Verification of SDL Specifications on the Basis of Stream Semantics

Ursula Hinkel
Institut für Informatik, Technische Universität München, 80290 München, Germany
Tel: +49 89 289 25352, Fax: +49 89 289 28183, hinkel@in.tum.de

Abstract

This paper presents a new approach to the formal verification of SDL specifications. SDL is given denotational semantics based on the concepts of streams and stream processing functions in the formal framework of FOCUS. The formalization of SDL revealed some aspects of SDL which are handled unprecisely in the Z.100, e.g., the concept of time, and gives a solution to them. The formal semantics is the starting point for a verification method for SDL specifications. Properties of SDL specifications can now be proved in the mathematical, logical framework of domain theory. To document the use of the verification method, we outline the results of a case study in which we proved the correctness of the SDL specification of the well-known alternating bit protocol.

Keywords

SDL, semantics, streams, stream processing functions, verification, time, FOCUS

1. INTRODUCTION

Formal verification allows to mathematically prove that a specification fulfils desired properties. As a prerequisite, formal semantics must be provided so that the meaning of the specification is precisely and clearly defined. Moreover, this semantics has to be formalized in a logical framework which supports formal reasoning about the specification. In our opinion there has not yet been proposed a formal method for the verification of SDL specifications.

Besides its precise syntax, a formal specification language is characterized by its formal semantics given in terms of logical and mathematical formulas and verification rules. In its present form, SDL is not a formal language because a formal semantics is missing. In 1988, a "formal semantics" for SDL was published by the ITU-T (former CCITT) in [10]. It gives the behaviour of an SDL specification by defining its interpretation on an abstract SDL-machine. This machine is specified with Meta IV ([2]) and CSP ([14]). Meta IV as well as CSP have a formal semantics, but for the combination of both, no precise, mathematical foundation is provided. Therefore, this SDL semantics is only informal and cannot be used to handle verification tasks. Moreover, the informal description of SDL given in the Z.100 leaves important aspects like the concept of time and fairness of nondeterministic features unclear. Thus, clear, mathematically founded semantics is urgently needed, not only for the formal verification of SDL, but also for the clarification of the meaning of SDL specifications.

This paper presents a framework for the formal verification of SDL specifications on the basis of stream semantics. We provide denotational semantics for SDL in terms of streams and stream processing functions within the framework of FOCUS. FOCUS is a methodology for the formal specification of distributed systems in the tradition of Kahn [12] based on the concepts of streams and stream processing functions. It has well-defined formal semantics and offers formal proof techniques of domain theory and

classical higher order logic. By transforming an SDL specification into a FOCUS specification, these proof techniques can be used to prove properties of the SDL specification.

Our semantics definition of SDL is an extension of [9] which considered only a restricted subset of SDL and concentrated on the time independent part of SDL. The formal semantics was used for the development of SDL specifications in FOCUS, whereas we use it as the starting point for the formal verification of SDL specifications. Our semantics will cover most features of what is called Basic SDL in [11]. Concerning SDL processes, states, start and stop symbols, input and output of signals with data, tasks, decisions, save symbols, timer mechanism and nondeterministic behaviour modelled by spontaneous inputs and nondeterministic decisions are included. The communication links are delaying and nondelaying channels and signalroutes. For an introduction to SDL we refer to [3] and [16]. Concerning the timing aspects of SDL we argue that the treatment of time in SDL specifications is unprecisely described in [11]. As a consequence, the notion of time is interpreted in different ways by SDL users. However, most of them agree that SDL includes no quantitative assumptions about the timing of the messages; nothing is known about the speed of the processes. We base our semantic definition upon this notion of time.

This paper aims at giving the basic ideas of our approach without going into technical details. For a detailed presentation of the approach we refer to [7]. This paper is structured as follows. In Section 2, the time concept of SDL is analysed. Section 3 introduces the underlying concepts of FOCUS, streams and stream processing functions. In Section 4, we present the denotational semantics of SDL in FOCUS, concentrating on the semantics of SDL processes. The verification method is given in Section 5. Next, in Section 6, we apply our verification method to a case study, the alternating bit protocol. Finally, Section 7 summarizes the results of this paper and discusses some future work.

2. THE CONCEPT OF TIME IN SDL

It is by now widely accepted that SDL is not well suited for the definition of hard real-time requirements. There are a number of papers about SDL which have a critical look at the real time concept in SDL (e.g. [8], [13], [15]) and discuss the expressiveness of real-time properties in SDL. However, a crucial point has not yet been discussed: the relationship between the progress of time and the behaviour of a system specified by SDL. Analysing the behaviour of SDL specifications we had to learn that the time related aspects of SDL are described only vaguely in the recommendations of the ITU-T and interpreted in different ways by SDL users.

When a timer is set in an SDL process, an absolute expiration time is to be given. Yet there is no clear statement in the SDL Recommendation [11] about the way time proceeds in systems described by SDL. Nothing is said about the amount of time which is consumed by actions like assignments in tasks, reading an input signal and outputting signals in an SDL process. In the SDL user guidelines [6] it is said that SDL processes are based on extended finite state machines whose transitions are regarded as taking zero time. However, the next sentence includes the statement that SDL allows for the possibility of non-zero transition time. Concerning the expression NOW by which SDL processes access the system time it is stated that the system time can but need not be different for each task in a transition.

Note that it is not explicitly stated that the time concept is intended to be left open. It is suggested that the progress of time should be related to the implementation of the system. However, the objective of using a formal specification language is to give an abstract model of the system to be developed and to describe its properties without referring to the later implementation. Thus, the SDL time concept in the specification ought to be independent of the later implementation of the system.

When discussing this topic with SDL users we were confronted with two contradictory interpretations of SDL time:

- Transitions are regarded as taking zero time. Time is only allowed to pass when all SDL processes are in an idle state waiting for further input signals, that is all input ports are empty (except for signals to be saved). Then time proceeds until an active timer expires or a signal from the environment is received. Thus, the proceeding of time is connected to the behaviour of the SDL processes. It might happen that time does not increase at all because no halt of the system occurs. This interpretation might be influenced by the way some SDL tools handle the proceeding of time during the simulation of SDL specifications.

- Time is always increasing. Time proceeds during the execution of transitions and in the states.

There exists a global clock which increases time, is accessable to all processes and drives all timers. Thus, time is independent of the behaviour of the SDL processes; time is an orthogonal concept to the system behaviour.

Most users support the second notion of time. To us this interpretation seems to be the more natural and intuitive one of both. Therefore we base the definition of the semantics on the following notion of time: We assume a global clock. Time is always allowed to pass. The execution of SDL processes takes time. However, nothing is known about the speed of the processes and about the amount of time spent in a transition or in a state. Thus, an SDL process behaves nondeterministically with regard to time consumption. Regarding an SDL process as a black box an arbitrary delay between the input of a signal and the output of signals is observed resulting from the transition triggered by the input signal. Furthermore, nothing is known about the time spent by the timer mechanism and about the time a timer signal is in the input queue until it is consumed by the SDL process. As a consequence, it makes no sense to specify quantitative expiration times for SDL timers. It is more reasonable to demand that a timer set by a process will expire after an arbitrary, but finite duration of time and will be put in the input queue of the process.

We cannot claim that the interpretation of time, which now underlies the formalization of SDL in Section 4, conforms to the interpretation of time in the Recommendations Z.100. Yet it is a reasonable and pragmatic view of time corresponding to the way most SDL users interpret SDL time.

3. FOCUS

In the following we introduce the underlying concepts of FOCUS which are relevant to the definition of the semantics in Section 4. The interested reader is referred to details on [4] for an introduction and more formalization.

In FOCUS, systems are modelled as networks of components communicating asynchronously via unbounded, one-way channels. A system is a network consisting of a number of components which either form a network by themselves or are specified as basic components which are not structured any further. The message flow between the components is modelled by timed streams. A timed stream is associated with a channel between two components and contains all information about *what* message is sent *when* between these components. Thereby, a global discrete model of time is used; time proceeds in equidistant time intervals. A special time signal $\surd$ is introduced, called time tick. A tick indicates the end of a time interval in a stream. Apart from the time ticks a stream may contain a finite or infinite number of messages. For instance, the timed stream $ab\surd c\surd\surd a$ contains the stream of messages $abca$. In the first interval, the messages a and b are communicated followed by message c in the second interval; in the third interval, there is no communication and in the fourth interval, message a is communicated.

Formally, a stream is a finite or infinite sequence of messages. Given a set N of messages, $N^{\infty\!\!-}$ denotes the set of all infinite timed streams. In some specifications, it may be sufficient to model the behaviour of a component without regarding the timing aspects. In this case, we can abstract from the timing information in the streams: N^{ω} denotes the set of all finite (N^*) and infinite streams (N^{∞}) without time ticks. If s is a timed stream, $\overline{s}$ denotes the stream without time ticks. $m\&s$ denotes the result of appending the message m on the stream s.

A component communicates with its environment through a set of input and output channels. Thus, the behaviour of a component is described by the relation between its input streams and its output streams. This relation is defined by a stream processing function that maps timed input streams to timed output streams. Stream processing functions have to fulfill semantic properties as continuity and time-guardedness, as explained in literature ([4]). A system of components can be defined either by recursive equations or by special composition operators.

Let S be the specification of a component which inputs messages of type I and outputs messages of type O. With this specification a relation R_S between the input and the output streams of the component is defined. The denotation of the specification S, written $[\![S]\!]$, is a set of stream processing functions:

$$[\![S]\!] =_{\text{def}} \{f_S : I^{\infty\!\!-} \to O^{\infty\!\!-} \mid \forall\, i \in I^{\infty\!\!-} : R_S[i;\ f_S(i)]\}$$

The semantics of a specification is compositional. The behaviour of a system can be deduced from the behaviour of its constituents.

4. FORMAL SEMANTICS

We present formal, denotational semantics for SDL using streams and stream processing functions. The semantics is achieved by specifying the different SDL constructs in FOCUS: an SDL specification is transformed into a FOCUS specification and by this it is provided with a clear denotation, as specifications in FOCUS have formal semantics. We outline the basic concepts of the semantic definition; the complete definition is given in [7].

4.1 Structuring elements of SDL

The structure of the FOCUS specification corresponds in a natural way to that of the SDL specification. A system specified with SDL is modelled by a component of type *System* in FOCUS which gives a black box view of the system. The structure of this component is derived from the SDL block structure. For each SDL block a FOCUS component of type *Block* is introduced. Concerning the communication links it has to be considered that in SDL channels and signalroutes may be bidirectional or unidirectional whereas in FOCUS channels are unidirectional and non-delaying. Therefore, bidirectional SDL channels and signalroutes are mapped to two unidirectional channels in FOCUS. Delaying SDL channels are modelled by components *Delay* which give an arbitrary delay to a signal before it is sent to its receiver. According to the structure of the corresponding SDL block, a component *Block* will be further structured either into components of type *Block* or into components of type *Process*. A component *Process* models the behaviour of the corresponding SDL process (see Section 4.3).

4.2 SDL data

The abstract data types of SDL are formalized in SPECTRUM, an algebraic specification language ([5]) which uses concepts of domain theory as FOCUS does. Thus, it forms an integral part of our formal framework. There is a library of abstract data types in SPECTRUM so that the predefined SDL types can be mapped to the SPECTRUM types.

4.3 SDL processes

An SDL process describes a complex behaviour which includes parts which are only implicitly specified in the SDL process graph, e.g., the input queue and the supervision of timers. In the semantics all behavioural aspects of an SDL process are explicitly handled. In order to give a modular semantics, the behaviour of an SDL process, modelled by the FOCUS component *Process*, will be split into several parts. Following the ideas of [9] an SDL process is modelled by a network of FOCUS components as illustrated in Figure 1:

- The component *Fair Merge* represents the internal, unbounded input queue in which all incoming signals are inserted. It merges the input streams arriving at the signalroutes, the timer signals and the *none*-signals into one stream.
- The component *PR* models the actual behaviour of the SDL process. It carries out the actual processing according to the state machine given by the SDL process graph.
- *Timer Delay* is a component which gives an arbitrary, but finite delay to timers set by the process. Whenever a timer is set during a transition the component *PR* sends a timeout signal with the name of the timer to the *Timer Delay* component. After an arbitrary but finite delay, *Timer Delay* sends this signal to *Fair Merge* which merges the stream of timer signals with the other streams of input signals.
- The component *None* is added in order to handle spontaneous transitions. It nondeterministically outputs a stream of signals called *none* which are merged by *Fair Merge* into the input stream of *PR* and consumed as ordinary input signals by *PR*.
- The component *Split* distributes the output signals of component *PR* to the output signal routes of the SDL process.

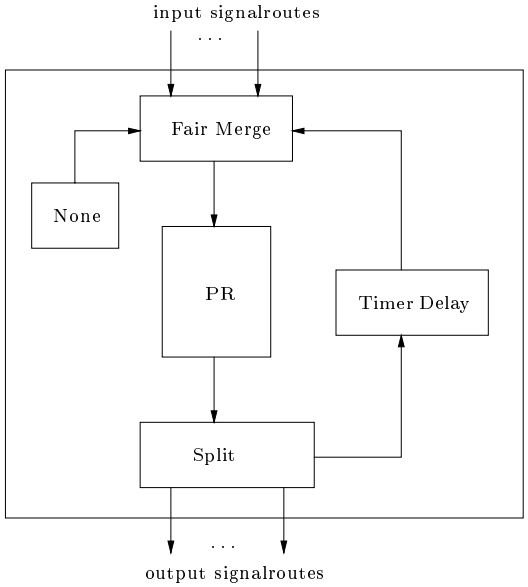


Figure 1: Modelling an SDL process in FOCUS

Except of the component *PR*, the specifications of all components are fixed and overloaded for any number of channels and for any signal sorts. Depending on the process symbols used in the corresponding SDL process the components are included into the network: for a process with timers, *Timer Delay* is included, for a process with spontaneous inputs, *None* is included.

The behaviour of the component *PR* which models the SDL process diagram is individually defined for each SDL process. It is given by a predicate which denotes a set of stream processing functions. Each function describes a possible input/output behaviour of the process. This reflects the nondeterminism inherent in the behaviour of an SDL process: a stream of input signals results in different streams of output signals. This is caused by the nondeterministic behaviour of a process in regard to its timing. The predicate consists of a number of equations on functions which are derived from the transitions of the SDL process graph. For each control state of the SDL process a function is introduced which describes the behaviour of the SDL process in this control state. The functions are named according to the corresponding control states. An equation models all actions performed by the process during a transition: consumption of the input signal, sending of output signals, manipulations of the local variables as well as the setting of timers.

An equation derived from a transition leading from control state S to state S' follows the pattern

$$S\ [d_1, \ldots, d_k, \varphi]\ (a \,\&\, in) = o_1 \,\&\, \ldots \,\&\, o_n \,\&\, S'\ [d'_1, \ldots, d'_k, \varphi']\ (in)$$

The function S receives the stream of input signals which consists of the first signal a followed by the stream in of further input signals. It reacts to the signal a by outputting a stream of signals $o_1, \ldots, o_n$ and calling function S' with the rest of the input stream in. In addition the data state, consisting of the values $d_1, \ldots, d_k$ of the local variables $v_1, \ldots, v_k$ of the process, may have changed during the transition. This is indicated by $d'_1, \ldots, d'_k$. In case that the process uses timers, a parameter φ is added which models the management of timers inside the SDL process.

As explained in Section 2 an SDL process behaves nondeterministically with regard to time consumption and is in fact a time dependent component. Between the consumption of an input signal and its corresponding output signals an arbitrary amount of time elapses. In the specification of *PR* we could model this delay by inserting an arbitrary, finite number of time ticks before each output signal in the stream of output signals. However, this would lead to a rather complex specification. Instead, we use the specification format for *time independent* components: in the specification of *PR*, all time ticks are abstracted away. Time ticks do only occur in the denotation of the specification. Thus, the delay between the input of a signal and its corresponding output signal is implied in the semantics of the specification,

it need not be explicitly modelled on the syntactic level.

For the complete semantics with all details we refer to [7]. To give an idea of the semantic definition we give the specification of *PR* for an SDL process *Transmitter* called *PR_Transmitter*. We use the specification style of ANDL ([19]), a schematized description language for FOCUS specifications which allows to write specifications by hiding the semantic details of FOCUS. *Transmitter* is part of the SDL specification of the alternating bit protocol which we will use in Section 6 as a case study.

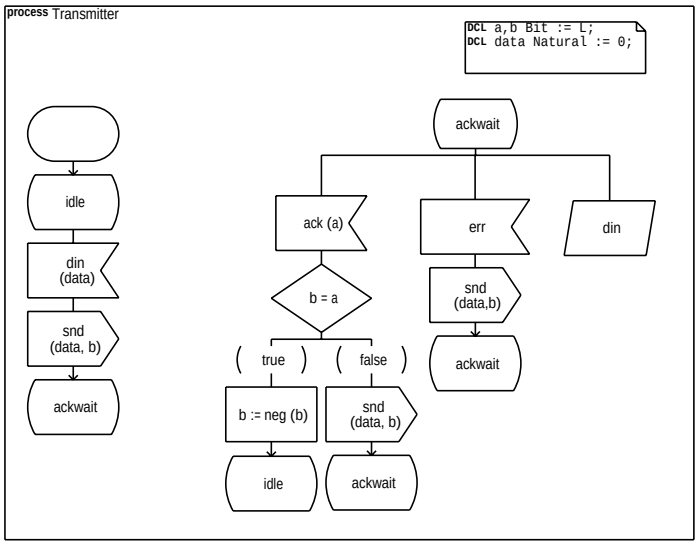


Figure 2: SDL specification of the sender of the alternating bit protocol

component *PR_Transmitter* – **time independent**

input channel $sin : S_{in}$ (* $S_{in} = \{ack, err, din\}$ *input signals* *)
output channel $dsend : In_{med1}$ (* $In_{med1} = \{snd\}$ *output signal* *)

is basic

$\exists\ idle : Nat \times Bit \times S_{in}^{\omega} \rightarrow In_{med1}^{\omega}.$ (* *function for control state* `idle` *)

$idle\ [0,\ L]\ (sin)\ =\ dsend$ (* 0 *and* L *are the initial values of the variables* `data` *and* `b` *)

where $\forall\ s : S_{in}^{\omega}.\ \forall\ a,\ b : Bit.\ \forall\ data,\ d : Nat.$

$idle\ [d,\ b]\ (din\ (data)\ \&\ s) = snd\ (data,\ b)\ \&\ ackwait\ [data,\ b]\ (s)\ \wedge$

$idle\ [d,\ b]\ (ack\ (a)\ \&\ s) = idle\ [d,\ b]\ (s)\ \wedge$ (* *Implicit transition* *)

$idle\ [d,\ b]\ (err\ \&\ s) =\ idle\ [d,\ b]\ (s)\ \wedge$ (* *Implicit transition* *)

$ackwait\ [d,\ b]\ (s) =$ (* *function for control state* `ackwait` *)
(* `search` *returns the first signal* `ack` *or* `err` *of* `s`*; save signals* `din` *are left in* `s`*)
(* `del` *deletes the the first occurrence of* `ack` *or* `err` *within* `s` *)

case $search(\{ack\ (a),\ err\},\ s)$ of
$ack\ (a)$: if $a = b$ (* *decision symbol is modeled by a conditional expression* *)
then $idle\ [d,\ \neg b]\ (del(ack(a), s))$
else $snd\ (d,\ b)\ \&\ ackwait\ [d,\ b]\ (del(ack(a), s))$
endif
err : $snd\ (d,\ b)\ \&\ ackwait\ [d,\ b]\ (del(err, s))$
endcase
end *PR_Transmitter*

Having transformed an SDL specification S into a FOCUS specification, the formal semantics, denoted by $[\![S]\!]$, can be assigned to the SDL specification: It is the set of all stream processing functions which fulfil the input/output behaviour of the SDL system. As the semantics is compositional, a function describing the behaviour of the system is composed of the functions describing the behaviour of the SDL processes.

5. VERIFICATION

The formal semantics is the basis for the verification of SDL specifications. Denotational semantics is based on a rich mathematical theory called domain theory ([20]). Domain theory offers abstract mathematical concepts of complete partial orders, continuous functions and least fixed points as well as various induction principles. In contrast to simulation or model checking, formal verification can deal directly with infinite state spaces. It relies on techniques like structural induction to prove over infinite domains.

For the verification of SDL specifications we propose the following process:

- Starting from an informal requirements specification the SDL specification S is derived. Guidelines for the transition from an informal specification to an SDL specification are given in literature (e.g. [3, 22]).
- Following our definition of the formal semantics in the previous section, the formal semantics $[\![S]\!]$ is derived for the SDL specification by transforming the SDL specification S into a FOCUS specification:
  - The structure of the SDL specification is modelled by a network *System* in FOCUS. For each SDL block and SDL process a component of type *Block* or *Process* is introduced.
  - The data types of SDL are mapped to the abstract data types of SPECTRUM.
  - For each SDL process a network is specified as illustrated in Figure 1. For the specification of *PR*, functional equations are derived from the SDL process graph. The specifications of the other components, e.g., *Fair Merge*, *None* and *Split*, are added with respect to the composition of the component *Process*.

  The denotation of the SDL specification is a set of stream processing functions. Each function describes a possible input/output behaviour of the SDL system. This reflects the nondeterminism inherent in the behaviour of an SDL specification.
- The properties of the system that are required to be true are formulated as logical predicates. Let P be the predicate denoting the desired properties. The theorem to be proved correct states that any possible input/output behaviour of the SDL system has to fulfil the properties which are defined in the predicate P. Let *in* and *out* be the input and output streams of the system of type *In* and *Out*:
$$\forall f \in [\![S]\!].\ \forall in : In^{\infty}.\ \forall out : Out^{\infty}.\ f(in) = out \Longrightarrow P(in, out)$$
- The verification task has to be carried out. It has to be shown that the SDL specification fulfils the properties of the predicate P by using mathematical and logical proof techniques. Here, the proof techniques known from functional logic, domain theory and classical higher order logic can be applied. If the verification task is successful, the SDL specification describes a system with the desired properties. Otherwise, the SDL specification has to be improved and the verification process has to be redone.

In order to ease the task of formalizing and verifying SDL, tool support is indispensable. We have defined the formal semantics in a schematic way, so that the transformation of SDL specifications into FOCUS specifications can be automatically generated. Concerning the formal verification, FOCUS offers an interface to Isabelle ([17]), an interactive theorem prover. Isabelle supports a variety of logics, among them HOLCF, a higher order logic of computable functions ([18]) which supports the notions of domain theory, so that FOCUS specification can be embedded into HOLCF. Again, this transformation can be generated automatically. Upon this basis, SDL specifications can be formalized in HOLCF and be verified mechanically using the proof support of Isabelle. The formalization of the SDL semantics is given in [7].

6. A CASE STUDY: THE ALTERNATING BIT PROTOCOL

The practicability of our verification method for SDL is evaluated by the well-known case study of the alternating bit protocol. This protocol supports a unidirectional flow of messages between a sender and a receiver over an unreliable medium. Each message is either correctly or erroneously transmitted. However, the medium behaves fair: if a message is sufficiently often given to the medium, it will finally be correctly transmitted. Sender and receiver use a single bit to check whether a message has been correctly transmitted by the medium.

First, the SDL specification of the protocol is derived according to the informal description. The structure of the specification is clear and not given here. The SDL system *ABP* consists of a block *ABPTransmitter* with a single process *Transmitter*, of a block *ABPReceiver* with a single process *Receiver* and of two blocks *Medium1* and *Medium2*. The behaviour of the media, however, is not specified by SDL. It seems to be obvious to model the behaviour of a medium by the nondeterministic decision with *any*: the input signal is either transmitted correctly or erroneously. However, there is no statement about the fairness of the evaluation of *any* and therefore, this specification does not guarantee the fairness of the medium. We specify the media directly in FOCUS. In the following we give the specification of *Medium 1* which receives input signals of type *snd (data, bit)* and outputs either the input signal or the error signal *err*. We use an oracle, an infinite sequence of O and L, which decides whether the signal is transmitted correctly or erroneously. As there are infinitly many L in the oracle, the medium behaves fair.

Then, we transform the SDL specification into a FOCUS-specification following the approach introduced in section 4. For the process *transmitter* the semantics has already been given in Section 4.3. We only give the structure of the system *ABP*, omitting the structure of the components of type *Block* and *Process*. The structure of the system *ABP* is specified by a number of equations following the pattern `<output channels> = component_name <input channels>`.

component *ABP*

input channel $data\_in : In_{sys}$
output channel $data\_out : Out_{sys}$

is network

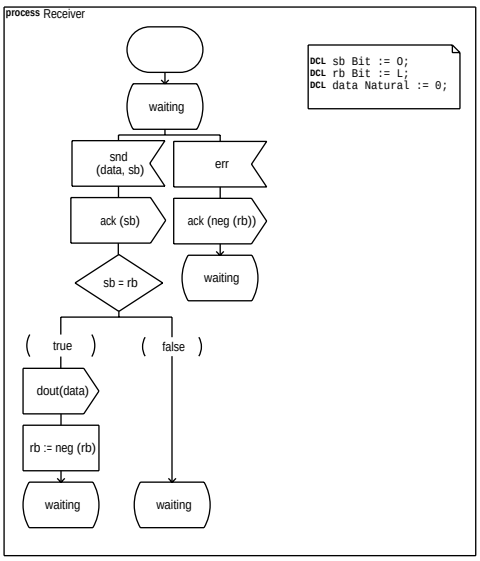
$< dsend > = ABP\_Transmitter < data\_in, arecv >;$

$< data\_out, asend > = ABP\_Receiver < drecv >;$

$< drecv > = Medium1 < dsend >;$

$< arecv > = Medium2 < asend >$

end *ABP*



component *Medium 1* – **time independent**

input channel $dsend : In_{med1}$
output channel $drecv : Out_{med1}$

is basic

$\exists\, o : Bool^{\infty},\ med_1 : Bool^{\infty} \to In^{\omega}_{med1} \to Out^{\omega}_{med1} :$

$\#(\{L\} \copyright o) = \infty\ \wedge\ med_1\ (o,\ dsend) = drecv$

with $\forall\, s : In^{\omega}_{med1},\ m : In_{med1},\ q : Bool^{\infty} :$

$med_1\ (O\ \&\ q,\ m\ \&\ s) = err\ \&\ med_1\ (q,\ s)$

$med_1\ (L\ \&\ q,\ m\ \&\ s) = m\ \&\ med_1\ (q,\ s)$

end Medium1

component *PR_Receiver* – **time independent**

input channel $drecv : Out_{med1}$
output channel $rout : R_{out}$

is basic

$\exists\ \ waiting :\ Bit \times Out^{\omega}_{med1} \to R^{\omega}_{out}.$

$waiting\ [L]\ (drecv)\ =\ rout$

where

$\forall\, s : Out^{\omega}_{med1}.\ \forall\, data : Nat.\ \forall\, rb, sb : Bit.$

$waiting\ [rb]\ (snd\ (data,\ sb)\ \&\ s) =$
if $rb = sb$
then $ack\ (sb)\ \&\ dout\ (data)\ \&\ waiting\ [\neg rb]\ (s)$
else $ack\ (sb)\ \&\ waiting\ [rb]\ (s)$
endif $\wedge$

$waiting\ [rb]\ (err\ \&\ s) =$
$ack\ (\neg rb)\ \&\ waiting\ [rb]\ (s)$

end *PR_Receiver*

We prove that the SDL specification guarantees a correct, reliable transmission of messages via the unreliable medium provided the medium behaves in a fair way. This is stated in the following theorem:

$$\forall f \in [\![ABP]\!].\ \forall in : In_{sys}^{\infty}.\ \forall out : Out_{sys}^{\infty}.\ f(in) = out \Longrightarrow data\_part(\overline{in}) = data\_part(\overline{out})$$

Every function which defines an input/output behaviour of *ABP* behaves like the identity function if the timing aspects are abstracted away and only the data parts of the input signals $din(data)$ and the output signals $dout(data)$ are regarded. The data part of the streams is extracted by the function *data_part*.

The proof is constructed by structural induction on the input stream of the protocol and is based on two lemmata. It reflects the different states which the SDL system takes during the transmission of a data item. Starting from the initial state the system reaches a state in which the first data item is already output to the receiver in the environment, but the sender has not yet received the corresponding control bit. From that state a new start state will be arrived which is identical to the initial state except that the first data item of the input stream has been removed and the bit has alternated. The proof uses properties of the fixed point of the recursive network of streams which is defined by the specification of *ABP*.

The case study shows that the denotational framework in which the semantics is defined is well suited for verification tasks. Further case studies should be done to gain more experience with the verification of SDL specifications, so that more detailed guidelines for verification can be developed.

7. CONCLUSION

In this paper we outlined how SDL specifications can be formally verified using stream based, denotational semantics of SDL. As far as we know this is the first approach to the formal verification of SDL specifications. The proof techniques of classical higher order logic and of domain theory can now be used for the verification of SDL specifications. Besides its use for verification, the formal semantics gives a concise meaning to current SDL, especially to its model of time. The stream semantics integrates all aspects of SDL, like structure, communications, behaviour and data, and is based on an adequate time concept. There are some other approaches to a formal semantics for SDL, e.g. [1], [21], yet no case studies using the semantics for verification tasks are known to us.

Our formalization of SDL revealed a number of unprecisely defined or missing features in the Z.100, for instance the timing concept of SDL or the fairness properties of nondeterministic features. Our survey among SDL users showed that time is interpreted in different ways even by SDL users who know the Recommendation Z.100 very well and have much experience in system development with SDL. We think that this is a situation which is not appropriate for a standardized specification language. Therefore, the features which are now unsufficiently specified in the Z.100 should be improved in the next revision of the SDL Recommendations.

ACKNOWLEDGEMENT

I thank Katharina Spies for helpful comments on a draft version of this paper.

REFERENCES

[1] J.A. Bergstra and C.A. Middelburg. Process algebra semantics of phiSDL. In *ACP '95, Report 95-14*, pages 309–346. Eindhoven University of Technology, Department of Mathematics and Computing Science, April 1995. Invited talk.

[2] Dines Bjørner and Cliff B. Jones. *The Vienna Development Method: The Meta-Language.* LNCS 61. Springer Verlag, 1978.

[3] Rolv Bræk and Oystein Haugen. *Engineering Real Time Systems.* Prentice Hall, 1993.

[4] M. Broy, F. Dederichs, C. Dendorfer, M. Fuchs, T. F. Gritzner, and R. Weber. The Design of Distributed Systems — An Introduction to FOCUS. SFB-Report 342/2/92 A, Technische Universität München, January 1993.

[5] M. Broy, C. Facchi, R. Grosu, R. Hettler, H. Hußmann, D. Nazareth, F. Regensburger, O. Slotosch, and K. Stølen. The Requirement and Design Specification Language SPECTRUM- Part I. Technical report TUM-I9311, Technische Universität München, 1993.

[6] CCITT. *Annex D to Recommendation Z.100, SDL User Guidelines.* ITU, 1989.

[7] Ursula Hinkel. *Formale, semantische Fundierung und eine darauf abgestützte Verifikationsmethode für SDL.* PhD thesis, Technische Universität München, 1998. submitted.

[8] Dieter Hogrefe and Stefan Leue. Specifying Real-Time Requirements for Communication Protocols. Technical Report IAM92-015, University of Berne, 1992.

[9] Eckhardt Holz and Ketil Stølen. An Attempt to Embed a Restricted Version of SDL as a Target Language in FOCUS. In Dieter Hogrefe and Stefan Leue, editors, *Proc. Forte'94*, pages 324–339. Chapman & Hall, 1994.

[10] ITU-T. *Annex F to Recommendation Z.100 (Formal Definition of SDL92).* ITU, 1993.

[11] ITU-T. *Recommendation Z.100, Specification and Description Language (SDL).* 1993.

[12] G. Kahn. The Semantics of a Simple Language for Parallel Programming. *Information Processing*, pages 471 – 475, 1974.

[13] Stefan Leue. Specifying Real-Time Requirements for SDL Specifications - A Temporal Logic-Based Approach. In *Protocol Specification, Testing, and Verification PSTV'95.* Chapman & Hall, 1995.

[14] Robin Milner. *A Calculus of Communicating Systems.* LNCS 92. Springer Verlag, 1980.

[15] Andreas Mitschele-Thiel and Bruno Müller-Clostermann. Extension of SDL and MSC to Support Performance Engineering: A Discussion of Design Issues. Internal report of the Workshop on Performance and Time in SDL and MSC, 1998.

[16] Anders Olsen, Ove Færgemand, Birger Møller-Pedersen, Rick Reed, and J. R. W. Smith. *Systems Engineering Using SDL-92.* Elsevier Science, 1994.

[17] Lawrence C. Paulson. *Isabelle: A Generic Theorem Prover - (LNCS 828).* Springer, 1994.

[18] Franz Regensburger. HOLCF: Higher Order Logic of Computable Functions . In Thomas E. Schubert, Phillip J. Windley, and James Alves-Foss, editors, *Higher Order Logic Theorem Proving and Its Application (HOL95)*, 1995.

[19] Bernhard Schätz and Katharina Spies. Formale Syntax zur logischen Kernsprache der FOCUS-Entwicklungsmethodik. SFB-Report 342/16/95 A, Technische Universität München, Institut für Informatik, 1995.

[20] Dana Scott and C. Gunter. *Handbook of Theoretical Computer Science*, chapter Semantic Domains and Denotational Semantics, pages 633 – 674. Elsevier Science Publisher, 1990.

[21] Jason Steggles and Piotr Kosiuczenko. A Formal Model for SDL Specifications based on Timed Rewriting Logic. In *Second Workshop on Precise Semantics for Software Modeling Techniques*, 1998. Technical report, Technische Universität München.

[22] Kenneth J. Turner, editor. *Using Formal Description Techniques - An Introduction to Estelle, Lotos and SDL.* John Wiley & Sons, 1993.