

DELIVERABLE SUMMARY SHEET

Project Number: *IST-2001-33049*

Project Acronym: **PROTCURE**

Title: *Improving medical protocols by formal methods*

Deliverable N°: 2

Due date: 31/3/2002

Delivery Date: 28/3/2002

Short Description:

Objectives

To formally define the semantics of main elements of the Asbru language.

Procedure

A subset of the rich set of elements of the Asbru language version 7.2 has been selected which covers all the elements which occur in the Asbru models of the selected case studies. The intended semantics of the elements has been discussed. The semantics has been written down using a semiformal, graphical notation similar to statecharts. A formal semantics has been constructed in a Structural Operational Semantics (SOS) style.

Results

The deliverable contains the formal semantics of main Asbru elements. The Asbru elements which were defined mostly cover the elements occurring in the Asbru models of the selected case studies. The semantics is operational and is notated in a Structural Operational Semantics (SOS) style. An intuitive overview of the semantics is given using a graphical statecharts notation. This graphical notation is very good for further discussion of the intended Asbru model behaviour. The SOS rules are very good for designing proof rules to verify Asbru plans.

A shorter version of the paper included in this deliverable has been accepted to the 2002 conference on Integrated Design & Process Technology (IDPT).

Partners owning: all

Partners contributed: Universität Augsburg, Technische Universität Wien, Vrije Universiteit Amsterdam

Made available to: Public

Formal Semantics of Asbru

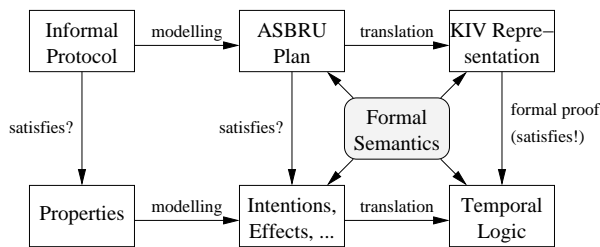
Michael Balser, Christoph Duelli, Wolfgang Reif
Lehrstuhl Softwaretechnik und Programmiersprachen
Universität Augsburg
86135 Augsburg, Germany
{balser,duelli,reif}@informatik.uni-augsburg.de

ABSTRACT: This paper gives part of the formal semantics of a planning language called Asbru which has been specifically designed for the medical framework. A formal semantics is an important step within the Procure project which is concerned with the quality assurance of medical guidelines and protocols. We have constructed a formal semantics in the style of Structural Operational Semantics (SOS). However, this paper is addressed to Asbru users. Therefore, we present sets of SOS rules as semiformal statecharts, which leads to a compact, graphical overview of the operational behaviour. In this style, the semantics documents the language best.

A shorter version of this paper has been accepted to the 2002 conference on Integrated Design & Process Technology (IDPT).

I. INTRODUCTION

This work is part of a European project called Procure [11], which is concerned with the quality assurance of medical protocols. The idea is to model existing informal medical guidelines and protocols in the planning language Asbru [7] [12] and to verify certain properties. Already, Asbru has been used to formalize a variety of examples from different fields of medicine: diabetes mellitus, jaundice in new born babies, artificial ventilation of prematured babies, and others. Other approaches to model medical protocols are e.g. [4] [8] [6]. One of our major goals is to further utilize formal methods in the medical domain by verifying properties of protocols with mathematical rigour in the interactive theorem prover KIV [3], leading to the following overall picture.



Defining a formal semantics of Asbru is an important step within this project. It is the basis for the KIV representation of Asbru plans and the calculus rules. On the other hand it

should also help to understand the Asbru language with all its details.

Our semantics for Asbru is operational and mainly consists of a number of Structural Operational Semantics (SOS) rules [9]. However, the complete set of rules turned out to be lengthy and difficult to understand for Asbru users. Therefore, we present sets of SOS rules in semiformal a statechart notation. The notation does not contain all the technical details of the original rules, but serves as a very good overview of the operational behaviour. The notation has been very useful for further discussions within our heterogeneous group of people from formal methods, medicine, planning, knowledge bases and language design.

The main part of this paper gives statecharts for the most important features of Asbru. In order to show that the behaviour is indeed formally defined, we will also sketch the mapping of the graphical notation to SOS rules.

The paper is organized as follows. In Sect. II we will give a short overview of Asbru followed by notational issues in Sect. III. Section IV gives an overview of the semantics. The hierarchy of plans is explained in Sect. V, which is followed by the basic plan state model of Asbru in Sect. VI. This model is enriched with further important concepts of Asbru in Sections VII to XII. Section XIII gives an outlook on how KIV is used to formally verify properties of Asbru plans and Sect. XIV concludes.

II. ASBRU IN A NUTSHELL

As an example, Fig. 1 displays a simplified version of one of the plans in the jaundice case study. Treating jaundice in newborn babies requires monitoring the level of bilirubin in the blood. Quantitative bilirubin levels are abstracted to qualitative values observation, pt_normal, pt_recommended, pt_intensive, and transfusion. The intention of this treatment plan is to maintain a bilirubin level lower than transfusion and to finally achieve a very low level of bilirubin called observation. The filter condition states that this plan is only applicable, if the bilirubin level is not too high in the beginning. The plan will be aborted, if bilirubin is too high or if it is very high and the decrease within 4 to 6 hours is not large enough. Four different alternative treatments are available. The applicability of these alternatives is determined by their own filter conditions (which are not contained in Fig. 1). For example, plan

```

plan Regular_Treatments_3
intentions
  intermediate-state maintain bilirubin ≠ transfusion
  overall-state achieve bilirubin = observation
conditions
  filter-precondition bilirubin ≠ transfusion
  abort-condition bilirubin = transfusion
    ∨ bilirubin = pt-intensive
    ∧ bilirubin_decrease < 1
      [[4 h, -], [-, 6 h], [-, -], now]
plan-body any-order
  wait-for Observation
  plan-activation Phototherapy_Intensive
  plan-activation Phototherapy_Normal_Prescription
  plan-activation Phototherapy_Normal_Recommended
  plan-activation Observation

```

Fig. 1. Example Asbru plan from Jaundice case study

Phototherapy_Intensive can only be used, if bilirubin level is pt_intensive. Because of the "wait-for" construct, the successful completion of plan Observation is mandatory, other plans are optional.

Asbru is a plan oriented language. Several plans are organized in a hierarchy of plans. A parent plan can refer to other sub plans in its plan body. Conditions are used to control the applicability of a plan and to monitor its execution. Conditions can be monitored over time according to so called time annotations. The sub plans in the plan body can be organized using different body types (e.g. any-order). The current state of a plan – especially if a plan has been rejected, aborted, or completed – is propagated according to the plan hierarchy to its parent and sub plans. If a plan is mandatory, it must be completed, otherwise it may also be rejected or aborted.

III. NOTATION

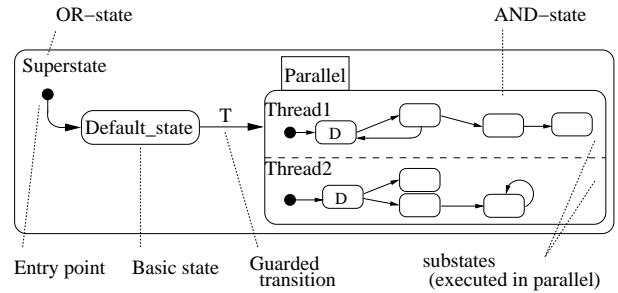
A. EBNF

We will use an EBNF-like notation to describe the syntax of constructs of Asbru. Terminal symbols are written in normal style, names of the grammar rules are typeset in *italic*. Square brackets [·] denote optional parts, and alternatives are written as (· | ·). Zero or more repetitions are denoted as · *.

B. Statecharts

A statechart is a directed graph representing a state machine (a nondeterministic automaton). It is used to specify a system's dynamic behaviour. In this paper we will adopt the syntax of STATEMATE [10]. We will explain its basic features and semantics on the basis of Figure 2.

States are depicted as rounded rectangles. *Superstate* contains two substates. As the system can be only in one



$$T : event[condition]/action$$

Fig. 2. Statechart notation

of these at a given time, *Superstate* is called an OR-state. When the system enters *Superstate*, it's initially in both *Superstate* itself and its substate *Default_state*. The default substate is marked with an arc pointing from a black bullet to the state.

Possible state transitions are represented as directed arcs that may be labelled with guards of the form *event[condition]/action*. A guarded transition can only be taken when *event* occurs and *condition* holds at the same time. Note that an enabled transition *must* be taken (non-deterministic choice if several transitions from a state are enabled).

Parallel is an AND-State. Its substates *Thread1* and *Thread2* – separated by a dashed line – are executed synchronously in parallel. Once transition *T* is enabled, *action* will be executed and then *Parallel* will become active, changing the system's active states to *Superstate*, *Parallel*, *Thread1*, *Thread2*, *Thread1.D* and *Thread2.D*. By means of composite AND- and OR-states, we can create a *state hierarchy*, thus facilitating the readability of the statechart.

C. SOS rules

The formal semantics is given in SOS rules. Our rules will be of the following form.

$$\frac{\llbracket \varphi \rrbracket_{\sigma} \quad g(\sigma) \rightarrow g(\sigma') \quad h(\sigma) \nrightarrow \dots}{f(\sigma) \rightarrow f(\sigma')}$$

A configuration $f(\sigma)$ may step to a configuration $f(\sigma')$, if formula φ holds in valuation σ (a positive premise of style $\llbracket \cdot \rrbracket_{\sigma}$), configuration $g(\sigma)$ is able to step to $g(\sigma')$ (a positive premise of style $c \rightarrow c'$) and $h(\sigma)$ is currently not able to take a step (a negative premise of style $c \nrightarrow$). Valuation σ assigns values to variables.

Details on this notation can e.g. be found in [1].

IV. SEMANTICS OVERVIEW

Plans may refer to sub plans in their plan body leading to a hierarchy of plans as described in Sect. V. The behaviour of a single plan is defined in the so called plan state



Fig. 3. Plan hierarchy

model: conditions are used to control selection and execution of plans. This is explained in Sect. VI. The relationship between parent and sub plans is encoded in events which synchronize the execution of sub plans (see Sect. VII), and the concept of propagation (see Sect. VIII). The definition of mandatory and optional sub plans is discussed in Sect. IX. Timeouts can be defined for the execution of plans (see Sect. X). The special case of cyclical execution of sub plans is explained in Sect. XI. Conditions are evaluated by an underlying data abstraction unit (see Sect. XII). The abstraction unit also takes care of monitoring data over a longer period of time as defined by time annotations in conditions. In our semantics intentions describe properties of plans and can be used as proof obligations for verification (see Sect. XIII).

So far, we only consider part of Asbru version 7.2 (as described in [12]) within this paper. However, we claim that the major concepts of Asbru are covered. Concepts which are neglected, include

- local variables and return values,
- context of parameters,
- suspending and reactivating plans,
- setup preconditions
- more complex cyclical plan execution.

Either these topics are well understood (e.g. local variables) or they are not used in our case studies (e.g. parameter context, suspending plans and setup preconditions). The formal semantics of cyclical plans is still work in progress.

V. PLAN HIERARCHY

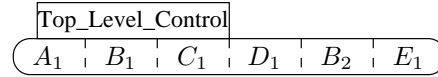
In Asbru, plans are organized in a hierarchy as shown on the left of Figure 3: a parent plan A refers to a number of sub plans B , C , and D in its plan body. Sub plans may refer to further plans resulting in a tree hierarchy. The plan name is used to reference a plan.

One and the same plan may occur several times within this hierarchy (e.g. plan B). Therefore we distinguish between plan references and plan instances. Each reference corresponds to a unique instance. On the right of Fig. 3, plan references have been numbered to give unique instances. The first occurrence of plan B is instance B_1 , the second is instance B_2 .

A. Semantics overview

In our semantics, all existing instances of plans are executed in parallel. The hierarchy of instances is flattened,

leading to one top level control. For the situation in Fig. 3, we denote this top level control with the following state-chart.



In this paper, we consider the list of plan instances fixed, i.e. no instances are created or discarded during execution. We will refer to the list of instances as $[P_1, \dots, P_n]$.

B. SOS rules

We use the configuration $\text{tlc}(\sigma)$ to define the semantics of the top level control. A single step of the top level control must adhere to the relation

$$\text{tlc}(\sigma) \rightarrow \text{tlc}(\sigma')$$

with σ and σ' being the states before and after execution.

The top level control executes all plan instances P_1, \dots, P_n synchronously. In one top level step, a step of every plan is executed. The semantics of a step of plan P_i is described by the following relation (also see Sect. VI).

$$\text{psm}_{P_i}(\sigma) \rightarrow \text{psm}_{P_i}(\sigma')$$

Not all plans may be able to take a step in the current state, i.e.

$$\text{psm}_{P_i}(\sigma) \nrightarrow$$

holds. Therefore we divide the list of plans into two lists P_{m_1}, \dots, P_{m_l} with plans which are able to progress, and $P_{m_{l+1}}, \dots, P_{m_n}$ which are currently blocked. If $1 \leq l$ at least some of the plans are currently able to progress. In this case a single step of the top level control is defined by the following SOS rule.

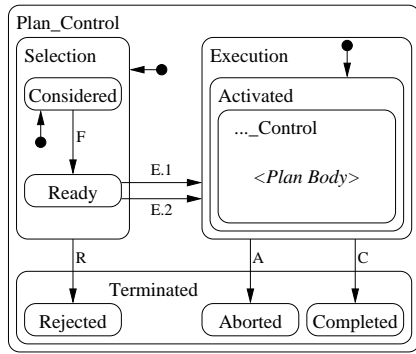
$$\frac{\begin{array}{l} \text{psm}_{P_{m_1}}(\sigma) \rightarrow \text{psm}_{P_{m_1}}(\sigma'_1) \\ \dots \\ \text{psm}_{P_{m_l}}(\sigma) \rightarrow \text{psm}_{P_{m_l}}(\sigma'_l) \\ \text{psm}_{P_{m_{l+1}}}(\sigma) \nrightarrow \\ \dots \\ \text{psm}_{P_{m_n}}(\sigma) \nrightarrow \end{array}}{\text{tlc}(\sigma) \rightarrow \text{tlc}(\sigma'_1 \cup \dots \cup \sigma'_l)}$$

(We omit details on how result states $\sigma'_1, \dots, \sigma'_l$ are united. It is sufficient to know, that each state variable is written by one plan instance only. Other plans may only read variables.)

If all plans are blocked, an environmental step is taken which is described by the following rule

$$\frac{\text{psm}_{P_1}(\sigma) \nrightarrow \dots \text{psm}_{P_n}(\sigma) \nrightarrow \text{env}(\sigma) \rightarrow \text{env}(\sigma')}{\text{tlc}(\sigma) \rightarrow \text{tlc}(\text{reset}(\sigma'))}$$

with relation $\text{env}(\sigma) \rightarrow \text{env}(\sigma')$ describing the nondeterministic update of external variables. Function reset resets all boolean variables corresponding to internal events to *false*.



- F : $[filter_tp = true]$
 R : $[filter_tp = false]$
 $E.1$: $[activate_mode \equiv automatic]$
 $E.2$: $select[activate_mode \equiv manual]$
 A : $[abort_tp = true]$
 C : $[complete_tp = true]$

Fig. 4. Semantics of plan state model

VI. PLAN STATE MODEL

The overall plan state model defines the semantics of the different conditions of a plan. Conditions are used to decide if the plan body is applicable (selection phase) and while executing the body, if execution should be interrupted (execution phase).

A. Syntax

The syntax of a plan is as follows.

```

plan = plan name
      [intentions]
      [conditions]
      [filter-precondition temporal-pattern]
      [activate-mode (automatic|manual)]
      [complete-condition temporal-pattern]
      [abort-condition temporal-pattern]]
      [plan-body]

```

A plan consists of intentions (see Sect. XIII), the definition of conditions (see below), and the plan body (see Sect. VII).

B. Semantics overview

A variation of the standard plan state model described in [7] is given in Fig. 4 to define the semantics of conditions. The *Plan_Control* is divided into the selection phase *Selection* and the execution phase *Execution*. Initially a plan is *Considered*. In this state, the filter condition *filter_tp* is checked. If the condition evaluates to *true*, control advances to state *Ready* (transition F). If activation is automatic, the plan is activated at once (transition E.1). If it is manual, an external event *select* is a prerequisite for activation (transition E.2). This event is controlled by the environment. If, during the selection phase, the filter condi-

tion evaluates to *false*, the plan is *Rejected* (transition R). In state *Activated*, the sub plans of the current plan are executed. This is described in Sect. VII. The execution of sub plans can be either completed successfully (transition C) or aborted in the case of emergency patient readings (transition A). We can refer to *Terminated*, if the reason for termination – rejection, completion, or abortion – is irrelevant.

C. SOS rules

In principle, each transition of Fig. 4 corresponds to one SOS rule. The current state of a plan P is stored in variables $P.state$ in σ , and also the external event *select* corresponds to variables $P.select$.

As an example, transition F originates from the following SOS rule:

$$\frac{[[P.state = Considered]]_{\sigma} \quad da(\Phi, \sigma) \rightarrow^* da(true, \sigma')}{psm_P(\sigma) \rightarrow psm_P(\sigma' [P.state/Ready])}$$

if $P.filter_precondition \equiv \Phi$. If P is in state *Considered* and the filter precondition Φ is evaluated to *true* by the data abstraction unit (see Sect. XII), plan P may progress to state *Ready*. Similar rules for the other transitions are defined.

Additionally, if P is currently active and the complete and abort conditions Φ and Ψ do not hold, we will execute the body of P (see Sect. VII). This is described by the following rule.

$$\frac{[[P.state \in Activated]]_{\sigma} \quad da(\neg \Phi, \sigma) \rightarrow^* da(true, \sigma') \quad da(\neg \Psi, \sigma') \rightarrow^* da(true, \sigma'') \quad bdy_P(\sigma'') \rightarrow bdy_P(\sigma''')}{psm_P(\sigma) \rightarrow psm_P(\sigma''')}$$

VII. PLAN BODY

The hierarchy of plan instances has been flattened. How the parent plan controls the plan instances in its plan body will be explained next.

A. Syntax

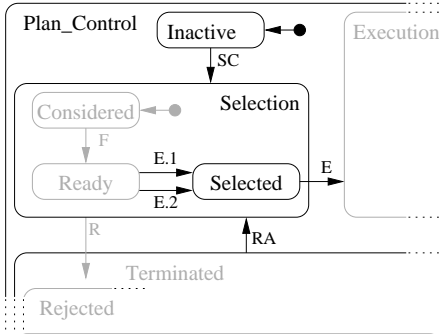
The syntax of the plan body is as follows.

```

plan-body
= plan-body ( sequential|parallel|any-order|unordered
              | on abort|cyclical)
  [wait-for-optional] [retry-aborted]
wait-for
(plan-activation name [time-annotation])*

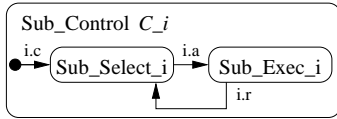
```

The type of the body is either sequential, parallel, any order, unordered, or "on abort". Additionally the body can be cyclical (see Sect. XI). With option "retry-aborted" aborted sub plans will be retried. The "wait-for" construct defines mandatory and optional plans (see Sect. IX – here also the option "wait-for-optional" is explained), and the names of the sub plans are listed as "plan-activations" with an optional time annotation (see Sect. X).



SC : *consider*
 E : *activate*
 RA : *retry*

Fig. 5. Synchronization states in plan state model



$i.c$: $/C_i.consider$
 $i.a$: $[in(C_i.Selected)]/C_i.activate$
 $i.r$: $[retry-aborted \equiv yes \wedge in(C_i.Aborted)]/C_i.retry$

Fig. 6. Controller for executing a sub plan with no restrictions

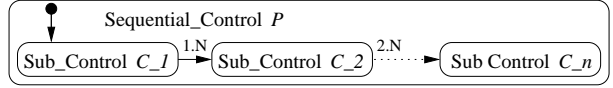
B. Semantics overview

Sub plans C_1, \dots, C_n are controlled in the body of a plan P . Their execution can be organized differently: they can be executed *sequentially* starting with C_1 , they can be executed in parallel either with synchronization (*parallel*) or without synchronization (*unordered*) of the selection and execution phases, and finally they can be executed sequentially, but *any order*, i.e. only one sub plan is executed at once, but the sequence is not fixed.¹ Additionally, aborted plans can be retried.

In order to allow synchronization of the selection and execution phases of the sub plans, and the retrial of plans, the plan state model has to be enriched with intermediate states *Inactive* and *Selected*, and additional transitions SC , E , RA , resulting in the adapted statechart of Fig. 5. The additional events *consider*, *activate*, and *retry* are used to externally control progress of a plan. A parent plan can thus synchronize the sub plans in its plan body. For this, the *Activated* state of the parent is refined with a controlling statechart.

If no restriction on the progress of sub plan C_i is required, the controlling statechart $Sub_Control_i$ C_i of Fig. 6 can be used. Sub plan C_i is considered immediately (transition $i.c$) and is activated as soon as it reaches state *Selected*

¹Cyclical execution will be explained in Sect. XI



$i.N$: $[retry-aborted \equiv no \wedge in(C_i.Terminated)]$
 $\vee retry-aborted \equiv yes \wedge in(C_i.Completed)]$
 $\vee retry-aborted \equiv yes \wedge in(C_i.Rejected)]$

Fig. 7. Controller for sequential execution

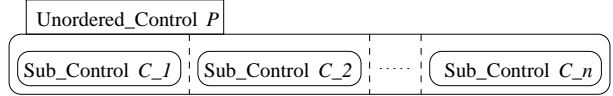


Fig. 8. Controller for unordered execution

(transition $i.a$). If option "retry-aborted" is chosen, then the sub plan is retried, if it aborts (transition $i.r$).

The different body types may oppose restrictions on the execution of sub plans. This is done by deferring the generation of the newly added events. Controlling statecharts for the different body types are explained next.

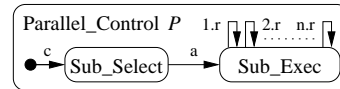
B.1 Sequential execution

The controller of Fig. 7 considers the first sub plan. As soon as it terminates, we continue with the second plan (transition $1.N$). During execution of one plan, no synchronization is required. Thus, we use $Sub_Control C_i$ to execute each sub plan. In the case of "retry-aborted", a sub plan is considered to terminate, if it is either completed or rejected. If it aborts, the controller $Sub_Control C_i$ will take care of retrying the plan immediately (see Fig. 6).

B.2 Unordered execution

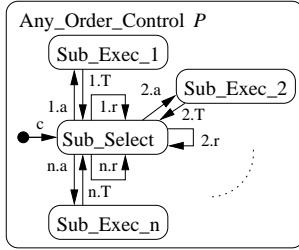
The controller in Fig. 8 executes the sub plans in parallel and no further synchronization is necessary.

Implicitly, if the "activate-mode" of sub plans is manual, arbitrary execution orders (sequential, parallel, etc.) are possible, because the *select* event is controlled by the environment and can be deferred arbitrarily long (see Sect. VI).



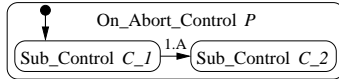
c : $/C_1.consider; \dots; C_n.consider$
 a : $[\bigwedge_{i=1}^n in(C_i.Selected)]$
 $/ C_1.activate; \dots; C_n.activate$

Fig. 9. Controller for parallel execution



$$i.T : [\text{in}(C_i.Terminated)]$$

Fig. 10. Controller for any order execution



$$i.A : [\text{in}(C_i.Aborted)]$$

Fig. 11. Controller for on abort execution

B.3 Parallel execution

The parallel operator (see Fig. 9) synchronizes selection and execution phases of all sub plans. The sub plans are considered immediately (transition c). They may only proceed to *Activated* state, if all sub plans are *Selected* (transition a).²

B.4 Any order execution

In Fig. 10 only the selection phases are executed in parallel. The execution phases of the sub plans are synchronized such that at most one sub plan is active at the same time. For this the plans are considered immediately (transition c). The first plan to become selectable is activated (transition $i.a$). Only if this plan terminates (transition $i.T$) another one can be activated. If several sub plans reach state *Selected* simultaneously, the choice is nondeterministic. If option "retry-aborted" is chosen, then transition $i.r$ is used to initiate a retrial.

B.5 On abort execution

This type of body, executes sub plan C_1 using standard control. If the plan aborts, then C_2 is executed (see transition $i.A$ in Fig. 11). Option "retry-aborted" only affects execution of C_2 . If C_1 aborts, then transition $1.A$ overrides transition $1.r$ of controller $Sub_Control C_1$ (see Fig. 6) which would have initiated a retrial of C_1 .

C. SOS rules

Again, the transitions correspond to SOS rules. Examples are given below. The additional events are stored for

²As will be explained in Sect. IX, it is sufficient that all mandatory sub plans are *Selected* in order to proceed to *Activated* state.

each plan P as variables $P.consider$ and $P.activate$ in σ .

C.1 Standard control

Transition $i.c$ in $Sub_Control C_i$:

$$\frac{[\![C_i.state = \text{Init}]\!]_{\sigma}}{\text{std}_P^{C_i}(\sigma) \rightarrow \text{std}_P^{C_i}(\sigma[C_i.consider/true])}$$

Transition $i.a$ in $Sub_Control C_i$:

$$\frac{[\![C_i.state = \text{Selected}]\!]_{\sigma}}{\text{sub}_P^{C_i}(\sigma) \rightarrow \text{sub}_P^{C_i}(\sigma[C_i.activate/true])}$$

Transition $i.r$ in $Sub_Control C_i$:

$$\frac{[\![C_i.state = \text{aborted}]\!]_{\sigma}}{\text{sub}_P^{C_i}(\sigma) \rightarrow \text{sub}_P^{C_i}(\sigma[C_i.retry/true])}$$

if $P.retry\text{-aborted} \equiv \text{yes}$.

C.2 Sequential execution

The intial transition maps to

$$\frac{[\![P.state = \text{Sub_Init}]\!]_{\sigma}}{\text{seq}_P(\sigma) \rightarrow \text{seq}_P(\sigma[P.state/\text{Sub_Exec}_1])}$$

Transition $i.N$ in $Sequential_Control P$ gives the following rules:

$$\frac{[\![P.state = \text{Sub_Exec}_i]\!]_{\sigma} \quad [\![C_i.state \in \text{Terminated}]\!]_{\sigma}}{\text{seq}_P(\sigma) \rightarrow \text{seq}_P(\sigma[P.state/\text{Sub_Exec}_{i+1}])}$$

for $1 \leq i < n$, if $P.retry\text{-aborted} \equiv \text{no}$.

$$\frac{[\![P.state = \text{Sub_Exec}_i]\!]_{\sigma} \quad [\![C_i.state \in \text{Completed}]\!]_{\sigma} \quad \vee \quad [\![C_i.state \in \text{Rejected}]\!]_{\sigma}}{\text{seq}_P(\sigma) \rightarrow \text{seq}_P(\sigma[P.state/\text{Sub_Exec}_{i+1}])}$$

for $1 \leq i < n$, if $P.retry\text{-aborted} \equiv \text{yes}$.

Sequential execution uses standard control to execute each sub plan. Therefore additional rules are necessary to embed standard control:

$$\frac{[\![P.state = \text{Sub_Exec}_i]\!]_{\sigma} \quad [\![C_i.state \notin \text{Terminated}]\!]_{\sigma}}{\text{std}_P^{C_i}(\sigma) \rightarrow \text{std}_P^{C_i}(\sigma')} \quad \text{seq}_P(\sigma) \rightarrow \text{seq}_P(\sigma')$$

for $1 \leq i \leq n$, if $P.retry\text{-aborted} \equiv \text{no}$.

$$\frac{[\![P.state = \text{Sub_Exec}_i]\!]_{\sigma} \quad [\![C_i.state \notin \text{Completed}]\!]_{\sigma} \quad \wedge \quad [\![C_i.state \notin \text{Rejected}]\!]_{\sigma}}{\text{std}_P^{C_i}(\sigma) \rightarrow \text{std}_P^{C_i}(\sigma')} \quad \text{seq}_P(\sigma) \rightarrow \text{seq}_P(\sigma')$$

for $1 \leq i \leq n$, if $P.retry\text{-aborted} \equiv \text{yes}$.

C.3 Unordered execution

For unordered execution all enabled sub plans take a step simultaneously (compare to tlc in Sect. IV).

$$\frac{\begin{array}{l} \text{std}_P^{C_{m_1}}(\sigma) \rightarrow \text{std}_P^{C_{m_1}}(\sigma'_1) \\ \dots \\ \text{std}_P^{C_{m_l}}(\sigma) \rightarrow \text{std}_P^{C_{m_l}}(\sigma'_l) \\ \text{std}_P^{C_{m_{l+1}}}(\sigma) \rightarrow \\ \dots \\ \text{std}_P^{C_{m_n}}(\sigma) \rightarrow \end{array}}{\text{uno}_P(\sigma) \rightarrow \text{uno}_P(\sigma'_1 \cup \dots \cup \sigma'_l)}$$

where $1 \leq l \leq n$.

C.4 Parallel execution

Transition *c* of *Parallel_Control P*:

$$\frac{\llbracket P.state = \text{Sub_Init} \rrbracket_\sigma}{\text{par}_P(\sigma) \rightarrow \text{par}_P(\sigma [\text{P.state}/\text{Sub_Select}] [C_i.consider/true] \dots [C_n.consider/true])}$$

Transition *a* of *Parallel_Control P*:

$$\frac{\begin{array}{l} \llbracket P.state = \text{Sub_Select} \rrbracket_\sigma \\ \llbracket C_1.state = \text{Selected} \rrbracket_\sigma \\ \dots \\ \llbracket C_n.state = \text{Selected} \rrbracket_\sigma \end{array}}{\text{par}_P(\sigma) \rightarrow \text{par}_P(\sigma [\text{P.state}/\text{Sub_Exec}] [C_1.activate/true] \dots [C_n.activate/true])}$$

Transition *i.r* of *Parallel_Control P*:

$$\frac{\llbracket P.state = \text{Sub_Exec} \rrbracket_\sigma \quad \llbracket C_i.state = \text{Aborted} \rrbracket_\sigma}{\text{par}_P(\sigma) \rightarrow \text{par}_P(\sigma [C_i.retry/true])}$$

for $1 \leq i \leq n$, if $P.retry\text{-aborted} \equiv \text{yes}$.

C.5 Any order execution

Transition *c* of *Any_Order_Control P*:

$$\frac{\llbracket P.state = \text{Sub_Init} \rrbracket_\sigma}{\text{any}_P(\sigma) \rightarrow \text{any}_P(\sigma [\text{P.state}/\text{Sub_Select}] [C_1.consider/true] \dots [C_n.consider/true])}$$

Transition *i.a* of *Any_Order_Control P*:

$$\frac{\llbracket P.state = \text{Sub_Select} \rrbracket_\sigma \quad \llbracket C_i.state = \text{Selected} \rrbracket_\sigma}{\text{any}_P(\sigma) \rightarrow \text{any}_P(\sigma [\text{P.state}/\text{Sub_Exec}] [C_i.activate/true])}$$

for $1 \leq i \leq n$.

Transition *i.T* of *Any_Order_Control P*:

$$\frac{\llbracket P.state = \text{Sub_Exec}_i \rrbracket_\sigma \quad \llbracket C_i.state \in \text{Terminated} \rrbracket_\sigma}{\text{any}_P(\sigma) \rightarrow \text{any}_P(\sigma [\text{P.state}/\text{Sub_Select}])}$$

for $1 \leq i \leq n$.

Transition *i.r* of *Any_Order_Control P*:

$$\frac{\llbracket P.state = \text{Sub_Select} \rrbracket_\sigma \quad \llbracket C_i.state = \text{Aborted} \rrbracket_\sigma}{\text{any}_P(\sigma) \rightarrow \text{any}_P(\sigma [C_i.retry/true])}$$

for $1 \leq i \leq n$, if $P.retry\text{-aborted} \equiv \text{yes}$.

D. Body

The semantics of the overall body of P depends on its type. If $\text{bodytype}(P) \equiv \text{sequential}$, we receive the additional rule

$$\frac{\text{seq}_P(\sigma) \rightarrow \text{seq}_P(\sigma')}{\text{bdy}_P(\sigma) \rightarrow \text{bdy}_P(\sigma')}$$

and similar for the other types.

E. Open issues

- ("on abort" vs. "on reject") Does "on abort" also mean "on reject"?
- (any order and "retry-aborted") Currently either a sub plan is retried or another plan is activated. Thus, if a sub plan is activated, the retrial of other plans is deferred. The semantics needs to be corrected here.

VIII. PROPAGATION

The hierarchy of plans has been flattened. The parent is able to control and synchronize progress of its sub plans as described in the previous section. Nevertheless additional control to propagate execution states of a sub plan to its parent and vice versa is necessary. For example, if a (mandatory) sub plan C_i aborts, then also the parent aborts. This is known as propagation in Asbru. There are a number of dependencies between sub plans and parent similar to this example. All of them are displayed as additional or refined transitions in Fig. 12. SOS rules for the added and refined transitions are straightforward.

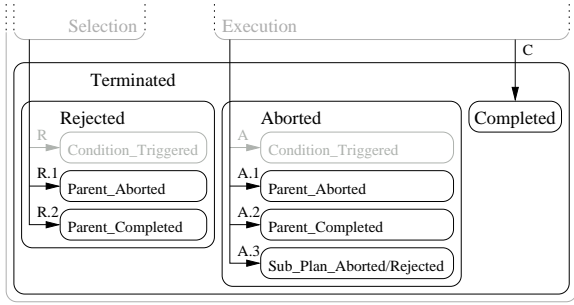
IX. CONTINUATION SPECIFICATION

Some of the sub plans are mandatory for the successful execution of the parent plan, others are optional. The "wait-for" construct determines, which or how many of the sub plans the parent requires to complete successfully.

A. Syntax

$$\text{wait-for} \\ = (\text{abstract-wait-for} | \text{all} | \text{one} | \text{number} | \text{none})$$

The parent either requires all, one, a fixed number or none of its sub plans to complete successfully. Alternatively, a



$$\begin{aligned}
 R.1 & : [\text{in}(P.Aborted)] \\
 R.2 & : [\text{in}(P.Completed)] \\
 A.1 & : [\text{in}(P.Aborted)] \\
 A.2 & : [\text{in}(P.Completed)] \\
 A.3 & : [\bigvee_{i=1}^n (\text{in}(C_i.Rejected)) \\
 & \quad \vee \text{in}(C_i.Aborted)] \\
 C & : [\text{complete}_{tp} = \dots \\
 & \quad \wedge \bigwedge_{i=1}^n \text{in}(C_i.Completed)]
 \end{aligned}$$

Fig. 12. Semantics of propagation

logical expression *abstract-wait-for* can be used to specify the set of plans to complete successfully. The syntax for the logical expression is as follows.

```

abstract-wait-for
= name
| not abstract-wait-for
| abstract-wait-for (and|or|xor) abstract-wait-for

```

We will also define the option "wait-for-optional" of the plan body (see Sect. VII) in this section.

B. Semantics Overview

Let C_1, \dots, C_n be the sub plans of plan P . The "wait-for" construct of P is transformed into a formula with variables v_1, \dots, v_n . For example the construct

$$\text{wait-for } (C_1 \text{ and } C_2) \text{ or } C_3$$

(which either requires plans C_1 and C_2 or C_3 to complete) is transformed into the following formula.

$$(v_1 \wedge v_2) \vee v_3$$

Within this formula, the variables can be replaced by boolean conditions concerning the current state of each sub plan. For our example, we can determine, if enough sub plans have completed already, by replacing the variables v_i with the conditions $\text{in}(C_i.Completed)$ leading to the expression

$$\begin{aligned}
 & (\text{in}(C_1.Completed) \wedge \text{in}(C_2.Completed)) \\
 & \vee \text{in}(C_3.Completed)
 \end{aligned}$$

If and only if this expression evaluates to *true*, enough sub plans have completed.

Similarly we can determine, if too many sub plans have been rejected or aborted, such that the parent cannot complete successfully any more. By replacing variables v_i with conditions

$$\neg (\text{in}(C_i.Rejected) \vee \text{in}(C_i.Aborted))$$

we receive

$$\begin{aligned}
 & \neg (\text{in}(C_1.Rejected) \vee \text{in}(C_1.Aborted)) \\
 & \wedge \neg (\text{in}(C_2.Rejected) \vee \text{in}(C_2.Aborted)) \\
 & \vee \neg (\text{in}(C_3.Rejected) \vee \text{in}(C_3.Aborted))
 \end{aligned}$$

This expression evaluates to *true*, if and only if still enough sub plans can complete. Vice versa, if it evaluates to *false*, too many sub plans have aborted and the parent needs to abort also.

Summarized, the following steps are necessary to take care of the continuation specification of P .

1. Extract the "wait for" construct of P ,
2. turn it into a formula with variables v_1, \dots, v_n ,
3. replace the variables with a given list of conditions $[b_1, \dots, b_n]$,
4. evaluate the resulting expression.

We will use a function $\text{cs}_P([b_1, \dots, b_n])$ to abstract from these steps.

With this function, the transitions $A.3$ and C of Sect. VIII can be adapted as follows.

$$\begin{aligned}
 A.3 & : [\neg \text{cs}_P([\neg (\text{in}(C_i.Rejected)) \\
 & \quad \vee \text{in}(C_i.Aborted)])] \\
 C & : [\text{complete}_{tp} = \dots \\
 & \quad \wedge \text{cs}_P([\text{in}(C_i.Completed)])]
 \end{aligned}$$

The parent plan will abort, if too many plans have been rejected or aborted (transition $A.3$), it will complete, if enough sub plans have completed (transition C).

Also – for parallel execution (see Sect. VII-B.3) – we will start execution, if enough sub plans are selected. For this, we adapt transition a as follows.

$$\begin{aligned}
 a & : \quad [\text{cs}_P([\text{in}(C_i.Selected)])] \\
 & \quad / \quad C_1.activate; \dots; C_n.activate
 \end{aligned}$$

If the body contains option "wait-for-optional", then again enough sub plans must complete to satisfy the continuation specification, and the parent plan additionally waits for all sub plans to at least terminate (either with or without success). Therefore transition C needs to be refined further.

$$\begin{aligned}
 C & : [\text{complete}_{tp} = \dots \\
 & \quad \wedge \text{cs}_P([\text{in}(C_i.Completed)]) \\
 & \quad \wedge \text{wait-for-optional} \equiv \text{yes} \\
 & \quad \rightarrow \bigwedge_{i=1}^n \text{in}(C_i.Terminated)]
 \end{aligned}$$

C. Semantics

Let P be a plan with sub plans C_1, \dots, C_n . Two functions wf and awf are used to turn the "wait for" construct wf into a formula $\varphi \in F$.

$$\begin{aligned} wf & : \textit{wait-for} \rightarrow F \\ awf & : \textit{abstract-wait-for} \rightarrow F \end{aligned}$$

Function wf translates the special cases – i.e. all, one, a fixed number or none of the sub plans must complete – into a formula. A more general specification of type *abstract-wait-for* is taken care of in function awf .

Function wf is defined as follows.

$$\begin{aligned} wf(\textit{all}) & = \bigwedge_{i=1}^n v_i \\ wf(\textit{one}) & = \bigvee_{i=1}^n v_i \\ wf(m) & = \bigvee_{(p)_i \in \textit{perm}([1, \dots, n])} \bigwedge_{i=1}^m v_{p_i} \\ wf(\textit{none}) & = \textit{true} \\ wf(awf) & = awf(awf) \end{aligned}$$

If a fixed number m of sub plans are required to complete, a formula is generated which takes all permutations $\textit{perm}([1, \dots, n])$ of numbers $1, \dots, n$ and requires the first m sub plans of the permuted list to complete. For the semantics it does not matter that the resulting formula is highly redundant.

The definition of function awf is straightforward.

$$\begin{aligned} awf(\textit{name}) & = v_i, \text{ if } \textit{name} \equiv C_i \\ awf(\textit{not } awf) & = \neg awf(awf) \\ awf(awf_1 \textit{ and } awf_2) & = awf(awf_1) \wedge awf(awf_2) \\ awf(awf_1 \textit{ or } awf_2) & = awf(awf_1) \vee awf(awf_2) \\ awf(awf_1 \textit{ xor } awf_2) & = awf(awf_1) \wedge \neg awf(awf_2) \\ & \quad \vee \neg awf(awf_1) \wedge awf(awf_2) \end{aligned}$$

Using these functions, we can define a function

$$cs_P : [\textit{bool}] \rightarrow \textit{bool}$$

which takes a list of boolean values and evaluates for a plan P the continuation specification wf_P by substituting all variables v_1, \dots, v_n in the generated formula $wf(wf_P)$ with the given boolean values as follows.

$$cs_P([b_1, \dots, b_n]) = wf(wf_P)[v_1, \dots, v_n/b_1, \dots, b_n]$$

How to adapt the SOS rules for the adapted transitions is straightforward.

D. Open Issues

- ("retry-aborted" in continuation specification) If option "retry aborted" is chosen, aborted sub plans may still complete. This is currently not considered in the continuation specification!

X. TIMEOUTS

In the example of Fig. 1, a time annotation has been used to describe the monitoring of conditions over time. This is taken care of in the data abstraction unit (see Sect. XII). Additionally, time annotations can be used to define timeouts for plan execution. For example

Regular.Treatments_3 $[[-, 3 \text{ h}], [4 \text{ h}, 6 \text{ h}], [2 \text{ h}, -], \textit{now}]$

would state that the plan needs to be activated within the next 3 hours. Also it should be completed within 4 to 6 hours and the duration of execution should last at least 2 hours.

A. Syntax

time-annotation
= time-range
[starting-shift [earliest *expression*] [latest *expression*]]
[finishing-shift [earliest *expression*] [latest *expression*]]
[duration
[minimum *expression*] [maximum *expression*]]
reference-point (*expression*|*now*)

Within a time-annotation, expressions are used to define a variety of time points. Informally, a plan must be activated within the starting shift. It must complete within the finishing shift and its duration of execution must comply with the minimum and maximum duration. Time values are relative to the given reference point. In this paper, time annotations are abbreviated as follows

$$[[\textit{ess}, \textit{lss}], [\textit{efs}, \textit{lfs}], [\textit{mindu}, \textit{maxdu}], \textit{ref}]$$

and we will use the underscore '_' to represent unspecified values.

[5] describes a number of static checks that a time annotation must satisfy to be considered well-formed. Here, we assume that every time annotation is well-formed.

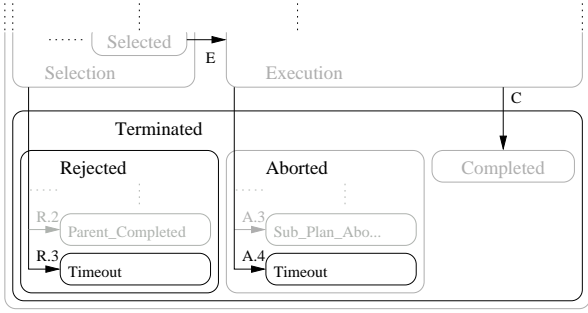
B. Semantics overview

Changes to the plan state model which capture the additional timing constraints are displayed in Figure 13. If, during the selection phase, the latest starting shift has elapsed, the plan is rejected (transition $R.3$). If, during execution, the latest finishing shift has expired, the plan is aborted (transition $A.4$). For the positive case of activating and completing in time, transitions E and C have been adapted. In order to distinguish between the different reasons of failure, the states *Rejected* and *Aborted* have been refined with sub states. Variable *time* is an external variable which is incremented in every step of the environment.

Additional and adapted SOS rules are straightforward.

XI. CYCLICAL PLANS

Cyclical plans are used to model repetition of a single sub plan.



$R.3 : [lss + ref < time]$
 $E : activate$
 $[ess + ref \leq time \leq lss + ref]$
 $/ start_time := time$
 $A.4 : [lfs + ref < time]$
 $\vee maxdu < time - start_time$
 $C : [\dots]$
 $\wedge wait\text{-for}\text{-optional} \equiv yes \rightarrow \dots$
 $\wedge efs + ref \leq time \leq lfs + ref$
 $\wedge mindu \leq time - start_time \leq maxdu$

Fig. 13. Semantics of time annotations for plan activation

A. Syntax

cyclical-plan
 $= cyclical\text{-}plan$
 $start\text{-}time \ number \ cyclical\text{-}time\text{-}annotation$
 $name$
 $/* \ repeat \ specification \ */$
 $(\ cyclical\text{-}time\text{-}annotation$
 $| \ repeat\text{-}specification$
 $[retry\text{-}delay \ [minimum] \ [maximum]]$
 $[duration \ [minimum] \ [maximum]])$
 $/* \ complete \ condition \ */$
 $(cyclical\text{-}complete\text{-}condition)^*$

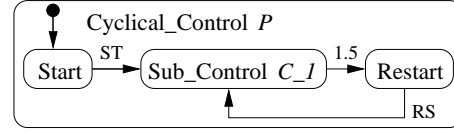
The cyclical plan consists of a "start-time", the name of a sub plane, a repeat specification, and a complete condition.

The complete condition can be of the following forms.

cyclical-complete-condition
 $= (\ end\text{-}time \ cyclical\text{-}time\text{-}annotation$
 $| \ until\text{-}condition \ temporal\text{-}pattern$
 $| \ times\text{-}completed \ number)$

Cyclical time annotations are extensively used in cyclical plans. The difference to time annotations of Sect. X is a more complex specification for the reference time point consisting of a time point, an offset and a frequency.

cyclical-time-annotation
 $= time\text{-}range$
 $[starting \ shift \ [minimum] \ [maximum]]$
 $[finishing \ shift \ [minimum] \ [maximum]]$
 $[duration \ [minimum] \ [maximum]]$



$ST: [ess + ref(n) \leq time \leq lss + ref(n)]/C_1.consider$
 $1.5: [in(C_1.Terminated)]/term_time := time$
 $RS: [mindel \leq term_time - time \leq maxdel]/C_1.retry$

Fig. 14. Controller for cyclical execution

time-point offset frequency

In the following, we will use an abbreviated syntax of cyclical time annotations which is as follows:

$[[ess, lss], [efs, lfs], [mindu, maxdu], tp, offset, frequency]$

B. Semantics overview

For a first version of the semantics we will take a look at the following definition:

cyclical-plan
 $start\text{-}time$
 n
 $[[ess, lss], [efs, lfs], [mindu_1, maxdu_1],$
 $tp, offset, frequency]$
 C_1
 $repeat\text{-}specification$
 $[retry\text{-}delay \ mindel \ maxdel]$
 $[duration \ mindu_2 \ maxdu_2]$

Here we define a cyclical plan, where the repeat specification includes "retry-delay" and duration. We do not use any complete condition.

Slot "start-time" contains a cyclical time annotation. This time annotation defines a set of time points. From this set, the n th time point is used as reference point. The n th time point $ref(n)$ can be calculated according to this formula

$$ref(n) = tp + offset + (n - 1) \cdot frequency$$

A controller for cyclical plans is as in Fig. 14. As soon as $time$ is within the starting interval, sub plan C_1 is considered (transition ST). If C_1 has terminated (transition 1.5), we will repeat the plan, but will wait for the specified "retry-delay" period (transition RS).

C. Open Issues

- How does finishing shift $[efs, lfs]$ or duration $[mindu_1, maxdu_1]$ of "start-time" influence execution?
- How does duration $[mindu_2, maxdu_2]$ of "repeat-specification" influence execution?

- In the above semantics, the cyclical plan waits for the starting time point to be reached and then activates the selection phase of the sub plan. If the sub plan terminates, then – after some delay – the plan is retried. Retrying a plan starts with re-executing the selection phase. Is this correct?

XII. DATA ABSTRACTION

Conditions are given as temporal patterns to allow monitoring of parameters over a longer period of time. Temporal patterns are evaluated in the data abstraction unit.

A. Syntax

temporal-pattern
 = parameter-proposition *formula time-annotation*
 | simple-condition *formula*
 | *temporal-pattern* (\vee | \wedge) *temporal-pattern*
 | \neg *temporal-pattern*

A temporal-pattern is either a parameter proposition (including a time annotation), a simple condition or several patterns combined with \wedge , \vee , or \neg . In contrast to parameter propositions, simple conditions are not evaluated over time.

B. Semantics overview

The underlying data abstraction unit is not described in detail here and only its purpose is summarized. The semantics of the abstraction unit is not operational, but functional in nature. As input, measurements of patient parameters are taken. The type of parameters can be very different reaching from quantitative values, like bilirubin levels in the blood, to boolean values, e.g. whether the patient is male or female. Data can be provided as a continuous stream of patient readings (high frequency domain as in artificial ventilation of prematured babies) or as sporadic measurements once every month (low frequency, e.g. diabetes mellitus).

The incoming data is memorized in the patient record. Quantitative values can be abstracted to qualitative values. An example has been provided in Sect. II. More important, the abstraction unit evaluates data over a longer time period, if the data is time annotated in an ASBRU plan. The example

bilirubin_decrease < 1 [[4 h, -], [-, 6 h], [-, -], *now*]

requires monitoring the decrease of bilirubin level over a period of at least 4 and up to 6 hours.

As output of the abstraction unit, the truth values of conditions are provided. As evaluation of a condition may take time, the result is either *true*, *false*, or yet *unknown*. A condition is *false* only, if it cannot be satisfied in the future. Otherwise, it would be considered *unknown*.

XIII. INTENTIONS

Intentions describe temporal properties of plans and can be verified as described next.

A. Syntax and semantics overview

intentions
 = intentions
 ((intermediate-state|overall-state)
 (avoid|maintain|achieve)
temporal-pattern)*

An intention is to either avoid, maintain, or achieve an overall or intermediate state which is described by a temporal pattern.

Intentions can be translated into temporal logic. Details are omitted here.

B. Verification

One major goal of our project is to formally verify the operational behaviour of Asbru plans against properties which are expressed as intentions. For the example in Sect. II the task would be to verify that *bilirubin* is never equal to transfusion throughout execution – which should be easy – and if the plan completes, *bilirubin* equals to Observation – which is not so obvious.

For verification we are using the interactive theorem prover KIV. We regard automatic verification not powerful enough to deal with the data involved and therefore an interactive verifier is necessary. However it would be worthwhile to define sub tasks which could be treated with model checkers. KIV already supports the verification of parallel programs against properties expressed in temporal logic. The verification strategy is to symbolically execute programs and to use induction, if necessary [2]. A similar approach shall be applied to Asbru plans.

Because of its functional nature, the tasks of the data abstraction unit can be translated directly into algebraic specifications. The difficulty is to capture the operational, state based, parallel behaviour of the Asbru plans themselves. Encoding the SOS rules representing the formal semantics directly into proof rules has been tried but turned out to be too inefficient. Hundreds of proof steps were necessary to execute one Asbru step. In part, this is because of the explicit encoding of the plan hierarchy. A more direct representation of Asbru plans with higher level proof rules is necessary. Currently we are translating Asbru plans into parallel programs preserving the hierarchy of parent and sub plans. With this representation we are able to verify the example intentions, which are translated into temporal logic. However, this translation is only possible for a subset of features and its correctness needs to be examined still. In a future step we would like to directly support Asbru syntax and design proof rules for executing Asbru. In order to be correct, these proof rules still need to adhere to the formal semantics presented here.

XIV. CONCLUSION

The formal semantics of major concepts of Asbru has been explained in this paper. We are confident that the for-

mal semantics alone will help to better understand Asbru plans and thereby improve quality of medical protocols. Furthermore the semantics is an important link between the modelling language and the representation in KIV. An overview on how we will use KIV to verify properties formally has been given. Further research on this topic will be our next major step.

We have chosen to give the formal semantics of Asbru in the form of SOS rules. However, these rules turned out to be difficult to understand. Representing rules as transitions in statecharts resulted in a more compact and intuitive picture of plan behaviour. Even if some of the technical details of the semantics are not correctly captured within these graphics, the statecharts are very suitable for discussions and language documentation.

ACKNOWLEDGEMENTS

This work was possible only thanks to long and fruitful discussions with Silvia Miksch (TU Vienna) and her group Andreas Seyfang and Robert Kosara. Also many thanks to Frank van Harmelen, Mar Marcos (VU Amsterdam), and Annette ten Teije (Univ. Utrecht) for their assistance and good collaboration. The work has been partially funded by the European Community as project 'Protocure' (nr. IST-2001-33049).

REFERENCES

- [1] L. Aceto, W. Fokkink, and C. Verhoef. Structural operational semantics. In J. A. Bergstra, A. Ponse, and S. A. Smolka, editors, *Handbook of Process Algebra*. Elsevier, 2001.
- [2] M. Balsler, C. Duelli, W. Reif, and G. Schellhorn. Verifying concurrent systems with symbolic execution. *Journal of Logic and Computation (Special Issue)*, 2002. (to appear).
- [3] M. Balsler, W. Reif, G. Schellhorn, K. Stenzel, and A. Thums. Formal system development with KIV. In T. Maibaum, editor, *Fundamental Approaches to Software Engineering*, number 1783 in LNCS. Springer, 2000.
- [4] J. Bury, J. Fox, and D. Sutton. The PROforma guideline specification language: progress and prospects. In *Proceedings of the First European Workshop, Computer-based Support for Clinical Guidelines and Protocols (EWGLP 2000)*, 2000.
- [5] G. Duftschmid and S. Miksch. Knowledge-based verification of clinical guidelines by detection of anomalies. *Artificial Intelligence in Medicine*, Special Issue: Workflow Management and Clinical Guidelines in Medicine(22(1)), 2001.
- [6] S. Herbert, C. Gordon, A. Jackson-Smale, and S. Renaud. Protocols for clinical care. *Computer Methods and Programs in Biomedicine*, 48, 1995.
- [7] S. Miksch, Y. Shahar, and P. Johnson. Asbru: A task-specific, intention-based, and time-oriented language for representing skeletal plans. In E. Motta, F. v. Harmelen, C. Pierret-Golbreich, I. Filby, and N. Wijngaards, editors, *7th Workshop on Knowledge Engineering: Methods & Languages (KEML-97)*. Milton Keynes, UK, 1997.
- [8] L. Ohno-Machado, J. Gennari, S. Murphy, N. Jain, S. Tu, D. Oliver, E. Pattison-Gordon, R. Greenes, E. Shortliffe, and G. Barnett. The GuideLine Interchange Format: A model for representing guidelines. *American Medical Association*, 5, 1998.
- [9] G. D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Aarhus University, 1981.
- [10] A. Pnueli and M. Shalev. What is in a step: On the semantics of statecharts. In *Symposium on Theoretical Aspects of Computer Software*, pages 244–264, 1991.
- [11] Protocure – Improving medical protocols by formal methods. <http://www.protocure.org>.
- [12] A. Seyfang, R. Kosara, and S. Miksch. Asbru's reference manual, asbru version 7.2, document revision 1. Technical report, Vienna University of Technology, Institute of Software Technology, 2000.

DELIVERABLES TABLE

Project Number: *IST-2001-33049*
Project Acronym: **PROTOCURE**
Title: *Improving medical protocols by formal methods*

Del. No.	Revision	Title	Type ¹	Classifi- cation ²	Due Date	Issue Date
D0	1	Project presentation	O*	Pub.	28/2/2002	26/2/2002
D1	1	Reference protocols and their Asbru models	R	Pub.	28/2/2002	28/2/2002
D2	1	Formal semantics of main Asbru elements	R	Pub.	31/3/2002	28/3/2002
D3		Desirable/required properties of main Asbru elements	R	Pub.	31/3/2002	
D3'		KIV formalisation	D	Pub.	31/5/2002	
D4		Verification of properties on the reference protocols	R	Pub.	30/9/2002	
D5		Evaluation of results	R	Pub.	30/11/2002	

¹ R: Report; D: Demonstrator; S: Software; W: Workshop; O: Other – Specify in footnote

² Int.: Internal circulation within project (and Commission Project Officer + reviewers if requested)

Rest.: Restricted circulation list (specify in footnote) and Commission SO + reviewers only

IST: Circulation within IST Programme participants

FP5: Circulation within Framework Programme participants

Pub.: Public document

* Report, webpage.