

1 Properties to verify concerning Asbru conditions ----> 1999

Georg Duftscheid¹⁾, Silvia Miksch²⁾

1) University of Vienna, Department of Medical Computer Sciences
Spitalgasse 23, A-1090 Vienna, Austria
{georg.duftscheid}@akh-wien.ac.at, www.akh-wien.ac.at/imc/

2) Vienna University of Technology, Institute of Software Technology
Favoritenstraße 9-11/188, A-1040 Vienna, Austria
silvia@ifs.tuwien.ac.at, www.ifs.tuwien.ac.at/

1.1 Assumptions and Definitions

The following properties are based on the assumption that Asbru-plans are mapped to two different rule bases, called *MPS* and *PC*. While *MPS* represents a direct transformation of the generic Asbru Model of Plan States into a rule base, *PC* contains all conditions of all existing Asbru plans.

Table 1. Rule Base *MPS*: generic Model of Plan States

<i>R1:</i>	<i>Considered(pl,pa)</i> /* starting state for each plan */
<i>R2:</i>	<i>Considered(pl,pa) ∧ filter(pl,pa) → possible(pl, pa)</i>
<i>R3:</i>	<i>Considered(pl,pa) ∧ ¬filter(pl,pa) → rejected(pl, pa)</i>
<i>R4:</i>	<i>possible(pl,pa) ∧ setup(pl,pa) → ready(pl, pa)</i>
<i>R5:</i>	<i>possible(pl,pa) ∧ ¬filter(pl,pa) → rejected(pl, pa)</i>
<i>R6:</i>	<i>ready(pl,pa) ∧ ¬filter(pl,pa) → rejected(pl, pa)</i>
<i>R7:</i>	<i>ready(pl,pa) ∧ ¬setup(pl,pa) → possible(pl, pa)</i>
<i>R8:</i>	<i>ready(pl,pa) ∧ activate(pl,pa) → activated(pl, pa)</i>
<i>R9:</i>	<i>activated(pl,pa) ∧ abort(pl,pa) → aborted(pl, pa)</i>
<i>R10:</i>	<i>activated(pl,pa) ∧ complete(pl,pa) → completed(pl, pa)</i>
<i>R11:</i>	<i>activated(pl,pa) ∧ suspend(pl,pa) → suspended(pl, pa)</i>
<i>R12:</i>	<i>suspended(pl,pa) ∧ reactivate(pl,pa) → activated(pl,pa)</i>
<i>R13:</i>	<i>suspended(pl,pa) ∧ abort(pl,pa) → aborted(pl, pa)</i>

Rule base *PC* is given by the unification of a set of individual rule bases *PC_i*, where each of these represents the mapping of the conditions of one single plan. Table 2 gives an example for a simplified version of a certain *PC_k*, a rule base for all conditions of the plan to treat noninsulin-dependent gestational diabetes mellitus (*GDM-TYPE II*):

Table 2. Rule Base *PC_k*: conditions of plan *GDM-TYPE II*

<i>Female(pa) ∧ pregnant(pa) → filter(GDM-TYPE II, pa)</i>
<i>Test_available(glucose-toler., pa) → setup(GDM-TYPE II, pa)</i>
<i>Activate(GDM-TYPE II, pa)</i> /* automatic activation */
<i>Delivered(pa) → complete(GDM-TYPE II, pa)</i>
<i>State(blood-glucose, high, pa) → suspend(GDM-TYPE II, pa)</i>
<i>State(blood-glucose, normal, pa) → reactivate(GDM-TYPE II, pa)</i>
<i>Insulin-indicator(pa) → abort(GDM-TYPE II, pa)</i>

In order to get a complete model for the execution control of a certain plan *i*, we have to unify its conditions base *PC_i* with the generic rule base for Asbru's Model of Plan States, formally *MPS* ∪ *PC_i*.

A hypotheses *H* can be *inferred* from a rule base *RB* for some environment *E*, if *H* is a logical consequence of supplying *E* as input to *RB*.

Formally: *infer(H, RB, E)* iff $(RB \cup E) \vdash H$, $E \in PatientSet$

A hypotheses *H* is *inferable* from a rule base *RB* if there is some environment *E* such that *H* can be inferred from *RB* for *E*.

Formally: *inferable(H, RB)* iff $(\exists E) infer(H, RB, E)$, $E \in PatientSet$

A rule *R* ∈ *RB* *fires* for some environment *E*, if the antecedent of *R* is a logical consequence of supplying *E* as input to *RB*.

Formally: *fire(R, RB, E)* iff $(\exists \sigma) (RB \cup E) \vdash antec(R)\sigma$, $E \in PatientSet$

A rule *R* ∈ *RB* is *fireable* if there is some environment *E* such that *R* fires for *E*.

Formally: *fireable(R, RB)* iff $(\exists E) fire(R, RB, E)$, $E \in PatientSet$

1.2 Properties

1.2.1 Every Condition must have a chance to be satisfied

Any single condition being part of a plan must have a chance to be satisfied during execution of the plan in order to have an influence on the plan's behavior.

Referring to the above rule bases, we equivalently specify that each rule of PC must be fireable.

Formally:

$$fireable(R, PC) \quad \forall R \in PC$$

1.2.2 Every valid sequence of plan states must be reachable

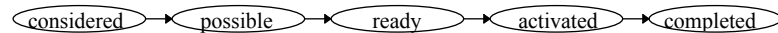
Any sequence of plan states, which defines a valid path according to Asbru's Model of Plan States, must also statically be possible within a single plan. Consequently, all conditions belonging to a valid path of plan states must be satisfiable. Otherwise, one or more states of the plan would not be reachable.

For every rule contained in every PC_i , we demand that there must exist a patient for whom the rule is fireable, considering all rules in PC_i , which have fired before for the same patient. For each rule R in PC_i , the set of rules, which must fire before considering R is defined through MPS . Informally we can therefore specify that for each plan i there must exist an environment E , such that each consequence in MPS can be inferred by supplying E as input to $MPS \cup PC_i$.

Formally:

$$(\exists \sigma) inferable(conseq(R)\sigma, MPS \cup PC_i) \quad \forall (R \in MPS, 1 \leq i \leq n)$$

Example:



According to Asbru's Model of Plan States, the path "CONSIDERED - POSSIBLE - READY - ACTIVATED - COMPLETED" is a valid sequence of plan states. It must therefore statically be reachable and the corresponding conditions must not contradict this demand.

1.2.3 Every plan must be able to complete

For the determination of a plan's ability to complete, it is necessary to check whether the COMPLETE-CONDITION and all predecessor conditions can be satisfied for all plans, belonging to the plan's *continuation set*. The *continuation set* of a certain plan P contains all plans that are *relevant* for plan P 's completion and is determined as follows:

1. Add plan P itself to the *continuation set* of plan P
2. Depending on the operator contained in plan P , add the *relevant subplans* of P to the *continuation set* of plan P . The *relevant subplans* are determined as follows:
 - In case of a DO-SOME-... operator, those children plans are *relevant*, which belonging to the operator's CONTINUATION-CONDITION
 - In case of a DO-ALL-... operator, all children plans are *relevant*
 - In case of a cyclical operator, the child plan is *relevant* only, if a TIMES-COMPLETED condition is specified for the operator
3. Proceed with each *relevant subplan* of plan P as described in Step 2 until the leave-plans are reached

Using the abbreviation CS_i for the *continuation set* of plan i , we specify: For each plan i there must exist an environment E such that the consequence of rule $R10 \in MPS$ can be inferred by supplying *the same* E as input to $MPS \cup PC_k$ for all $k \in CS_i$.

Formally:

$$(\exists E \in PatientSet, \sigma) (infer(conseq(R10)\sigma, MPS \cup PC_k, E)) \\ \forall (1 \leq i \leq n; k \in CS_i; R10 \in MPS)$$

Another point, which is essential for the completions of a plan is that plans, which belong to the same *continuation set*, should not behave antagonistically in the following way: It must not be possible that the completion of one of them causes any other one to be aborted or rejected. Otherwise, the associated parent plan would not be able to complete.

Let us call

$UTR = \{R3, R5, R6, R9, R13\} \subseteq MPS$: ("unsuccessful termination rules") set of rules, leading to final states $\{REJECTED, ABORTED\}$

Formally:

$$\neg (\exists \sigma) infer(conseq(R10)\sigma, MPS \cup PC_j, E) \rightarrow infer(conseq(R)\sigma, MPS \cup PC_k, E) \\ \forall (R \in UTR; R10 \in MPS; j, k \in CS_i; 1 \leq i \leq n; E \in PatientSet)$$

1.2.4 The target state of a transition should be unambiguous

Each time a state transition takes place during the execution of a plan, it must be clear which will be the target state of the transition, as only one state can be active at a time.

Conditions, that fork the state path are the source of potential problems: If more than one of these conditions become true at the same time, the successor plan state will be ambiguous.

The most intuitive solution seems to be to request *mutual exclusion* from *forking-conditions*. However, this strategy is probably unrealistic: We would impose a too harsh restriction on a plan designer by demanding that each set of forking-conditions had to be mutual exclusive. Especially if the conditions are complex, it will be annoying for the designer to be forced to design the conditions in a way that they exclude each other. The extra-code, required to guarantee the mutual exclusion of forking-conditions, would probably seem artificial and unwanted in most cases.

A more relaxed strategy would be to demand that forking-conditions should be independent in a way that their concurrent satisfaction is at least not statically foreseeable. In other words we could specify that the antecedents of forking-conditions must not entail each other. As this demand does not rule out a concurrent satisfaction of two forking-conditions, we must provide a heuristic to determine the proper target-state in a conflict situation of this kind.

For the purpose of the definition of the following anomaly, we will interpret all successor states of a state path fork as *semantic constraint expressions*, meaning that their concurrent occurrence does not make semantic sense.

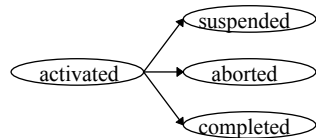
$ConstraintsSet = \{(possible(pl, pa), rejected(pl, pa)), (ready(pl, pa), rejected(pl, pa)), (suspended(pl, pa), aborted(pl, pa)), (suspended(pl, pa), completed(pl, pa)), (completed(pl, pa), aborted(pl, pa)), (activated(pl, pa), aborted(pl, pa))\}$

In order to avoid statically predictable, ambiguous state transitions, we can request that there must not be any pair of rules R and R' in $MPS \cup PC_i$, such that the antecedent of R entails the antecedent of R' , and their consequents infer a semantic constraint expression from $ConstraintsSet$.

Formally:

$$\neg (\exists \sigma) (antec(R)\sigma \rightarrow antec(R')\sigma) \wedge (\{conseq(R)\sigma, conseq(R')\sigma\} \in ConstraintsSet) \\ \forall (R, R' \in MPS \cup PC_i; 1 \leq i \leq n)$$

Example:



The conditions `SUSPEND-CONDITION`, `ABORT-CONDITION` and `COMPLETE-CONDITION` should not be satisfied at the same time.

1.2.5 Child plan should not be stopped by parent

The plan states `{REJECTED, ABORTED, SUSPENDED, COMPLETED}` may be propagated from a plan to its subplans, thereby stopping the latter. This kind of overruling a plan's actual state should not be the ordinary case, and it should therefore be assured, that the relevant conditions avoid such a situation. As state propagation might be deliberately applied in some cases, the violation of this specification is not considered an error, but a warning.

In terms of our knowledge base we informally specify:

SS_i = set of all subplans of plan i

$FSR = \{R3, R5, R6, R9, R10, R13\} \subseteq MPS$: ("final state rules") set of all rules, leading to final states `{REJECTED, ABORTED, COMPLETED}`

$R11 \subseteq MPS$: rule, leading to state `{SUSPENDED}`

Then we demand for each plan i : Whenever the consequent of a rule R from FSR can be inferred for a certain environment E , then it must also be possible to infer the consequent of a rule R' from FSR for each of plan i 's subplans for the same environment E .

Formally:

$$infer(conseq(R)\sigma, MPS \cup PC_i, E) \rightarrow infer(conseq(R')\sigma, MPS \cup PC_k, E) \\ \forall (1 \leq i \leq n; R, R' \in FSR \cup R11; k \in SS_i; E \in PatientSet, \sigma)$$

Remark: By definition, a plan may only complete if its `COMPLETE-CONDITION` is satisfied *and* all plans within its continuation set have completed. Therefore, the state `COMPLETED` may only be propagated from a parent to those of its child plans, which do not belong to its continuation set as the relevant child plans must already be in the `COMPLETED` state at this point.

1.2.6 No state of a plan should be skipped

If the condition of a plan state is unconditionally true as soon as it is checked, because of one or more poorly designed conditions, the plan state at which the condition is checked is skipped and thus unnecessary. Hence, our first request will be that there must not be any single condition, which is inherently true.

Accordingly, we specify informally that there must not be any rule in PC , which fires for every possible environment E .

Formally:

$$\neg (\exists R \in PC) \text{ fire}(R, PC, E) \quad \forall (E \in PatientSet)$$

Our second request concerns the order in which conditions are considered, enforced by MPS . We demand for every pair of conditions within the same plan: The firing of the preceding condition in the inference chain induced by MPS , must not entail the firing of a condition, considered later in the inference chain. Instead, every condition in the chain should check additional information in order to make sense.

Formally:

$$\neg (\exists \sigma) \text{ antec}(R) \rightarrow \text{antec}(R')\sigma \quad \forall (R, R' \in MPS \cup PC_i, 1 \leq i \leq n, R \prec R')$$

$R \prec R'$ means R is considered first in the inference chain.

1.2.7 Every plan must have a chance to execute its body

If there are certain statically predictable situations, in which the conditions activating (FILTER-PRECONDITION and SETUP-PRECONDITION) a plan entail the conditions stopping (WRONG FILTER-, ABORT- and COMPLETE-CONDITION) a plan, the plan has no chance to execute its body then.

Let us call

$FSR = \{R3, R5, R6, R9, R10, R13\} \subseteq MPS$: (“final state rules”) set of all rules, leading to final states {REJECTED, ABORTED, COMPLETED}

We can then informally request that the activation of a plan must not entail its termination.

Formally:

$$\neg (\exists \sigma) \text{ infer}(\text{conseq}(R4)\sigma, MPS \cup PC_i, E) \rightarrow \text{infer}(\text{conseq}(R)\sigma, MPS \cup PC_i, E) \\ \forall (1 \leq i \leq n; R4 \in MPS, R \in FSR; E \in PatientSet)$$

1.2.8 A plan must not loop in its execution

We should avoid situations, where the medical status present after the termination of a plan automatically reactivates the same plan again. We therefore request that a plan’s stopping-conditions must not entail the activation of the plan.

Remember

$FSR = \{R3, R5, R6, R9, R10, R13\} \subseteq MPS$: (“final state rules”) set of all rules, leading to final states {REJECTED, ABORTED, COMPLETED}

Formally:

$$\neg (\exists \sigma) \text{ infer}(\text{conseq}(R)\sigma, MPS \cup PC_i, E) \rightarrow \text{infer}(\text{conseq}(R4)\sigma, MPS \cup PC_i, E) \\ \forall (1 \leq i \leq n; R4 \in MPS, R \in FSR; E \in PatientSet)$$

1.2.9 Every Condition, checked at a “wait-state” must be defined in a way, that it makes sense to wait for its satisfaction

We call states of the Model of Plan States, which typically loop in the checking of their conditions *wait-states*. A plan, which resides in a wait-state does not immediately react to the failure of one of its conditions in sense of a state transition, instead it waits until one of them succeeds.

This in due course requires that the conditions checked at wait-states must be defined in a way, that there is still a chance for their satisfaction, after they failed at a prior evaluation. It only makes sense to wait for a later satisfaction of a condition associated with a wait-state after it failed earlier, if it is referring to a parameter of variable type. Actually most clinical parameters will be variable, there are just a few exceptions that will only allow constant values (e.g. “gender”(“?”), “blood-group”).

Let us call

$\text{CondsCheckedAtWaitStates} = \{\text{setup}(pl, pa), \text{complete}(pl, pa), \text{suspend}(pl, pa), \\ \text{abort}(pl, pa), \text{reactivate}(pl, pa)\}$

We can then formally request:

$$\neg (\exists R \in PC) (\text{conseq}(R) \in \text{CondsCheckedAtWaitStates} \wedge \text{constant}(R))$$

1.2.10 A Condition must only refer to states of other plans, which are reachable

As can be seen from the definition of the <temporal-pattern> concept, a condition may refer to the state of another plan. For every such case it must be ensured that there exists at least one patient for whom both, the state where the condition is checked, and the state of the referred plan can be reached.

A condition referring to a certain state of another plan would be equivalent to a rule R in a rule base PC_i of plan i , which incorporates the consequent of another rule R' in its antecedent. R' would belong to MPS instantiated for PC_j of another plan j , therefore the consequent of R' would be a specific plan state of plan j .

Let us call

$States(R)$ = Set of plan states, at which the condition corresponding to rule $R \in PC$ is checked.

Example: For $R = \text{Insulin-indicator}(pa) \rightarrow \text{abort}(\text{GDM-TYPE II}, pa)$ we would get $States(R) = \{\text{activated}(\text{GDM-TYPE II}, pa), \text{suspended}(\text{GDM-TYPE II}, pa)\}$.

Formally:

$$(\exists E \in \text{PatientSet}, \sigma) (\text{conseq}(R')\sigma \in \text{antec}(R)) \rightarrow \text{infer}(States(R)\sigma, MPS \cup PC_i, E) \wedge \text{infer}(\text{conseq}(R')\sigma, MPS \cup PC_j, E) \quad \forall (R \in PC_i, R' \in MPS \cup PC_j, 1 \leq i, j \leq n)$$

A special situation arises, if plan B referred to by a certain condition of plan A is actually a subplan of plan A: A child plan is not considered before its parent plan reaches the state ACTIVATED. Therefore, a parent plan must not refer to the state of one of its subplans within its FILTER- or SETUP-CONDITION. The only possible state for the child plan would be CONSIDERED at that time and referring to this state does not say anything, as each plan is in the CONSIDERED state upon its creation.

Let us call

$$\text{NoReferenceConditions} = \{\text{filter}(pl, pa), \text{setup}(pl, pa)\}$$

SS_i = set of all subplans of plan i

Formally:

$$\neg (\exists R \in PC_i, R' \in MPS \cup PC_j, \sigma) ((\text{conseq}(R) \in \text{NoReferenceConditions}) \wedge (\text{conseq}(R')\sigma \in \text{antec}(R))) \quad \forall (1 \leq i \leq n; j \in SS_i)$$

1.2.11 A Condition should not contain redundant parameter propositions

A condition may be composed of several conjunctive parameter propositions. Each parameter proposition within a condition should obviously check for some additional data, redundant queries would not make sense. In other words, we would like to avoid conditions containing several identical or entailing parameter propositions.

We assume that each rule is of form $L1 \wedge \dots \wedge Ln \rightarrow M$.

Formally:

$$\neg ((Li \rightarrow Lj) \wedge (i \neq j)) \quad \forall (R \in PC; 1 \leq i, j \leq n)$$