V. Padovani, H. Comon, J. Leneutre and R. Tingaud

A formal verification of telephone supplementary service interactions

Research Report LSV–99–5, May 1999

Laboratoire Spécification et Vérification

Ecole Normale Supérieure de Cachan
61, avenue du Président Wilson
94235 Cachan Cedex France

http://www.lsv.ens-cachan.fr/Publis/
Research Report LSV-99-5, Lab. Spécification et Vérification, CNRS & ENS de Cachan, France, May 1999

A formal verification of telephone supplementary service interactions

V. Padovani LSV
H. Comon LSV
J. Leneutre Alcatel
R. Tingaud Alcatel

May 21, 1999

1 Introduction

More and more telephone services are offered to the users. Each service alone is assumed to provide the user with an additional facility. For instance the Call Return service allows the user B to automatically call back A if A attempted to call him (her) while (s)he was busy. Over one hundred such additional services have been proposed. A careless development of these services may yield unexpected behaviors when they interact. For instance the Originating Call Screening forbids a user to originate any call from a given screening list. However, when combined with the Call Return service, it may happen that B automatically calls back somebody who is on his screening list (which (s)he may not want).

This problem of services interactions is well-known (see for example the annual workshop [KB98]). There are several attempts to identify most of the interactions (see [Zav99]) or to classify such interactions (e.g. [CGL+94]). See also [KK98] for a recent survey of supplementary services interactions, together with over one hundred references on the subject.

The ALCATEL company is developping services and seeks predicting their interactions beforehand, avoiding unexpected behaviors. This paper describes a case study in service interactions which has been carried jointly by ALCATEL and the public research laboratory LSV. The purpose of the case study is to formally check the expected properties of a service against a specification of its implementation, in presence of additional services. More generally, we want to derive necessary and sufficient conditions under which the expected property of a service is not modified when some other services interact. Up to our knowledge, this is the first formal exhaustive study of some supplementary services interactions. There are a number of formal studies of interactions (for instance [KL98], and, for a more exhaustive list, [KK98]). However, all these studies may succeed in detecting some undesirable interactions (called *interferences*, following the terminology of [KK98]), but do not prove that the list is exhaustive. In principle, it is possible to detect interferences by some well chosen simulations, while proving that there are no (other) interferences (than those already

detected) requires a formal proof of some universally quantified statement. One other originality of our work is to consider three supplementary services together (while, usually, the interactions are considered pairwise, which, as we will see, is not sufficient).

Goals

We consider here three supplementary services: the *Closed User Groups* (CUG, [cug92]), the *Explicit Call Transfer* (ECT, [ect97]) and the *Conference* (CONF, [con88]) services. Roughly, CUG enables users to form groups to and from which access is restricted. A user may subscribe to several groups. In addition, a user, for each of his CUG memberships, may subscribe to several options, according to which he may (or may not) receive (resp. originate) some calls outside of the group. This service is described in more details in section 2.1. The ECT service allows a user which is communicating with two other users to connect the two calls into a single call (see figure 4). The CONF service allows a user to create a conference bridge and to add to the bridge any user with whom he is communicating. (See section 2.3 for more details). Both the expected behaviors and the specifications of the implementations are described in technical requirements documents. We focused on the following property:

(P) Are two connected users allowed to communicate?

More precisely, from the CUG expected properties, some communications are not allowed. For instance, if A belongs to the group G and B does not and if A does not have the outgoing access (OA), then he cannot originate a call to B. We want first to be sure that the description of the CUG implementation indeed rejects such communicating attempts (and not any other attempt). Then we want to find all situations in which the above property fails in presence of ECT (resp. CONF, resp. both services).

Building the model

We first constructed a formal communication model. We chose to describe the communications in first-order logic, regardless of the details of the communication protocols. This choice is relevant here since the implementation description can be specified at this high level of abstraction. Typically, we consider a finite number of events which are call attempts, attempts to use features provided by the supplementary services, or call releases. Given an event sequence ES, we specify a predicate $\mathsf{Connected}(ES, x, y)$ which holds true if the users x and y are indeed connected after the sequence ES. This specification is constructed according to the implementation descriptions: some attempts are rejected according to the actual verifications which are performed before establishing the communication. We present in more details the data specifications and the definition of $\mathsf{Connected}$ in section 3. We also design a formula ϕ which expresses property (P), according to the general description of CUG.

The choice of first-order logic, instead of temporal logic which underlies most of previous formal approaches to the problem, is relevant here since our property (P) expresses that something holds, regardless to the event sequence. (i.e. for all event sequences). This would not be appropriate for more elaborated temporal properties such as "a property Q is eventually satisfied, each time an event E occurs".

Results

The next step is to try to formally prove the property (P), i.e.

$$(P) \qquad \mathcal{S} \models \forall ES, x, y.\mathsf{Connected}(ES, x, y) \Rightarrow \phi(x, y)$$

For this purpose, we used a proof assistant: the Larch Prover [GG95]. Hence, in what follows, when we say that something is proved, we always mean that the proof has been mechanically checked. We managed to complete the proof for the CUG service alone, showing (to some extent) that the implementation of CUG meets indeed the requirements. Then we found an additional condition $\mathcal{C}_{CUG+ECT}$, restricting the use of call tranfers when they interact with CUG and prove that this condition is necessary: if $\mathcal{C}_{CUG+ECT}$ is (strictly) weakened to any condition C', then we proved that $\mathcal{S}, C' \models \exists ES, x, y.\mathsf{Connected}(ES, x, y) \wedge \neg\phi(x, y)$. We also proved that $\mathcal{C}_{CUG+ECT}$ is sufficient:

$$\mathcal{S}, \mathcal{C}_{CUG+ECT} \models \forall ES, x, y.\mathsf{Connected}(ES, x, y) \Rightarrow \phi(x, y)$$

Then the same work has been done with the three services together, deriving a condition $\mathcal{C}_{CUG+ECT+CONF}$ with the same properties.

2 Description of the services

In what follows, we distinguish a *call* from a *connection*: the former one has an origin and a destination, while the second one is symmetric. We will see in the next section a more formal definition (roughly, a *call* is an event while a *connection* is a predicate).

2.1 Closed User Groups

Any user may subscribe to one or more *closed user groups* and, for each of them one or more basic services are specified. Together with the CUG subscription, the user may subscribe to additional options: the *outgoing access* (OA) which allows to originate normal calls, and the *incoming access* (IA) which allows to receive normal calls. Within a specific group, a user can also subscribe to *Outgoing Call Barring* (OCaB) which prevents from originating calls to other members, and *Incoming Call Barring* (ICaB) which prevents from receiving calls from them.

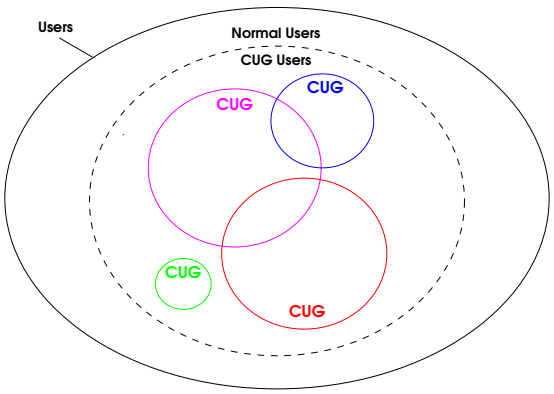

Figure 1: The CUG supplementary service

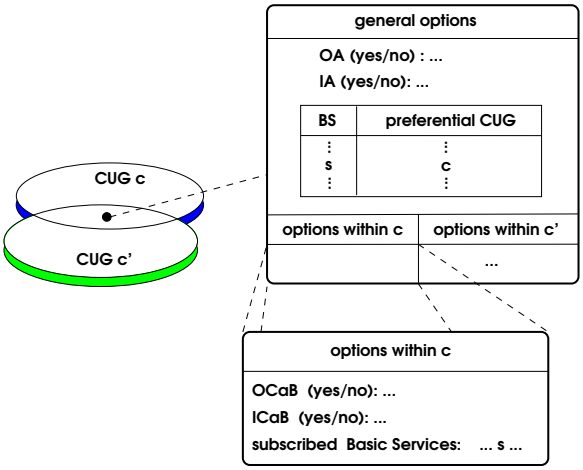

Figure 2: CUG: Subscription options

Finally, any user who subscribed, for the same basic service, to several closed user groups, can specify a preferential group for each of these services. All the subscription options are summarized in figure 2.

A quick analysis of this (informal) specification shows that there are five cases where a user A is allowed to call B. These cases are depicted on figure 3. The *type* of a communication is either *Normal* or *CUG*, and, in the latter case, it has as an attribute the CUG name. The first four cases yield normal communications, while the last one yields a CUG call.

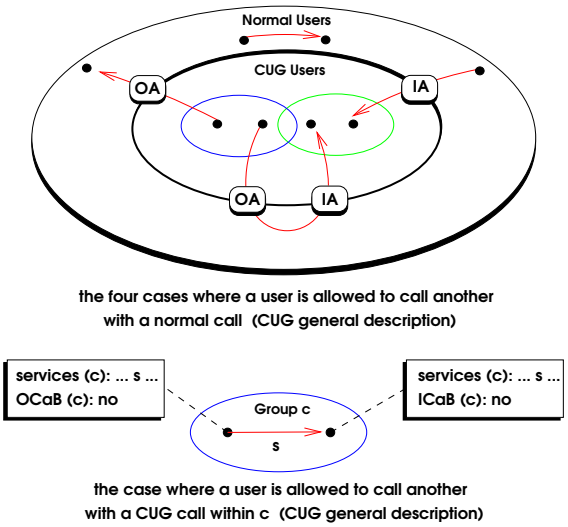

Figure 3: Allowed calls

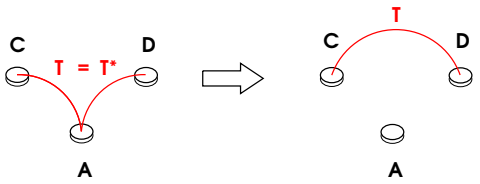

Figure 4: Explicit Call Transfer

2.2 Explicit Call Transfer

When A subscribed to the ECT service and when he has two communications, one with C and one with D, he may invoke the ECT and join the two communications into a single one. In case of CUG subscribers, there is an additional restriction: the two communications should have the same type (both normal or both within the same CUG). The resulting communication is of that type. After the transfer, the two original communications are released. This is depicted on figure 4.

2.3 Conference

The conference service enables a user to participate in, and control, a simultaneous communication involving a number of users. All users are connected to a *bridge*, which allows them to hear each other. The creator of the conference controls it until it is released. While the conference is active, the controller may

still receive and originate calls and also add new participants to the bridge.

A conference assumes the type of the communication between its creator and the first added user. The creator may subsequently add any user which he has a private communication with, provided the latter and the conference are of same type.

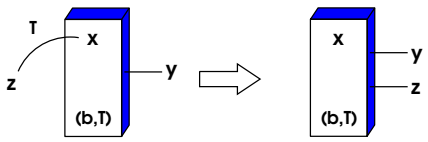

Figure 5: Addition of z to a bridge b of type T, controlled by x

Any conference participant, except its creator, may transfer by ECT its connection to the bridge to another user which he is communicating with, as long as the bridge and the communication are of same type. Also, when a conferee is simultaneously connected to two bridges of same type, he can perform an ECT between these connection, as long as the transfer does not create any loop.

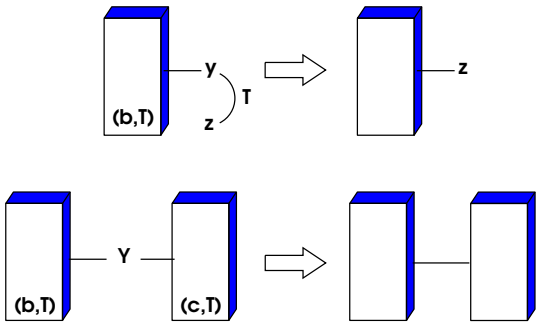

Figure 6: Transfers in the combination of CUG, CONF and ECT

3 Specifications

We use here the specification language Larch [GG95], which is based on multi-sorted first-order logic. In order to have more readable specifications, we are not going to give details nor to use the Larch syntax.

3.1 Basic data types and predicates

We consider as basic data types the users, the lines, the call types, the groups, the bridges, the basic services and the event sequences. Lines are intended to keep track of all connections between users, their states being adjusted in accordance

with new events. We assume an arbitrary supply of users, groups, bridges and lines. Depending on the services we consider, the event sequence can be built in different ways.

Subscription options are described using predicates such as $\mathsf{Subscribed_Service}(x, s, g)$ whose intended meaning is that user x subscribed to the basic service s within the group g.

The predicate $\mathsf{Allowed_Call}(x, s, y, T)$ is intended to hold true when x may call y, using the basic service s and with a resulting call type T.

The predicate $\mathsf{Connected}(x, y, T, ES)$ expresses that, after the event sequence ES, x is connected to y with a connection of type T. For instance, the property (P) can be formally expressed as

$$\mathcal{S} \models \forall x, y, T, ES. \left(\mathsf{Connected}(x, y, T, ES) \Rightarrow \exists s. \left(\begin{array}{ll} & \mathsf{Allowed_Call}(x, s, y, T) \\ \vee & \mathsf{Allowed_Call}(y, s, x, T) \end{array} \right) \right)$$

which is a refinement of the expression given in introduction.

3.2 Specification of allowedness

In this setting, the allowedness is easy to specify, translating figure 3. For instance

$$\mathsf{Allowed_Call}(x, s, y, \mathsf{Normal}) \stackrel{\text{def}}{=} \left(\begin{array}{ll} & (x \neq y) \\ & \wedge \\ & \left(\begin{array}{ll} & \mathsf{Normal_User}(x) \\ \wedge & \mathsf{Normal_User}(y) \end{array} \right) \\ \vee & \left(\begin{array}{ll} & \mathsf{CUG_User}(x) \\ \wedge & \mathsf{OA}(x) \\ \wedge & \mathsf{Normal_User}(y) \end{array} \right) \\ \vee & \left(\begin{array}{ll} & \mathsf{Normal_User}(x) \\ \wedge & \mathsf{CUG_User}(y) \\ \wedge & \mathsf{IA}(y) \end{array} \right) \\ \vee & \left(\begin{array}{ll} & \mathsf{CUG_User}(x) \\ \wedge & \mathsf{OA}(x) \\ \wedge & \mathsf{CUG_User}(y) \\ \wedge & \mathsf{OA}(y) \end{array} \right) \end{array} \right)$$

where $\mathsf{Normal_User}(x) \stackrel{\text{def}}{=} \neg \mathsf{CUG_User}(x)$.

3.3 Specification of the CUG alone

In the case where the CUG service is considered alone, there are only two kinds of events: call attempts and call releases. Call attempts take as arguments all relevant parameters to establish a communication, including a caller, a callee, a requested basic service, possibly a CUG request - for instance a request that the call is performed within a specific group. All these informations are gathered into a *Setup*.

The technical requirements describe how some verifications are performed on a setup at the originating network and at the destination network. During these two phases, depending on the subscription options of both users, the setup may be rejected or not. If not, the communication is established on the basis of a computed call type. The whole process is modelized by the function CUG_Check. Applied to a setup, it returns two possible kinds of values: reject, or accept(T) where T is the computed call type.

Formally, events are defined as an inductive type whose constructors are

- The empty event (constant)
- Call : setup $\times$ line $\times$ event sequence $\rightarrow$ event sequence
- Release : line $\times$ event sequence $\rightarrow$ event sequence

Note that besides a setup, a call attempt takes a line as an argument. At the time of the call, there are two possible states for it: free, or already connecting two users, the connection being of normal type or within some closed user group. The line is checked to be free, and subsequently connects the caller and the callee by a connection of the computed call type if the attempt succeeds that is, if the setup passes the CUG checks.

At this stage, the predicate Connected(x, y, T, ES) holds if and only if there exists at least one line connecting, after ES, x and y by a connection of type T:

$$\mathsf{Connected}(x, y, T, ES) \Leftrightarrow \exists L\, (\mathsf{Line_State}(L, ES) = \mathsf{Connecting}(x, y, T))$$

We cannot fully describe the CUG checks in this paper (which depends for instance on the preferential CUG, or the specified CUG at the call attempt). The only thing which is important to notice is that, in this simple situation, event sequences are not relevant: it suffices to verify (P) for a single call attempt. Then, the verification can be completed within Monadic Second Order Logic (and that is what we also did, using MONA [KM98]). Even more: we could use a safe abstraction, considering only two users, two basic services and two CUGs and translate the problem into propositional calculus, since the number of subscription options is then bounded (it depends only on about 20 propositional variables).

3.4 Introducing Explicit Call Transfers

Now, the picture becomes more complicated since an arbitrary long chain of transfers, involving arbitrarily many users may be involved in a communication. Events are still defined as an inductive type whose constructors are the ones mentioned above, plus

- ECT : line $\times$ line $\times$ line $\times$ event sequence $\rightarrow$ event sequence

which takes as arguments three lines. In order for a transfer to be successful, the first two lines must be, at the time of the transfer, connecting a single user

to two others by two connections of same type, and the third line must be free. This is expressed in the model with the following axiom:

$$\left(\begin{array}{ll} & \text{Line_State}(J, ES) = \text{Connecting}(x, y, T) \\ \wedge & \text{Line_State}(K, ES) = \text{Connecting}(x, z, T) \\ \wedge & \text{Line_State}(L, ES) = \text{Free} \end{array}\right)$$

$$\Downarrow$$

$$\left(\begin{array}{ll} & \text{Line_State}(J, \text{ECT}(J, K, L, ES)) = \text{Free} \\ \wedge & \text{Line_State}(K, \text{ECT}(J, K, L, ES)) = \text{Free} \\ \wedge & \text{Line_State}(L, \text{ECT}(J, K, L, ES)) = \text{Connecting}(y, z, T) \end{array}\right)$$

Another axiom states that if the left part of this implication does not hold, then the states of J, K and L must remain the same.

3.5 Introducing Conference Bridges

The previous type is enriched with new constructors corresponding to the start and the release of conferences, and the addition of a new participant to a conference in progress:

- Start_Conference : user $\times$ bridge $\times$ event sequence $\rightarrow$ event sequence
- End_Conference : bridge $\times$ event sequence $\rightarrow$ event sequence
- Add_to_Conference : line $\times$ bridge $\times$ event sequence $\rightarrow$ event sequence

Also, two new lines states are considered, namely "connecting a user to a bridge" and "connecting two bridges". Note that the notion of connection between two users is now more complex to define, as it is not restricted to the existence of a line connecting them. Indeed, a conference controller is (indirectly) connected to each conferee. Because they are connected to the same bridge, conferees are also connected together. When a bridge is connected to another, all users connected to the first one are also connected to the other, etc... More formally, a connection is now the former notion of connection or the transitive symmetric closure of the connection to a bridge.

4 Proofs

4.1 The CUG service alone

First, we proved that if a setup S passes the CUG checks and returns the call type T, then the caller is always allowed to call the callee with the requested basic service, with a call of type T. This result ensures that setups, when they request a call which is forbidden by the general description of the CUG service,

are always rejected by the CUG checks:

$$\mathcal{S} \models (\mathsf{CUG_Check}(S) = \mathsf{accept}(T)) \Rightarrow \mathsf{Allowed_Call} \begin{pmatrix} S.caller, \\ S.basic_service, \\ S.callee, \\ T \end{pmatrix}$$

Next we proved that if x is allowed to call y with s, with a call of type T, then there exists a setup S of caller x, callee y and basic service s, such that this setup passes the CUG checks and returns the call type T. In other words if, according to the general description of the CUG service, a call is allowed, then at least one setup can request this call and pass the CUG checks:

$$\mathcal{S} \models \mathsf{Allowed_Call}(x, s, y, T) \Rightarrow \exists S \begin{pmatrix} & \mathsf{CUG_Check}(S) = \mathsf{accept}(T) \\ \wedge & S.caller = x \\ \wedge & S.basic_service = s \\ \wedge & S.callee = y \end{pmatrix}$$

Together, these results prove that the CUG specifications alone are correct in the sense that there is an adequation between the general description of the service and the protocol implementing it. Furthermore, since a connection always results from a call attempt whose setup passed the CUG checks, the property (P) follows easily from the first theorem and the definition of Connected in this particular case.

4.2 Combination with ECT

The property (P) fails without any additional condition on ECT (following the definition of [KL98], there is an interference between ECT and CUG). For instance when two users belong to the same CUG, but do not share any basic service, they cannot call each other within the group, but there might still exist a third user which could call both of them and connect them with a transfer.

However, according to the designers, this case would be forbidden in practice. So, for the sake of simplicity, let us forget about basic services and take only in account subscriptions to OA, IA, OCaB and ICaB. Consider the following situations:

(1) x is able to call y and z with two normal calls, where

- y and z are CUG users without OA, or
- y and z are CUG users without IA.

(2) x is able to call y and z within c, where

- y and z have both subscribed to OCaB within c, or
- y and z have both subscribed to ICaB within c.

In the first case, though they are not allowed to call each other with a normal call, an ECT performed by x will successfully connect y and z by a normal connection. In the second, though they are not allowed to call each other within c, an ECT performed by x will connect y and z within this group. The next figure describes how situations (1) and (2) can happen in practice.

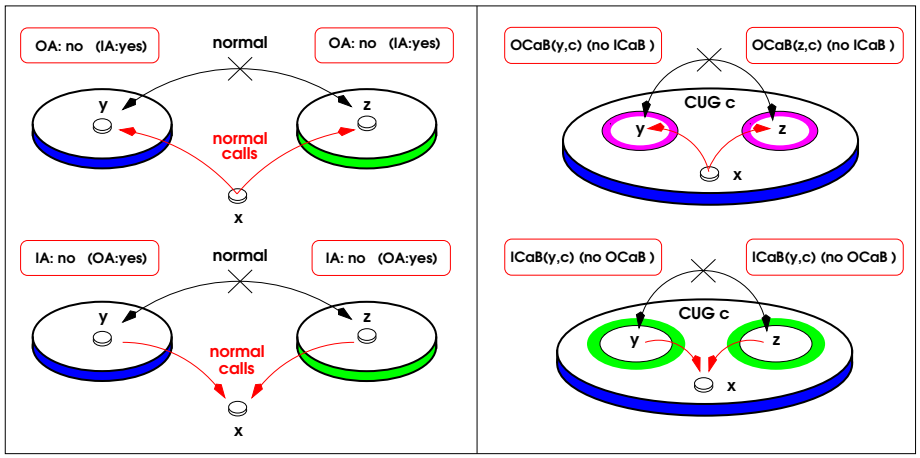

Figure 7: CUG+ECT: All situations where (P) is not preserved

We proved that in all other situations where a user is connected to two others with two calls of same type, one of the latter is always allowed to call the other for this type of call. Therefore, the assumption (denoted as $\mathcal{C}_{CUG+ECT}$ in section 1) that a transfer is always rejected in cases (1) and (2) is necessary and sufficient to preserve the property (P) when CUG and ECT are combined: if we let $\phi(x, y, T, ES)$ be

$$\forall x, y, T, ES. \left(\mathsf{Connected}(x, y, T, ES) \Rightarrow \exists s. \left(\begin{array}{l} \mathsf{Allowed_Call}(x, s, y, T) \\ \vee \quad \mathsf{Allowed_Call}(y, s, x, T) \end{array} \right) \right)$$

Theorem 1 *$\mathcal{S}, \mathcal{C}_{CUG+ECT} \models \forall x, y, T, ES. \phi(x, y, s, T)$ and, for any condition $\mathcal{C}'$ which does not imply $\mathcal{C}_{CUG+ECT}$, we have $\mathcal{S}, \mathcal{C}' \not\models \forall x, y, T, ES. \phi(x, y, s, T)$.*

The first proof requires an induction on the event sequence.

4.3 CUG+ECT+CONF

Despite the fact that the notion of connection between two users is more complex in this case, the situation is actually very close to the previous one. For instance, the fact that a conference controller is able to call two users and add them to the same conference bridge does not ensure that these users are allowed to call

each other. Again, it is possible to define a minimal, necessary and sufficient assumption that guarantees that after any sequence of events, the property (P) is satisfied. First, let us define the relation $\mathsf{Check}(y, z, T)$ by:

- if T is "normal", $\mathsf{Check}(y, z, T)$ holds if and only if
  - y, z are not CUG users without OA, and
  - y, z are not CUG users without IA.
- if T is "within the CUG c", $\mathsf{Check}(y, z, T)$ holds if and only if
  - y and z have not both subscribed to ICaB within c, and
  - y and z have not both subscribed to OCaB within c, and
  - (if this property is not guaranteed) there is at least one basic service to which y and z have both subscribed within c.

Now, suppose that:

- whenever a conference controller x adds a user y to a bridge b of type T, for all z distinct from y and connected to b, $\mathsf{Check}(y, z, T)$ is satisfied.
- whenever a user x performs an ECT between two lines,
  - if the lines are connecting x to y and z with connections of type T, then $\mathsf{Check}(y, z, T)$ is satisfied.
  - if one line connects x to y and the other connects him to a bridge b, the connection and the bridge being of same type T, then for all z distinct from y and connected to b, $\mathsf{Check}(y, z, T)$ is satisfied.
  - if the lines are connecting x to two bridges b and c, of same type T, then for all y connected to b, for all z distinct from y and connected to c, $\mathsf{Check}(y, z, T)$ is satisfied.

(see also figure 8). This defines a condition $\mathcal{C}_{CUG+ECT+CONF}$ As for the combination of CUG and ECT, we proved that this assumption is sufficient, necessary and minimal to preserve the property (P) in the combination of CUG, ECT and CONF.

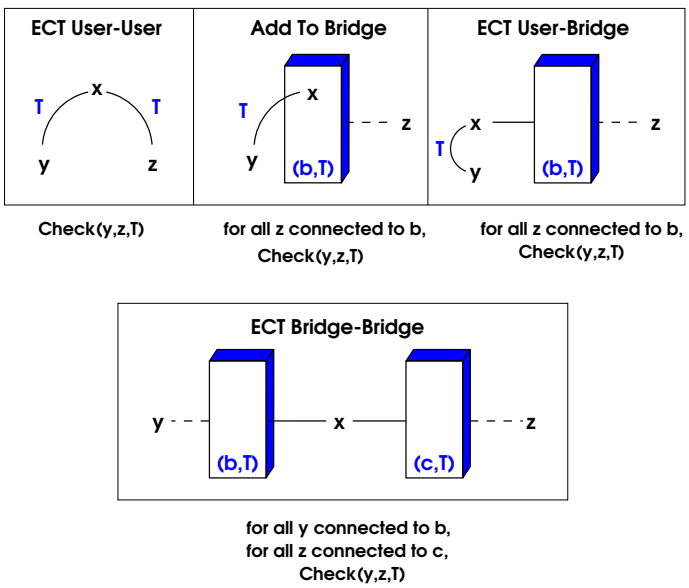

Figure 8: CUG+ECT+CONF: the assumption $\mathcal{C}_{CUG+ECT+CONF}$

Theorem 2 *$\mathcal{S}, \mathcal{C}_{CUG+ECT+CONF} \models \forall x, y, s, T.\phi(x, y, s, T)$ and for any condition $\mathcal{C}'$ which is strictly weaker than $\mathcal{C}_{CUG+ECT+CONF}$, we have $\mathcal{S}, \mathcal{C}' \not\models \forall x, y, s, T.\phi(x, y, s, T)$.*

5 Conclusion

Up to our knowledge, this is the first exhaustive formal study of supplementary service interactions. Of course, we are far from the hundred of services which may interact. More supplementary services would basically result in more predicate symbols and more events, but we believe that we can extend the results to an arbitrary combination of services, as far as only property P is involved.

We have still to point out that this approach cannot be applied to verify some properties which are protocol-dependent. For instance, we cannot check that the *Number Portability Service* works as expected. There are also some expected properties which require to model time, either within first-order logic or using a temporal logic. Formally proving such properties (or, more precisely designing the minimal conditions under which they hold) would require much more efforts. The tool LP may also be not adequate for this purpose.

We did not follow the classification of [CGL+94]: we do not believe that the distinction between customer features and system features is relevant to verification. For instance, we could also apply similar techniques for billing (which is included in system features, whereas our verification is concerned with customer features).

Finally, we point out that interferences were discovered at the specification phase. The proofs only ensure that we indeed discovered all of them.

References

[CGL+94] E. Cameron, N. Griffeth, Y.-J. Lin, M. Nilson, W. Schnure, and H. Velthuijsen. A feature interaction benchmark for in and beyond. In W. Bouma and H. Velthuijsen, editors, *Feature Interactions in Telecommunications Systems*. IOS Press, 1994.

[con88] Multiparty supplementary services: conference calling (conf). ITU-T Recommendation I.254.1, Geneva, 1988.

[cug92] Closed user group. ITU-T Recommendation I.255.1, Geneva, 1992.

[ect97] Call offering supplementary services: Explicit call transfer. ITU-T Recommendation I.252.7, Geneva, 1997.

[GG95] Stephen J. Garland and John V. Guttag. *LP, the Larch Prover, Version 3.1.* MIT Laboratory for Computer Science, January 1995. available as http://www.sds.lcs.mit.edu/spd/larch/LP/overview.html.

[KB98] K. Kimbler and L. Bouma, editors. *Feature Interactions in Telecommunications and software systems V*. IOS Press, 1998.

[KK98] D. Keck and P. Kuehn. The feature and service interaction problem in telecommunications systems: a survey. *IEEE transactions on software engineering*, 24(10):779–796, October 1998.

[KL98] J. Kamoun and L. Logrippo. Goal-oriented feature interaction detection. In K. Kimbler and L. Bouma, editors, *Proc. Feature Interactions in Telecommunications and software systems V*. IOS Press, 1998.

[KM98] Nils Klarlund and Anders Moeller. *MONA version 1.2 User Manual.* BRICS, Department of Computer Science, Univ. of Aarhus, June 1998. available as http://www.brics.dk/mona/manual.html.

[Zav99] Pamela Zave. Systematic design of call-coverage fetaures. In *Algebraic Methodology and Software Technology'98*, volume 1548 of *Lecture Notes in Computer Science*, pages 23–27. Springer-Verlag, 99.