code – *acbcode* is used to represent this code. The following three formulae together express that events other than another subscriber dialling A keep the subscriber to be automatically called back unchanged. The last formula expresses that there can be at most one subscriber to be automatically called back.

In formulae of the form 2, the until operator only plays an inessential part; it is only used to deal with internal steps of the system as modelled by means of SDL. In general, one expects intuitively that the until operator will play an essential part in describing the Automatic Call Back feature. It is surprising that it turned out to be relatively easy to devise the formulae given above, while a satisfactory formulation using the until operator in an essential way could not be found.

In case of the Automatic Call Back feature, the auxiliary predicate that had to be introduced does not seem to be artificial; the notion of the subscriber to be automatically called back, in case the ACB code is dialled, is natural and very relevant to the subscriber having activated the feature. A similar remark applies to the Automatic Recall feature – what could be expected – and also for various other features that require the introduction of auxiliary predicates. It also applies to the predicate needed for the Call Waiting feature. It presents the essentials of a call waiting situation, viz. the subcriber to whom is spoken and the subscriber being on hold. However, this predicate could also be viewed as a phase of a call where at least one subscriber having activated the Call Waiting feature is involved. Note that, from this viewpoint, the Call Waiting feature adds to certain calls a new phase which did not occur before its introduction.

5 Observers to check properties

Observers, which are explained below, can be used to check whether properties expressed by linear-time formulae are satisfied.

Linear-time formulae are formulae of the general form $\boldsymbol{A}\,\varphi$ where φ is a temporal formula without the operator $\boldsymbol{A}$. The formulae of the form 1 given in the previous section are linear-time formulae, but the formulae of the form 2 are not (they are true branching-time formulae). However, for formulae φ and ψ without temporal operators,

$$\boldsymbol{A}\,\boldsymbol{G}(\varphi \;\Rightarrow\; [a]\,\psi)$$

is equivalent to

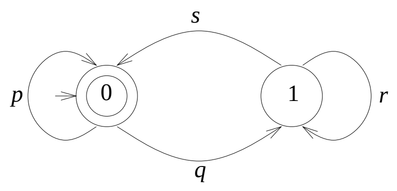
$$\boldsymbol{A}\,\boldsymbol{G}(\varphi \;\Rightarrow\; \neg\; \boldsymbol{X}_a \,\neg\, ((\boldsymbol{X}_\mathcal{T} \,\top)\; \boldsymbol{U}\; \psi))$$

in ACTL* – where all paths through the underlying transition systems are considered (see also [6]).

We first used the branching time form, because it is intuitively less clear for the linear time form that it is an appropriate one to express state transition rules.

An observer is essentially a deterministic automaton accepting certain paths. The principle of model checking by means of observers is that, for a formula $\boldsymbol{A}\,\varphi$ to be checked, a deterministic automaton is constructed that accepts a path if and only if φ is true of that path. This means that, for the formula $\boldsymbol{A}\,\varphi$ to hold, all paths must be accepted by the corresponding observer. Such an observer can always be constructed for a linear-time formula if variables range over a finite domain. The use of observers is supported by, for example, the SDL-toolset GEODE [1], but the construction of observers from temporal formulae – as described in, for example, [8] – has to be done manually yet.

However, the observer for a formula of the form 1 is trivial and the observer for a formula of the form 2, i.e. $\boldsymbol{A}\,\boldsymbol{G}(\varphi \;\Rightarrow\; [a]\,\psi)$, is as follows:



where transition p is labelled with $\neg\,(\varphi \wedge \boldsymbol{X}_a \top)$, q with $\varphi \wedge \boldsymbol{X}_a \top$, r with $\neg\,\psi \wedge \boldsymbol{X}_\mathcal{T} \top$ and s with ψ. The observer will start in state 0 and it will keep this state as long as $\neg\,(\varphi \wedge \boldsymbol{X}_a \top)$ holds. It will pass to state 1 as soon as $\varphi \wedge \boldsymbol{X}_a \top$ holds and it will keep this state as long as $\neg\,\psi \wedge \boldsymbol{X}_\mathcal{T} \top$ holds. It will pass back to state 0 as soon as ψ holds. Thereafter, this pattern of behaviour will repeat itself indefinitely. Note that it may not occur that $\neg\,\psi \wedge \neg\,\boldsymbol{X}_\mathcal{T} \top$ holds while the observer is in state 1. Thus, it will accept exactly the paths that satisfy $\boldsymbol{G}(\varphi \;\Rightarrow\; [a]\,\psi)$!

So, if we stick to temporal formulae of the above-mentioned two general forms, the construction of observers for them is quite easy. The automation of this construction will require only a small effort.

6 Pre-defined functions and predicates

In the previous sections, we used predicates for the observable states of telephones, the phases of telephone calls, etc. To check whether a model described in SDL satisfies properties formulated in (a restricted version of) ACTL*, these predicates must be defined explicitly in terms of the values of certain variables, and the like, that are extant in the model. This section gives an overview of a specialization of ACTL* suited for SDL.

In appendix A, the uninterpreted first-order version of ACTL* is defined. A *partially interpreted* version appears to be more suited for SDL. This means that temporal structures with specific functions and predicates are assumed for certain function and predicate symbols. It seems also useful to have several different domains instead of one, which requires a *many-sorted* version with sort symbols to distinguish between them. Specific domains are assumed for certain sort symbols as well. Thus, we get a specialization of ACTL* for SDL which offers a number of pre-defined sorts, functions and predicates. First the envisaged specialization is sketched and next the grounds for it are given. In the following saved signals and timers are not taken into account.

The pre-defined sorts needed include the pre-defined sorts of SDL (including *PId*, the set of *process instance identifiers*) and additionally:

ChanNm	a finite set of *channel names*;
StateNm	a finite set of *state names*;
Signal	a countably infinite set of *signals*.

Each pre-defined sort of SDL comes together with certain pre-defined functions to construct values of the sort or to extract other values from them; and so does the pre-defined sort *Signal*. In addition to these state-independent functions, some state-dependent functions and predicates are needed.

For each variable declared in an SDL system definition by dcl *vnm Snm*, a function $vnm : PId \to Snm$ is needed; $vnm(pid)$ yields the current contents of the variable with name *vnm* owned by the process instance with identifier *pid*. Besides, the following functions are needed:

$state : PId \to StateNm$	$state(pid)$ yields the name of the current process state of the process instance with identifier *pid*;
$ipfirst : PId \to Signal$	$ipfirst(pid)$ yields the first signal in the input port queue of the process instance with identifier *pid*;
$chfirst : ChanNm \to Signal$	$chfirst(cnm)$ yields the first signal in the channel with name *cnm*.

The following predicates are also needed:

$init : PId$	$init(pid)$ is true if the process instance with identifier *pid* is in the start state;
$ipempty : PId$	$ipempty(pid)$ is true if there are no signals in the input port queue of the process instance with identifier *pid*;
$chempty : ChanNm$	$chempty(cnm)$ is true if there are no signals in the channel with name *cnm*;
$existing : PId$	$existing(pid)$ is true if the process instance with identifier *pid* exists.

In what precedes, we viewed the observable states of the subscriber's telephones, the phases of a call that are recognizable through the observable states and the features that subscribers have activated as predicates on the internal states of the telecommunication system. In order to verify a model described in SDL with respect to properties expressed as ACTL* formulae of certain forms, these predicates have to be defined explicitly in terms of functions and predicates that depend on the system states extant in the model. Such a system state consists of a collection of process instances, where each instance is uniquely identified by a process instance identifier, and a collection of named channels. The process instances contained in the former collection may vary from system state to system state, but the channels contained in the latter collection are the same channels in all system states. The channels convey signals which may catch a delay. A process instance comprises a named process state prescribing its future behaviour, a storage of variables, which determines the value of the variables associated with it, and an input port queue containing signals received by the process instance but not consumed by it.

Clearly, the sorts, functions and predicates proposed above as the pre-defined ones are sufficient to consult any detail of the system states. The whole is essentially part of the sorts, functions and predicates proposed as the pre-defined ones in [19]. Only the fundamental ones to consult all details of the system states are left in the current paper.

7 Closing remarks

Although the crucial properties of many features can currently be expressed by formulae of the above-mentioned two general forms, it is possible that these forms will turn out to be too restrictive in the future. Preliminary investigations indicate that the SMV model checker [4] can be adapted such that it can be used to check whether models described in SDL satisfy properties formulated in ACTL. Roughly, ACTL differs from ACTL* in that each occurrence of $\boldsymbol{A}$ in a ACTL formula must be followed by a boolean combination of formulae of the forms $\boldsymbol{X}\ \varphi$, $\boldsymbol{X}_\alpha\ \varphi$ and $\varphi\ \boldsymbol{U}\ \psi$, where φ and ψ must be ACTL formulae as well. Of course, quantification could also be allowed, provided that the variables concerned range over finite domains.

The description of features by formulae of the above-mentioned two general forms seems closely related to the specification style with state invariants and pre- and post-condition style specifications of operations that is used with VDM [13]. This style enables to associate a number of relatively simple proof obligations with two specifications of a program, whose discharge is sufficient to show that one specification correctly refines the other specification in the following sense: each program satisfying the former specification simulates – when viewed as a transition system – some program satisfying the latter one. A similar approach is advocated to be used with Z [20]. It is useful to investigate whether there are connections between this notion of refinement and undesirable feature interactions.

There are also close connections with the approach to feature interaction detection proposed in [10]. What is new in the current paper, is that τ transitions are taken into account. The "network properties" from that paper are exactly the temporal formulae of the form 1, and the "declarative transition rules" are the temporal formulae of the form 2 apart from the account of τ transitions. There would be an exact match if we had chosen $[a]\,\varphi$ to stand for $\boldsymbol{A}\neg\ \boldsymbol{X}_a\ \neg\ \varphi$. There is nothing like τ transitions in the official semantics of SDL [17]. However, recent proposals for an operational semantics by which the meaning of a system definition is defined as a labelled transition system, all introduce τ transitions to model the internal steps of the system (see e.g. [11]). The current paper also indicates how the notation used in [10] relates to ACTL* and how it can be checked with existing techniques whether models described in SDL satisfy properties expressed in such a notation.

Acknowledgements

Thanks go to Wiet Bouma, Alfo Melisse, Hugo Velthuijsen, Anders Gammelgaard and Simon Pickin for helpful conversations and feedback on the subject of this paper.

References

[1] B. Algayres, Y. Lejeune, F. Hugonnet, and F. Hantz. The AVALON project: A validation enviroment for SDL/MSC descriptions. In O. Færgemand and A. Sarma, editors, *SDL '93: Using Objects*, pages 221–235. North-Holland, 1993.

[2] F. Belina, D. Hogrefe, and A. Sarma. *SDL with Applications from Protocol Specification.* Prentice-Hall, 1991.

[3] L.G. Bouma, W.G. Levelt, A.A.J. Melisse, C.A. Middelburg, and L. Verhaard. Formalisation of properties for feature interaction detection: Experience in a real-life situation. To appear in proceedings of IS&N Conference, September 1994.

[4] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Proceedings Logic in Computer Science 1990.* Computer Society Press of the IEEE, 1990.

[5] E.J. Cameron, N. Griffith, Y.-J. Lin, M.E. Nilson, W.K. Schnure, and H. Velthuijsen. A feature interaction benchmark for IN and beyond. In L.G. Bouma and H. Velthuijsen, editors, *Feature Interactions in Telecommunictions Systems*, pages 1–23. IOS Press, 1994.

[6] E.M. Clarke and I.A. Draghicescu. Expressibility results for linear-time and branching-time logics. In J.W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, pages 428–437. Springer Verlag, LNCS 354, 1989.

[7] P. Combes and S. Pickin. Formalisation of a user view of network and services for feature interaction detection. In L.G. Bouma and H. Velthuijsen, editors, *Feature Interactions in Telecommunictions Systems*, pages 120–135. IOS Press, 1994.

[8] E.A. Emerson and A.P. Sistla. Deciding full branching time logic. *Information and Control*, 61(3):175–201, 1984.

[9] E.A. Emerson and J. Srinivasan. Branching time temporal logic. In J.W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, pages 123–172. Springer Verlag, LNCS 354, 1989.

[10] A. Gammelgaard and J.E. Kristensen. Interaction detection, a logical approach. In L.G. Bouma and H. Velthuijsen, editors, *Feature Interactions in Telecommunictions Systems*, pages 178–196. IOS Press, 1994.

[11] J.C. Godskesen. An operational semantics for Basic SDL. Technical Report TFL RR 1991-2, TFL, 1991.

[12] M. Hennessy and R. Milner. Algebraic laws for non-determinism and concurrency. *Journal of the ACM*, 32(1):137–161, 1985.

[13] C.B. Jones. *Systematic Software Development Using VDM.* Prentice-Hall, second edition, 1990.

[14] R. Milner. *Communication and Concurrency.* Prentice-Hall, 1989.

[15] R. de Nicola and F. Vaandrager. Action versus state based logics for transition systems. In I. Guessarian, editor, *Semantics of Concurrency*, pages 407–419. Springer Verlag, LNCS 469, 1990.

[16] SCORE-Methods and Tools. Report on methods and tools for service creation. Deliverable D2031 – R2017/SCO/WP2/DS/P/027/b2, RACE project 2017 (SCORE), December 1993.

[17] SDL formal definition, dynamic semantics. CCITT Document COM X-R 31: Revised Annex F.3 to Recommendation Z.100, 1992.

[18] SPECS-Semantics and Analysis. Final definition of a property language on the common semantics. Deliverable WP5.13 – 46/SPE/WP5/DS/A/013/a1, RACE project 1046 (SPECS), December 1992.

[19] SPECS-Specification Handling. Final methods and tools for the handling of SDL specifications. Deliverable WP4.15 – 46/SPE/WP4/DS/A/015/b1, RACE project 1046 (SPECS), October 1992.

[20] J.M. Spivey. *The Z Notation: A Reference Manual.* Prentice-Hall, second edition, 1992.

A Definition of first-order ACTL*

Syntax

The language of the first-order version of ACTL*, containing terms and formulae, is defined over a set $\mathcal{F}$ of *function* symbols, a set $\mathcal{P}$ of *predicate* symbols, a set $\mathcal{X}$ of *variable* symbols, and a set A of *actions*. The *silent action* τ is not in A. Every function or predicate symbol has an *arity* n ($n \geq 0$).

The terms of ACTL* are inductively defined by the following formation rules:

- variable symbols are terms;
- if f is a function symbol of arity n and $t_1, \ldots, t_n$ are terms, then $f(t_1, \ldots, t_n)$ is a term.

The formulae of ACTL* are inductively defined by the following formation rules:

- $\top$ is a formula;
- if P is a predicate symbol of arity n and $t_1, \ldots, t_n$ are terms, then $P(t_1, \ldots, t_n)$ is a formula;
- if t_1 and t_2 are terms, then $t_1 = t_2$ is a formula;
- if φ is a formula, then $\neg\ \varphi$ is a formula;
- if φ_1 and φ_2 are formulae, then $\varphi_1 \wedge \varphi_2$ is a formula;
- if φ is a formula and x is a variable symbol, then $\forall x \cdot \varphi$ is a formula;
- if φ is a formula, then $\boldsymbol{X}\ \varphi$ is a formula;
- if φ is a formula and $\alpha \in A \cup \{\tau\}$, then $\boldsymbol{X}_\alpha\ \varphi$ is a formula;
- if φ_1 and φ_2 are formulae, then $\varphi_1\ \boldsymbol{U}\ \varphi_2$ is a formula;
- if φ is a formula, then $\boldsymbol{A}\,\varphi$ is a formula.

The string representation of formulae suggested by these formation rules can lead to syntactic ambiguities: parentheses are used to avoid such ambiguities.

Semantics

The semantics of ACTL* terms and formulae is defined with respect to a temporal structure. A (first-order) *temporal structure* $\mathcal{K}$ is a quintuple $\langle S, A, \rightarrow, D, L\rangle$ where:

- S is a set of *states*;
- A is the set of *actions*;
- $\rightarrow \subseteq S \times (A \cup \{\tau\}) \times S$ is the *transition relation*, the elements from $\rightarrow$ are called *transitions*;
- D is a set, the *domain* of $\mathcal{K}$;
- L maps each state $s \in S$ to an interpretation $L(s)$ that assigns an appropriate meaning over D to all function and predicate symbols of the temporal language, i.e.:
  - for every n-ary function symbol f, a total function $f^{L(s)} : D^n \rightarrow D$;
  - for every n-ary predicate symbol P, a total function $P^{L(s)} : D^n \rightarrow \{\mathsf{T}, \mathsf{F}\}$.

A temporal structure $\langle S, A, \rightarrow, D, L\rangle$ can be viewed as a labelled transition system $\langle S, A, \rightarrow\rangle$ together with a structure $\langle D, L(s)\rangle$ of classical first-order logic for each state $s \in S$ so as to provide for functions and predicates which may vary from state to state.

The truth of formulae is defined for fullpaths in $\mathcal{K}$. A *path* in $\mathcal{K}$ is an element $\pi = \langle s_0, \langle\langle\alpha_1, s_1\rangle, \langle\alpha_2, s_2\rangle, \ldots\rangle\rangle$ from $S \times ((A \cup \{\tau\}) \times S)^\infty$ such that $\langle s_i, \alpha_{i+1}, s_{i+1}\rangle \in \rightarrow$ for all $i < |\pi|$. Here $|\pi|$ is the length of the second component of π, i.e. the number of transitions represented. π is a *fullpath* in $\mathcal{K}$ iff there is no $\langle\alpha, s\rangle \in (A \cup \{\tau\}) \times S$ that appended to the second component of π yields again a path.

Let $\pi = \langle s_0, \langle\langle\alpha_1, s_1\rangle, \langle\alpha_2, s_2\rangle, \ldots\rangle\rangle$ be a path. Then we write π_0 for s_0, π_{l1} for α_1, and π^i $(0 \leq i \leq |\pi|)$ for the path $\langle s_i, \langle\langle\alpha_{i+1}, s_{i+1}\rangle, \ldots\rangle\rangle$. We also write $\mathit{fullpaths}_{\mathcal{K}}(s)$ for the set $\{\pi | \pi \textit{ is a fullpath in } \mathcal{K} \text{ and } \pi_0 = s\}$.

The interpretation of terms and formulae of ACTL* in $\mathcal{K}$ is further given under an assignment in $\mathcal{K}$ – assigning a value to each variable symbol. An *assignment* in $\mathcal{K}$ is a function $\xi : \mathcal{X} \rightarrow D$. For every assignment ξ, variable symbol x and element $d \in D$, we write $\xi(x \rightarrow d)$ for the assignment ξ' such that $\xi'(y) = \xi(y)$ if $y \not\equiv x$ and $\xi'(x) = d$.

The meaning of terms is given by a function mapping term t, structure $\mathcal{K}$, fullpath π and assignment ξ to the element of D that is the value of t in the first state of π under assignment ξ. We write $[\![t]\!]_\xi^{\mathcal{K},\pi}$ to denote the value of this function for the arguments t, $\mathcal{K}$, π and ξ. Similarly, the meaning of formulae is given by a relation associating formula φ, structure $\mathcal{K}$, fullpath π and assignment ξ if φ is true of π in $\mathcal{K}$ under assignment ξ. We write $\mathcal{K}, \pi \models_\xi \varphi$ to indicate that this relation holds for the arguments φ, $\mathcal{K}$, π and ξ. The interpretation functions for terms and formulae are inductively defined by

$$\begin{array}{lcl} [\![x]\!]_\xi^{\mathcal{K},\pi} & = & \xi(x)\ , \\ [\![f(t_1, \ldots, t_n)]\!]_\xi^{\mathcal{K},\pi} & = & f^{L(\pi_0)}([\![t_1]\!]_\xi^{\mathcal{K},\pi}, \ldots, [\![t_n]\!]_\xi^{\mathcal{K},\pi}) \end{array}$$

and

$$\begin{array}{lll}
\mathcal{K},\pi \models_\xi \top & \text{always} & \\
\mathcal{K},\pi \models_\xi P(t_1,\ldots,t_n) & \text{iff} & P^{L(\pi_0)}([\![t_1]\!]_\xi^{\mathcal{K},\pi},\ldots,[\![t_n]\!]_\xi^{\mathcal{K},\pi}) = \mathsf{T}\ , \\
\mathcal{K},\pi \models_\xi t_1 = t_2 & \text{iff} & [\![t_1]\!]_\xi^{\mathcal{K},\pi} = [\![t_2]\!]_\xi^{\mathcal{K},\pi}\ , \\
\mathcal{K},\pi \models_\xi \neg\,\varphi & \text{iff} & \text{not } \mathcal{K},\pi \models_\xi \varphi\ , \\
\mathcal{K},\pi \models_\xi \varphi_1 \wedge \varphi_2 & \text{iff} & \mathcal{K},\pi \models_\xi \varphi_1 \text{ and } \mathcal{K},\pi \models_\xi \varphi_2\ , \\
\mathcal{K},\pi \models_\xi \forall x \cdot \varphi & \text{iff} & \text{for all } d \in D,\ \mathcal{K},\pi \models_{\xi(x\to d)} \varphi\ , \\
\mathcal{K},\pi \models_\xi \boldsymbol{X}\ \varphi & \text{iff} & |\pi| \geq 1 \text{ and } \mathcal{K},\pi^1 \models_\xi \varphi\ , \\
\mathcal{K},\pi \models_\xi \boldsymbol{X}\alpha\ \varphi & \text{iff} & |\pi| \geq 1 \text{ and } \pi_{l1} = \alpha \text{ and } \mathcal{K},\pi^1 \models_\xi \varphi\ , \\
\mathcal{K},\pi \models_\xi \varphi_1\ \boldsymbol{U}\ \varphi_2 & \text{iff} & \text{for some } i,\ \mathcal{K},\pi^i \models_\xi \varphi_2 \text{ and for all } j < i,\ \mathcal{K},\pi^j \models_\xi \varphi_1\ , \\
\mathcal{K},\pi \models_\xi \boldsymbol{A}\,\varphi & \text{iff} & \text{for all } \pi' \in \mathit{fullpaths}_{\mathcal{K}}(\pi_0),\ \mathcal{K},\pi' \models_\xi \varphi\ .
\end{array}$$

A formula φ is *true* of fullpath π in temporal structure $\mathcal{K}$, written $\mathcal{K},\pi \models \varphi$, iff $\mathcal{K},\pi \models_\xi \varphi$ for every assigment ξ.

A formula φ is *valid*, written $\models \varphi$, iff $\mathcal{K},\pi \models \varphi$ for every temporal structure $\mathcal{K}$ and every fullpath π in $\mathcal{K}$.