Model Based Reactive Planning and Prediction for Autonomic Systems[*]

Invited Paper

Peter H. Deussen
Fraunhofer Institute for Open Communication Systems
Berlin, Germany
peter.deussen@fokus.fraunhofer.de

ABSTRACT

Dynamic adaptation according to situational variety requires the employment of planning algorithms for reactive and proactive dealing with detected or predicted conditions. The complexity of large scaled, distributed systems however prohibits planning strategies acting on the granularity of single components because (a) a local system view from the perspective of singular components is usually not sufficient, and (b) the synthesis (or a-priory definition) of a global perspective is far beyond feasibility. We therefore propose the use of dynamic abstraction mechanisms to generate planning models of a suitable degree of granularity. The framework of Abstract Interpretation is used to define both a reactive planning algorithm and a pro-active prediction.

Categories and Subject Descriptors

F.1.2 [**Computation by Abstract Devices**]: Modes of Computation—*Parallelism and concurrency, Relativized computation*

Keywords

Models, Autonomic Systems, Pomsets, Pomtrees, Abstraction, Embedding, Zoom, Abstract Interpretation

1. INTRODUCTION

Autonomic Computing [13] and Communication [20] have been introduced as paradigms to cope with the complexity of highly distributed systems and networks where a centralized management approach is doomed to fail. A number of approaches have been presented to perform system operations on the basis of local information, based on the vision of a set of—more or less—simple rules associated with each system component, with the underlying idea that useful behavioral properties "emerge" on the system level.

[*]This work has been supported by the project "Component-ware for Autonomic, Situation-aware Communications, And Dynamically Adaptable Services (CASCADAS)" (IST-027807) funded by the FET Program of the European Commission.

In this paper, we use a slightly different complementary vision of management that is to be performed using information that goes beyond the system perception of a single component, but does not necessarily involve global system view, leading to an approach that performs management action on a level that is as global as needed and as local as possible.

A key element of our approach is that we assume the existence of behavioral descriptions of system components, subsystems, and systems, and that these *models* can be accessed and processed at system runtime. This model based view opens a variety of possible research topics on system and interaction definition and management, thus for instance:

1. A model description of possible interaction modes (semantically annotated protocols) describing the set of services provided by a system component can be used for automatic service composition, based e. g. on negotiation. Models become a mean of **communication**.

2. Moreover, interaction modes of communities of components can be expressed as suitable combinations of the models of the components involved. Models become a mean to understand **emergent behavior**.

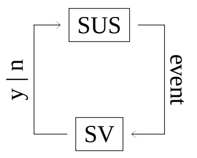
3. Both local component behavior and emergent behavior of communities can be compared with the model descriptions of this behavior, leading to a autonomous self-management approach. Models become a mean to **supervise** systems and system components.

The above list defines a challenging research agenda. In this paper, we are going to provide an approach towards the third topic.

Model-based System Supervision. Supervision may be defined as the ongoing (passive) observation and assessment of the state of a system and the active intervention if an undesired state (such as a failure state) is encountered, and to guide the system back into a more desirable state. Alternatively, intervention can be done proactively to prevent the system from entering a problem state. Both approaches require the availability of planning strategies. A potential supervisor needs to be able to estimate the effect of a supervision action to the overall system state in order to determine whether or not to perform this action. Following the model based approach, we assume that operational system models can be used to estimate effects of system executions.

Clearly, the construction and usage of an overall system model of suitable granularity is—even if this model comprises a finite number of states—far beyond feasibility. Moreover, since autonomic systems are supposed to be open systems able to react on a (probably infinite) set of stimuli from their environment, the finite state assumption is not longer valid. Thus a model based supervision approach has either to exploit the component structure of the supervised system (compositional approach) or, alternatively, has to work with model abstractions exhibiting a smaller (or even finite) state space, or a combination of both alternatives.

The work presented in this paper is of mathematical nature, thus a number of simplifications is imposed. The first one concerns the actuation model that is used. It bases on the permission of possible actions (events) that the system under supervision (SUS) is able to execute, i. e. if there is a choice between different alternative behaviors, the supervisor (SV) is able to control this choice. With other words, we assume that management capabilities are already represented in the SUS, to be used by the SV. The right hand side figure shows the proposed "architecture" for the most simple case.



Another simplification concerns the notion of a "system". We use *deterministic labeled transition systems* with probably infinite state space as generic system "syntax". A partial order semantics based on *partially ordered multisets* [12, 21, 19] is employed to take into account concurrency issues (and to reduce computation complexity that occurs from the explicit representation of interleaving of concurrent actions). *Pomtrees* (also known as *Event Structures* [18]) can be viewed as partial order decision trees. We use pomtrees to represent finite reactive and proactive supervision plans. For the computation of those plans, results from the Petri net theory are exploited, namely the construction of *finite prefixes of branching processes* [16].

Abstract interpretations has been introduced by Cousot & Cousot as a framework for static program analysis [4]. We use a notion of abstraction defined in terms of morphisms between systems and show that the Cousot framework can be applied in this setting: Planning for reactive and pro-active supervision is presented as solutions of fixed point iterations performed in a concrete or abstract domain.

Finally, we explain how to apply the result of this paper in the context of hierarchically organized system models, and discuss how planning for reactive and proactive supervision can be performed on the most concrete (i. e. local) level of a model hierarchy (given a set of abstractions/embeddings).

Methodological Remark. To carry out this work, a combination of a number of approaches from different field has been exploited. We therefore refrain from including the usual section on "Related Work" which could only presented as a seemingly unconnected list of references. We have included a number of remarks that discuss related work in an appropriate context.

We assume that the reader is familiar with the basics of *Lattice theory* (partial orders, (complete) lattices, Tarski's fixed point theorem); the standard references is [2]. Moreover, the language of *commutating diagrams* used in *Category theory* will prove as useful. Deeper knowledge in this area is not required, the interested reader is referred to [1, 14], as well as to Goldblatts excellent introduction [11, Chapter 3]. Examples are presented using a process algebra with semantics presented in a "structures operational semantics (SOS)" style. Milners book on CCS [17] is probably the best introduction into this field.

Organization of the Paper.. Section 2 summarizes the mathematical notions used in this paper, including a brief discussion of Abstract Interpretation. Section 3 introduces labeled transition systems as generic system descriptions, and defines their partial order semantics in terms of pomsets. Moreover, pomtrees are introduced, and computational issues are discussed. In Section 4, abstractions and embeddings are introduced. Embeddings serve as a tool to describe system composition. Supervision is addressed in Section 5. Both reactive planning and pro-active prediction of problem situations is considered. Section 6 shows how to apply the results presented so far in the setting of hierarchical models. Finally, Section 7 summarizes the paper and gives an outlook on further works.

2. BACKGROUND

Basic Notations. For the sake of brevity we adopt the convention that whenever a structure $A = \langle X, Y, \ldots \rangle$ is introduced in a definition, then its components are denoted by X_A, Y_A, …. Since we make extensive use of relations, we use some non-standard notations: Let $R \subseteq A \times B$ be a binary relation. Then for some $a \in A$ we let $R(a) =_{\text{df}} \{b \in B \mid a\,R\,b\}$. This notion applies also to order relations like $\leqslant$. For $C \subseteq A$ we put $R(C) =_{\text{df}} \bigcup_{a \in C} R(a)$. The *inverse* of R is the relation $R^{-1} \subseteq B \times A$ and is defined by: $b\,R^{-1}\,a \Leftrightarrow_{\text{df}} a\,R\,b$. The identity relation $\text{id}_A \subseteq A \times A$ is defined as $a\,\text{id}_A\,b \Leftrightarrow_{\text{df}} a = b$. If $R \subseteq A \times A$ is a relation, we denote by R^+ the smallest transitive relation that contains R, and R^* is $R^+ \cup \text{id}_A$. By $f : A \rightharpoonup B$ we denote the fact that f is a partial function from A to B (i.e. it might be the case that for some $a \in A$ there is no $b \in B$ such that $\langle a, b \rangle \in f$), while as usual $f : A \to B$ indicates that f is a total function. Functional restriction is denoted by $f \restriction C$ for $f : A \rightharpoonup B$ and $C \subseteq A$ and is defined as $f \restriction C =_{\text{df}} f \cap (C \times B)$. If $f : A \rightharpoonup B$ be a partial function, we use as a further shortcut equations like $f(a) = b$ to indicate both that $f(a)$ is defined *and* that f yields b if applied to a.

$\mathbb{N}$ denotes the set of non-negative integers and $\mathbb{O}$ is the set of finite and transfinite ordinals with first limit ordinal ω.

Let $f : X \to X$ be a function. A fixed point of f is some $x \in X$ such that $f(x) = x$. If X is partially ordered by $\sqsubseteq$, then x is a *least fixed point* if $x \sqsubseteq y$ for all other fixed points y of f.

The following is (the first part of) the well-known fixed point theorem of Tarski [22].

THEOREM 1. *Let $\mathscr{X} = \langle X, \leqslant, \bot, \top, \sqcup, \sqcap \rangle$ be a complete lattice, and $\Phi : X \to X$ be a monotone function, i. e. $x \sqsubseteq \Phi(x)$ for all $x \in X$. We define:*

$$\begin{aligned} \Phi^0 &=_{df} \text{id}_X, \\ \Phi^{\delta+1} &=_{df} \Phi \circ \Phi^\delta \text{ for successor ordinals } \delta + 1 \in \mathbb{O}, \\ \Phi^\lambda &=_{df} \bigsqcup_{\delta < \lambda} \Phi^\delta \text{ for limit ordinals } \lambda \in \mathbb{O}. \end{aligned}$$

Then $\bigsqcup_{\lambda \in \mathbb{O}} \Phi^\lambda(\bot)$ is the least fixed point of Φ.

Sometimes, sequences of fixed point iterations are considered, each step taking the result of the previous step as starting point. To make the definitions of those chains more readable, we use the notion $\mathbf{lfp}[\Phi, \bot] =_{\text{df}} \bigsqcup_{\lambda \in \mathbb{O}} \Phi^\lambda(\bot)$.

Abstract Interpretation. In this paper, we adopt the framework of *Abstract Interpretation* (AI) introduced by Cousot & Cousot [4]. AI has its roots in static program analysis (strictness analysis for functional programs and ground term analysis for logic programs are classical examples). The basic idea is to "emulate" the execution of a program not in its original domain of variable values but in an abstract, "more simple" version of this domain. A number of variants of the AI framework are possible [5], that mainly differ on the set of operators for abstraction, concretization, and computation that is used, and the consistency criteria between abstract and concrete program executions. In this paper, we use a variant that comprises of abstraction and concretization functions that form *Galois connections*.

Let $\mathscr{C} = \langle X_{\mathscr{C}}, \sqsubseteq_{\mathscr{C}} \rangle$ and $\mathscr{A} = \langle X_{\mathscr{A}}, \sqsubseteq_{\mathscr{A}} \rangle$ be partially ordered sets. A *Galois connection* between $\mathscr{C}$ and $\mathscr{A}$ is a pair of mappings $\alpha : X_{\mathscr{C}} \to X_{\mathscr{A}}$ and $\gamma : X_{\mathscr{A}} \to X_{\mathscr{C}}$ such that $\alpha(x) \sqsubseteq_{\mathscr{A}} y \Leftrightarrow x \sqsubseteq_{\mathscr{C}} \gamma(y)$ does hold. We write $\alpha, \gamma : \mathscr{D} \rightleftharpoons \mathscr{A}$ in this case.

Let $\mathscr{C} = \langle X_{\mathscr{C}}, \sqsubseteq_{\mathscr{C}}, \bot_{\mathscr{C}}, \top_{\mathscr{C}}, \sqcup_{\mathscr{C}}, \sqcap_{\mathscr{C}} \rangle$ be a complete lattice, where the domain $X_{\mathscr{C}}$ is interpreted as states of some computation process, and $x \sqsubseteq y$ expresses the fact that the state y is more advanced that the state x, i. e. y occurs in a later stage of the computation process. The computation process itself is expressed by some monotone function $\boldsymbol{\Phi} : X_{\mathscr{C}} \to X_{\mathscr{C}}$, and the initial state is the bottom element $\bot_{\mathscr{C}}$. Then the complete computation processes is expressed as

$$\bot_{\mathscr{C}}, \boldsymbol{\Phi}(\bot_{\mathscr{C}}), \boldsymbol{\Phi}^2(\bot_{\mathscr{C}}), \ldots, \boldsymbol{\Phi}^{\omega}(\bot_{\mathscr{C}}), \boldsymbol{\Phi}^{\omega+1}(\bot_{\mathscr{C}}), \ldots$$

which, by Theorem 1, has the limit $\mathbf{lfp}[\boldsymbol{\Phi}, \bot_{\mathscr{C}}]$.[1]

Let $\alpha, \gamma : \mathscr{D} \rightleftharpoons \mathscr{A}$ be a Galois connection for complete lattices $\mathscr{C}$ and $\mathscr{A}$, and let $\boldsymbol{\Psi} : X_{\mathscr{A}} \to X_{\mathscr{A}}$ be a monotone function. Then we have $x \sqsubseteq_{\mathscr{C}} (\gamma \circ \boldsymbol{\Psi} \circ \alpha)(x)$, i. e. $\boldsymbol{\Phi}(\boldsymbol{\Phi}^{\lambda}(\bot_{\mathscr{C}})) \sqsubseteq_{\mathscr{C}} (\gamma \circ \boldsymbol{\Psi} \circ \alpha)(\boldsymbol{\Phi}^{\lambda}(\bot_{\mathscr{C}}))$, and $\mathbf{lfp}[\boldsymbol{\Phi}, \bot_{\mathscr{C}}] \sqsubseteq_{\mathscr{C}} \gamma(\mathbf{lfp}[\boldsymbol{\Psi}, \bot_{\mathscr{A}}])$. Thus the abstract computation process

$$\bot_{\mathscr{A}}, \boldsymbol{\Psi}(\bot_{\mathscr{A}}), \boldsymbol{\Psi}^2(\bot_{\mathscr{A}}), \ldots, \boldsymbol{\Psi}^{\omega}(\bot_{\mathscr{A}}), \boldsymbol{\Psi}^{\omega+1}(\bot_{\mathscr{A}}), \ldots$$

has a limit that is an over-approximation of the limit of the concrete computation process, and moreover, each state of the abstract computation process is an over-approximation of its concrete counterpart.

Remark 1. The approach described above can be further improved by means of so-called *widening* and *narrowing* operators. The idea is that widening is used to accelerate the stabilization of the abstract computation process by replacing the various stages of the process by over-approximations so that the resulting process contains only finitely many steps. The price that is paid is that the resulting limit my exceed the limit of the original computation process, i. e. the now finite process reaches a limit $x \sqsupset \mathbf{lfp}[\boldsymbol{\Psi}, \bot]$. A further element of imprecision is added. Narrowing is then used to compute a solution y with $x \sqsupseteq y \sqsupseteq \mathbf{lfp}[\boldsymbol{\Psi}, \bot]$ as a corrective measure. Still, the result might be an over-approximation. It is however shown in [6] that Abstract Interpretation based on widening/narrowing is more general that the Galois connection variant.

While computation sequences with widening/narrowing can be defined in a fairly generic way, the concrete definition of the these operators seem to require a distinction between program control structure and program data: widening is evidently used to terminate program loops by "jumping" to program data that fulfil the termination condition.

In this paper, we use labeled transition systems to describe computation processes, so no particular syntactically identifiable control structure is available. It is therefore not obvious how to apply the widening/narrowing approach to the notion of systems used in this paper. A possible direction is to exploit the fact that so-called cut-off events (introduced in Section 3) define—in a certain sense—articulation points of loops within sequential processes. This however is subject of further work. $\square$

[1]Usually, countable domains X are used. In this case, the computation process converges at some $\lambda \leqslant \omega$.

3. SYSTEMS AND THEIR PARTIAL ORDER SEMANTICS

A very generic notion of the term "systems" bases on a notion of the *states* that the system can assume, and a relation on states that models the actions that modify states. We consider furthermore the situation that certain actions can be executed concurrently, i. e. in no definite order, and assume that concurrent executions are reflected in the structure of the system (diamond properties, see below). This section summarizes the necessary definitions and propositions on systems that are needed throughout this paper. Proofs of lemmas and theorems can be found in [7] if not otherwise stated.

Distributed Alphabets and Systems. Let A be a set of symbols, called *actions*, and $I \subseteq A \times A$ be an irreflexive and symmetric *independence relation*. Then $\boldsymbol{\Sigma} = \langle A, I \rangle$ is called a *distributed alphabet*, or, for short, a d-alphabet. By $D_{\boldsymbol{\Sigma}} =_{\text{df}} (A \times A) \setminus I$ we denote the *dependence relation* associated with $\boldsymbol{\Sigma}$. We moreover stipulate that for each $a \in A$, the set $D(a)$ is finite.

Example 1. Consider a distributed system comprising a number of communicating sequential processes. Each of these processes can be assumed to run at some "location" (e. g. an Internet host, a JAVA thread, etc.). Give a set of *locations* L, and an assignment $l : A \to \mathscr{P}(L)$ encoding the fact that an action $a \in A$ is executed at the locations $l(a) \subseteq L$, we can define a dependence relation $D \subseteq A \times A$ as $a\,D\,b \Leftrightarrow_{\text{df}} l(a) \cap l(b) \neq \varnothing$. Then $\boldsymbol{\Sigma} = \langle A, D \rangle$ forms a d-alphabet. Conversely, if $\boldsymbol{\Sigma}$ is a d-alphabet, then we may define a $D_{\boldsymbol{\Sigma}}$-clique $C \subseteq A_{\boldsymbol{\Sigma}}$ in $\boldsymbol{\Sigma}$ as a maximal dependent subset, i. e. $C \in \max_{\subseteq}\{C' \subseteq A_{\boldsymbol{\Sigma}} \mid C' \subseteq D_{\boldsymbol{\Sigma}} \times D_{\boldsymbol{\Sigma}}\}$. Let $\boldsymbol{C}(\boldsymbol{\Sigma})$ denote the set of all $D_{\boldsymbol{\Sigma}}$-cliques in $\boldsymbol{\Sigma}$. Then $\boldsymbol{C}(\boldsymbol{\Sigma})$ can be used as a set of locations, and the mapping $l : A_{\boldsymbol{\Sigma}} \to \mathscr{P}(\boldsymbol{C}(\boldsymbol{\Sigma}))$ is simply given by $l(a) = \{C \in \boldsymbol{C}(\boldsymbol{\Sigma}) : a \in C\}$. $\square$

A *labeled transition system* (lts) over a d-alphabet $\boldsymbol{\Sigma}$ is a pair $\mathscr{S} = \langle S, \delta \rangle$, where S is a (not necessarily finite) set of *states*, and $\delta = \{\delta_a : S \rightharpoonup S\}_{a \in A}$ is a family of partial transition functions. We usually abbreviate $\delta_a(s)$ by $a_{\mathscr{S}}(s)$. $\mathscr{I} = \langle S, \delta, \varphi \rangle$ is an *interpretation* of $\boldsymbol{\Sigma}$ if $\langle S, \delta \rangle$ is a lts over $\boldsymbol{\Sigma}$ and for all $s, s_1, s_2 \in S$ and for all $a, b \in A_{\boldsymbol{\Sigma}}$ with $a\,I_{\boldsymbol{\Sigma}}\,b$ the following *diamond properties* are fulfilled whenever the expressions displayed are defined:

$$a_{\mathscr{I}}(s) = s_1 \;\&\; b_{\mathscr{I}}(s) = s_2 \Rightarrow b_{\mathscr{I}}(s_1) = a_{\mathscr{I}}(s_2),$$
$$a_{\mathscr{I}}(b_{\mathscr{I}}(s)) = b_{\mathscr{I}}(a_{\mathscr{I}}(s)).$$

Further, $\varphi \subseteq S$ is a *state invariant*. By $\text{en}_{\mathscr{I}}(s) =_{\text{df}} \{a \in A_{\boldsymbol{\Sigma}} : a_{\mathscr{I}}(s) \text{ is defined}\}$ we denote the set of actions *enabled* at a state $s \in S$. The class of interpretations of a d-alphabet $\boldsymbol{\Sigma}$ is denoted by $\mathbf{I}(\boldsymbol{\Sigma})$.

Example 2. For examples, we need a concrete syntax to express distributed systems. The obvious choice is to use directed graphs, where nodes represent states, and edges are labeled with actions. Another choice would be some state machine syntax, probably equipped with state variables, transition conditions and transitions actions (depending on the values of the state variables), i.e. some type of Extended Finite State Machine. A third option (which would make the definition of d-alphabets quite instructive) would be a Petri-net based formalism.

Graphical formalisms however have the disadvantage that the dynamic creation of system components and their termination requires some extension to the language (e. g. graph grammar rules), since introducing a new component means to modify the graph itself. To avoid concentrating too much on the explanation of the example formalism, which already assumes a significant part of this paper, we therefore use a text based formalism where dynamic component creation can be defined in terms of textual replacement.

$$P ::= \varepsilon \mid a.P \mid x \leftarrow d.P \mid X \mid \mathbf{0} \mid \mathbf{0}_v \mid \textstyle\sum_{i\in I} P_i \mid P_1 \parallel P_2 \mid X =_{\mathrm{df}} P \mid X^x =_{\mathrm{df}} P \mid X^{x\leftarrow v};$$

(a) *Syntax*. $v \in R$ is a process identifier, $d \in D$ is a value of domain D, x a variable, X a process variable, and a is an action (probably indexed with values v or variables x). ε denotes the *empty process*.

$$(\mathrm{step})\frac{}{a.P;R,\sigma \xrightarrow{a} P;R,\sigma};\quad (\mathrm{ass})\frac{}{x \leftarrow d.P,R,\sigma \xrightarrow{a} P,R,\sigma[x/d]};$$

$$(\mathrm{rec}_1)\frac{X^{x\leftarrow v};R,\sigma}{P[x\leftarrow v];R,\sigma} X^x =_{\mathrm{df}} P;\quad (\mathrm{rec}_2)\frac{X^x;R,\sigma}{P[x\leftarrow v];R\setminus\{v\},\sigma} X^x =_{\mathrm{df}} P \,\&\, v \in R;\quad (\mathrm{rec}_3)\frac{X;R,\sigma}{P;R,\sigma} X =_{\mathrm{df}} P$$

$$(\mathrm{asyn}_1)\frac{P_1;R,\sigma \xrightarrow{!a} P_2;R,\sigma}{P_1 \parallel P;R,\sigma \xrightarrow{!a} P_2 \parallel P;R,\sigma};\quad (\mathrm{asyn}_2)\frac{P_1;R,\sigma \xrightarrow{!a} P_2;R,\sigma;P_3;R,\sigma \xrightarrow{?a} P_4;R,\sigma}{P_1 \parallel P_3;R,\sigma \xrightarrow{!a\circ_\Sigma ?a} P_2 \parallel P_4;R,\sigma};\quad (\mathrm{syn})\frac{P_1;R,\sigma \xrightarrow{a} P_2;R,\sigma;P_3;R,\sigma \xrightarrow{a} P_4;R,\sigma}{P_1 \parallel P_3;R,\sigma \xrightarrow{a} P_2 \parallel P_4;R,\sigma};$$

$$(\mathrm{com})\frac{P_1 \parallel P_2;R,\sigma}{P_2 \parallel P_1;R,\sigma};\quad (\mathrm{choice})\frac{P_i;R,\sigma \xrightarrow{a} P;R,\sigma}{\sum_{i\in I} P_i;R,\sigma \xrightarrow{a} P;R,\sigma} i \in I;\quad (\mathrm{stop}_1)\frac{\mathbf{0};R,\sigma}{R,\sigma};\quad (\mathrm{stop}_2)\frac{\mathbf{0}_i;R,\sigma}{R\cup\{i\},\sigma};\quad (\mathrm{cnt})\frac{C[P;R,\sigma],R}{C[P];R,\sigma}.$$

(b) *Semantics*. $P[x \leftarrow v]$ is obtained by replacing the variable x in the term P by the value v, and $C[P]$ means that the term or state P occurs in the syntactic context C. For $\sigma : V \to D$, $\sigma[x/d] : V \to D$ is defined by $\sigma[x/d](y) =_{\mathrm{df}} d$, if $x = y$, and $\sigma[x/d](y) =_{\mathrm{df}} \sigma(y)$ otherwise.

Figure 1: Example process algebra

Figure 1 describes syntax and semantics of a process algebra with variables (taken from a set V) over a data domain D. States of distributed systems described in this language are tuples $\langle P,R,\sigma\rangle$ (written as of $P;R,\sigma$) comprising a *term* P of the algebra, together with a variable state $\sigma : V \to D$, and a countable infinite set R of unused *process identifiers*. This set is used to create new components, and to keep track of the termination of existing ones. Processes are defined by recursive equations of the form $X^x =_{\mathrm{df}} P$, where X is a process variable and x is a process identifier variable ranging over R. We allow x to be omitted, and to be target of an assignment (in this case we write $X^{x\leftarrow r}$). The transition function δ_a is written in a relational style using arrows, i. e. $P_1;R_1,\sigma_1 \xrightarrow{a} P_2;R_2\sigma_2 \Leftrightarrow_{\mathrm{df}} \delta_a(\langle P_1,R_1,\sigma_1\rangle) = \langle P_2,R_2,\sigma_2\rangle$. An example system is defined in the following paragraphs.

System Actions and Dependencies. Let $I \subseteq R$ be a set of *user identifiers*, J be a set of *scheduler identifiers*, and Q be a set of *service identifiers*. The action set A_Σ of a distributed alphabet Σ is defined by the regular expression $(?+!)(\mathrm{r}_{q,i,j}+(\mathrm{g}+\mathrm{d}+\mathrm{p}+\mathrm{e})_{q,i})+\mathrm{u}_{q,i}+x_q \leftarrow d$ for $q \in Q$, $i \in I$, and $j \in J$, and variable symbols x_q for each service $q \in Q$. Moreover, to define the dependence relation D_Σ, we impose the rule that actions a and b (described by the above syntax) are dependent if they share the same user identifier, the same service identifer, or the same scheduler identifier.

Process Equations. Our example system consists of the following processes, defined by a set of recursive equations:

$$\begin{aligned}
G &=_{\mathrm{df}} \sum_{q\in Q} U_q^x \parallel G\\
U_q^x &=_{\mathrm{df}} \sum_{j\in J} !\mathrm{r}_{q,x,j}.\{?\mathrm{g}_{q,x}.?\mathrm{p}_{q,x}.\mathrm{u}_{q,x}.!\mathrm{e}_{q,x}+?\mathrm{d}_{q,x}\}.\mathbf{0}_x\\
C_j &=_{\mathrm{df}} \sum_{q\in Q} ?\mathrm{r}_{q,i,j}.\{!\mathrm{g}_{q,i}.(V_q^{x\leftarrow i} \parallel C_j)+!\mathrm{d}_{q,i}.C_j\}\\
V_q^x &=_{\mathrm{df}} x_q \leftarrow x_q+1.!\mathrm{p}_{q,x}.?\mathrm{e}_{q,x}.x_q \leftarrow x_q-1.\mathbf{0},
\end{aligned}$$

where for each service q the variable x_q ranges over the domain $D = \mathbb{N}$. We chose $G \parallel ||_{j\in J} C_j;R,\sigma : x_q \mapsto 0$ as an initial state of the system's executions.

The intuition of the example is as follow: A generator process G initiates an arbitrary number of user processes U_q^i which in turn contact one of the scheduler processes C_j by sending a request $!\mathrm{r}_{q,i,j}$ for a service q. The scheduler decides whether to grant the request by replying with $!\mathrm{g}_{q,i}$, or to deny it by sending $!\mathrm{d}_{q,i}$. In the former case, the scheduler initiates a service process V_q^i. This process increases the number of users of the service q stored in the variable x_q, and sends a $!\mathrm{p}_{q,i}$ to the user i to indicate that the service is ready to be used. User i executes the internal action $\mathrm{u}_{q,i}$, then it indicates the exit from the service usage by sending $!\mathrm{e}_{q,i}$ to the service process which then decreases x_q. Finally, both the user process and the service process terminate.

State Invariant and Error Mode. We now assume that the number of users that are allowed to use a service q is limited by some constant N_q. We establish the state invariant $\varphi =_{\mathrm{df}} \{\langle P,R,\sigma\rangle : \forall q \in Q.\sigma(x_q) \leqslant N_q\}$.

The trouble with our example is now that the state invariant is defined in terms that do not have an influence to the execution of the system—-a user process will use the service q regardless wether $x_q \leqslant N_q$ does hold or not. Instead of extending our process language and the example even more, we simply impose new semantic rules:

$$(\mathrm{use})\frac{\mathrm{u}_q.P;R,\sigma}{P;R,\sigma}\sigma(x_q) \leqslant N_q,\ (\mathrm{crash})\frac{\mathrm{u}_q.P;R,\sigma}{\varepsilon;R,\sigma[x_q/N_q+1]}\sigma(x_q) > N_q,$$

i. e. we "crash" a user process that tries to access a service q in case $\sigma(x_q) > N_q$, blocking this service.[2] □

Pomsets. A *labeled partial order* (lpo) $u = \langle E,\leqslant,\lambda\rangle$ over some alphabet A comprises a set of *events* E, a partial order $\leqslant\, \subseteq E \times E$ called *causal order* of u, and a *labeling function* $\lambda : E \to A$. We define $e_1 \lessdot e_2 =_{\mathrm{df}} \forall e \in e.[e_1 \leqslant e \leqslant e_2 \Rightarrow (e = e_1 \vee e = e_2)]$. We usually write $\tilde{e}$ instead of $\lambda(e)$, if it is clear from the context which pomset u is meant. For subsets $E \subseteq E_u$ we put $E^\downarrow =_{\mathrm{df}} \leqslant_u^{-1}(E)$, and $e^\downarrow =_{\mathrm{df}} \{e\}^\downarrow$ for $e \in E_u$, if confusion is not possible. By $\varepsilon =_{\mathrm{df}} \langle\varnothing,\varnothing,\varnothing\rangle$ we denote the *empty lpo*. Moreover, if $a \in A$, we denote the *letter* $\langle\{0\},\varnothing,\lambda : 0 \mapsto a\rangle$ also by a.

A subset $E \subseteq E_u$ is called pre-closed in u if $E^\downarrow = E$. For pre-closed sets E we define $u[E] =_{\mathrm{df}} \langle E,\leqslant_u \cap E \times E,\lambda_u \restriction E\rangle$. The *concurrency relation* of u is given by $e_1 \ \mathrm{co}_u\ e_2 =_{\mathrm{df}} \neg(e_1 \leqslant_u e_2)$ & $\neg(e_2 \leqslant_u e_1)$. A co-set of u is a set $E \subseteq E_u$ such that $E \times E \subseteq \mathrm{co}_u$. A lpo u is called finitary if $E^\downarrow$ is finite for each finite co-set E of u. Lpos u and v are *isomorphic* if there is a bijection $f : E_u \to E_v$ such that $e_1 \leqslant_u e_2 \Leftrightarrow f(e_1) \leqslant_v f(e_2)$ and $\lambda_v \circ f = \lambda_u$. By $[u]$ we denote the equivalence class of u and call $[u]$ a *partially ordered multiset* (pomset).

Remark 2. To get rid of the just introduced notion $[u]$ we do not distinguish between u and its equivalence class $[u]$. From a formal

[2]The service can actually still be used, as $\sigma(x_q)$ can be decreased by service processes that are active when the "crash" happens.

point of view, this is clearly an oversimplification. But since all concepts that we will discuss throughout this paper are either independent of the concrete choice of a representative $v \in [u]$, or—if not—can be "corrected" by the right choice of v, no harm is done. $\square$

Let Σ be a d-alphabet and u be a pomset. Assume $\tilde{e}_1 \; D_\Sigma \; \tilde{e}_2 \Rightarrow e_1 \leqslant_u e_2 \vee e_2 \leqslant_u e_1$ does hold. Then u is called a *pomset* over Σ. By $\mathbf{Ps}(\Sigma)$ we denote the class of pomsets over Σ. Throughout this paper, we assume that all pomsets $u \in \mathbf{Ps}(\Sigma)$ are finitary, and moreover, that E_u is a countable set.

LEMMA 1. *If $u \in \mathbf{Ps}(\Sigma)$ is finitary, then $u = \bigvee\{v \in \mathbf{Ps}(\Sigma) : v \leqslant u$ and v is finite$\}$.*

An example where blurring up representatives and equivalence classes requires the right choice of a representative is the following: Define $u \leqslant v$ if there is some injection $f : E_u \to E_v$ such that $f\left(e^{\downarrow}\right) = f(e)^{\downarrow}$ and $\lambda_v \circ f = \lambda_u$. If $u \leqslant v$ holds, then u is called a *prefix* of v. Furthermore, put $u \preccurlyeq v$ if there is some bijection $f : E_u \to E_v$ such that $f\left(e^{\downarrow}\right) \subseteq f(e)^{\downarrow}$ and, as before, $\lambda_v \circ f = \lambda_u$. Then u is called *weaker* than v.

For some $u \in \mathbf{Ps}(\Sigma)$, we define $\langle u \rangle_\Sigma =_{\mathrm{df}} \langle E_u, R^*, \lambda_u \rangle$, with $e_1 \; R \; e_2 \Leftrightarrow_{\mathrm{df}} e_1 \leqslant_u e_2 \;\&\; \tilde{e}_1 \; D_\Sigma \; \tilde{e}_2$. $\langle u \rangle_\Sigma$ is called the *weakening* of u w. r. t. Σ. We define $\mathbf{Ps}^{\mathrm{w}}(\Sigma) =_{\mathrm{df}} \{\langle u \rangle_\Sigma : u \in \mathbf{Ps}(\Sigma)\}$

LEMMA 2. *For $u, v \in \mathbf{Ps}(\Sigma)$, (a) $\langle u \rangle_\Sigma \in \mathbf{Ps}(\Sigma)$, (b) $v \preccurlyeq \langle u \rangle_\Sigma \Rightarrow v = \langle u \rangle_\Sigma$; (c) $u \preccurlyeq v \Rightarrow \langle u \rangle_\Sigma = \langle v \rangle_\Sigma$; (d) $\langle \langle u \rangle_\Sigma \rangle_\Sigma = \langle u \rangle_\Sigma$.*

Furthermore, let us assume $E_u \cap E_v = \varnothing$ for appropriate representatives of $u, v \in \mathbf{Ps}(\Sigma)$. We define $u \circ_\Sigma v =_{\mathrm{df}} \langle E_u \cup E_v, R^*, \lambda_u, \lambda_v \rangle$, with $e_1 \; R \; e_2 =_{\mathrm{df}} e_1 \leqslant_u e_2 \vee e_1 \leqslant_v e_2 \vee (e_1 \in E_u \;\&\; e_2 \in E_v \;\&\; \tilde{e}_1 \; D_\Sigma \; \tilde{e}_2)$. $u \circ_\Sigma v$ is called the *weak sequential composition* of u and v w. r. t. Σ.

LEMMA 3. *For all $u, v, w \in \mathbf{Ps}^{\mathrm{w}}(\Sigma)$, (a) $u \circ_\Sigma v \in \mathbf{Ps}^{\mathrm{w}}(\Sigma)$, (b) $\langle u \rangle_\Sigma \circ_\Sigma \langle v \rangle_\Sigma = \langle u \circ_\Sigma v \rangle_\Sigma$; (c) $\varepsilon \circ_\Sigma u = u = u \circ_\Sigma \varepsilon$; (d) $u \circ_\Sigma (v \circ_\Sigma w) = (u \circ_\Sigma v) \circ_\Sigma w$.*

Semantics. By virtue of the operator $\circ_\Sigma$, the transition function $\delta_{\mathscr{I}}$ of a system $\mathscr{I} \in \mathbf{I}(\Sigma)$ can be extended to a partial mapping $\delta_u^v : S_{\mathscr{I}} \rightharpoonup S_{\mathscr{I}}$ for $u \in \mathbf{Ps}^{\mathrm{w}}(\Sigma)$ as follows: $\delta_\varepsilon =_{\mathrm{df}} \mathrm{id}_{S_{\mathscr{I}}}$, $\delta_{u \circ_\Sigma a} =_{\mathrm{df}} \delta_{\mathscr{I},a} \circ \delta_u$ for finite u. We say that $\delta_u(s)$ *is defined* even for infinite u if $\delta_v(s)$ is defined for all finite v with $v < u$. As before, we abbreviate $\delta_v(s) =_{\mathrm{df}} v_{\mathscr{I}}(s)$. By $\mathbf{L}_{\mathscr{I}}(s) =_{\mathrm{df}} \{v \in \mathbf{Ps}^{\mathrm{w}}(\Sigma) : v_{\mathscr{I}}(s)$ is defined$\}$ we denote the *pomset language* of $\mathscr{I}$ at $s \in S_{\mathscr{I}}$.

Example 3. Consider the system from Example 2 and suppose for simplicity $J = \{c\}$, and $Q = \{q\}$. The first steps of an example execution are shown in Fig. 2. Note that the resulting pomset forms a chain as in the example execution no concurrent computations are performed.

Pomsets are more readable if drawn as directed acyclic graphs (without transitive edges). Fig. 3 shows a pomset representing an execution of our example system where two users try to access a service q, the first one succeeds while the second one fails. $\square$

$$\begin{tikzcd}
!\mathrm{r}_{q,1,c} \arrow[r] & ?\mathrm{r}_{q,1,c} \arrow[r] & !\mathrm{g}_{q,1} \arrow[r] \arrow[dr] & ?\mathrm{g}_{q,1} \arrow[r] & x_q^+ \arrow[r] & !\mathrm{p}_{q,1} \arrow[r] & ?\mathrm{p}_{q,1} \arrow[r] & \mathrm{u}_{q,1} \arrow[r] & !\mathrm{e}_{q,1} \arrow[r] & ?\mathrm{e}_{q,1} \arrow[r] & x_q^- \\
 & & & !\mathrm{r}_{q,2,c} \arrow[r] & ?\mathrm{r}_{q,2,c} \arrow[r] & !\mathrm{d}_{q,2} \arrow[r] & ?\mathrm{d}_{q,2} & & & &
\end{tikzcd}$$

(x_q^+ (x_q^-) abbreviates $x_q \leftarrow x_q + 1$ ($x_q \leftarrow x_q - 1$))

Figure 3: Example execution (pomset).

$$\begin{tikzcd}
!\mathrm{r}_{q,1,c} \arrow[r] & ?\mathrm{r}_{q,1,c} \arrow[r] & !\mathrm{g}_{q,1} \arrow[r] \arrow[dr] & ?\mathrm{g}_{q,1} \arrow[r] & x_q^+ \arrow[r] & !\mathrm{p}_{q,1} \arrow[r] & ?\mathrm{p}_{q,1} \arrow[r] & \mathrm{u}_{q,1} \arrow[r] & !\mathrm{e}_{q,1} \arrow[r] & ?\mathrm{e}_{q,1} \arrow[r] & x_q^- \\
 & & & \mathrm{r}_{q,2,c} \arrow[r] & ?\mathrm{r}_{q,2,c} \arrow[r] \arrow[dr] & !\mathrm{d}_{q,2} \arrow[r] \arrow[d, dash, dashed] \arrow[dr, dash, dashed] & ?\mathrm{d}_{q,2} \arrow[d, dash, dashed] \arrow[dl, dash, dashed] & & & & \\
 & & & & & !\mathrm{g}_{q,2} \arrow[r] & ?\mathrm{g}_{q,2} & & & &
\end{tikzcd}$$

Figure 4: A pomtree.

Pomtrees. The trouble with pomsets is that they do not form a complete lattice which is required to apply the AI framework of Cousot & Cousot, because $u \vee v$ might not be a pomset over Σ. We therefore extend the definition of pomsets to incorporate a notion of "conflict" or "branching behavior":[3]

Let Σ be a d-alphabet. A *pomtree* over Σ is a pomset ζ is a pomset over A_Σ together with a *conflict relation* $\sharp \subseteq E_\zeta \times E_\zeta$ defined to be the smallest irreflexive and symmetric relation satisfying: (a) $\tilde{e}_1 \; D_\Sigma \; \tilde{e}_2 \;\&\; e_1 \; \mathrm{co}_\zeta \; e_2 \Rightarrow e_1 \sharp e_2$, and (b) $e_1 \sharp e_2 \;\&\; e_1 \leqslant_\zeta e_3 \Rightarrow e_3 \sharp e_2$. By $\mathbf{Pt}(\Sigma)$ we denote the class of pomtrees over Σ.

Example 4. Fig. 4 shows an example of a pomtree that expresses two different executions of the system described in Example 2. The conflict relation is represented by a dashed line. $\square$

A configuration of a pomtree ζ is a prefix $u \leqslant \zeta$ of ζ such that $u \in \mathbf{Ps}^{\mathrm{w}}(\Sigma)$. By $\mathbf{Cf}(\zeta)$ we denote the class of configurations of ζ. Further, define $\mathbf{Cf}_\vee(\zeta) =_{\mathrm{df}} \max_\leqslant \mathbf{Cf}(\zeta)$

LEMMA 4. *Let $\zeta \in \mathbf{Pt}(\Sigma)$ and let $E \subseteq E_\zeta$ a pre-closed set of event. If $E \times E \cap \sharp_\zeta = \varnothing$, then $\zeta[E] \in \mathbf{Cf}(\zeta)$.*

LEMMA 5. *For all $\zeta, \xi \in \mathbf{Pt}(\Sigma)$, $\zeta \vee \xi$ is defined and is a least upper bound of ζ and ξ w. r. t. $\leqslant$. Moreover, $\zeta = \bigvee \mathbf{Cf}(\zeta)$.*

Obviously, $\mathbf{Cf}(\zeta)$ is prefix closed, in particular, $\varepsilon \in \mathbf{Cf}(\zeta)$. Furthermore, note that $\sharp_\zeta$ is uniquely defined by ζ and Σ. Thus each pomset $u \in \mathbf{Ps}(A_\Sigma)$ can be viewed as a pomtree over Σ with generated conflict relation $\sharp_u = \varnothing$). We therefore allow expressions like $u \vee v$ for $u, v \in \mathbf{Ps}^{\mathrm{w}}(\Sigma)$, even if $u \vee v \notin \mathbf{Ps}^{\mathrm{w}}(\Sigma)$, since (using the implicit construction rule for $\sharp_{u \vee v}$) $u \vee v \in \mathbf{Pt}(\Sigma)$.

THEOREM 2. *Let $\mathscr{I} \in \mathbf{I}(\Sigma)$ and $s \in S_{\mathscr{I}}$. Define $\mathbf{T}_{\mathscr{I}}(s) =_{df} \{\bigvee\{u : \exists v \in X . u \leqslant v\} : X \subseteq \mathbf{L}_{\mathscr{I}}(s)\}$ to be the pomtree language of $\mathscr{I}$ at s. Then the structure $\langle \mathbf{T}_{\mathscr{I}}(s), \leqslant, \varepsilon, \bigvee \mathbf{T}_{\mathscr{I}}(s), \vee, \wedge \rangle$ forms a complete lattice.*

Remark 3. We occasionally use the notion $\zeta \circ_\Sigma \xi$ for some $\zeta, \xi \in \mathbf{Pt}(\Sigma)$ to denote one of the pomtrees $\bigvee(\{u \circ_\Sigma v : v \in \mathbf{Cf}_\vee(\xi)\} \cup \mathbf{Cf}(\zeta) \setminus \{u\})$ for $u \in \mathbf{Cf}(\zeta)$. Clearly, applied to pomtrees, the operation $\circ_\Sigma$ is not longer well-defined (it depends on the choice of u). Thus we restrict the usage of expressions of this form to those cases where the concrete result does not matter.

Finite Representations of Pomtrees. Let $\mathscr{I} \in \mathbf{I}(\Sigma)$ be an interpretation of Σ and let $s \in S_{\mathscr{I}}$. In this paragraph, we will show that if the set of states reachable from s if finite that there is a finite pomtree ζ which is *complete* in the sense that each state reachable from s is represented in ζ.

Let $s \in S_{\mathscr{I}}$ and let $\zeta \in \mathbf{T}_{\mathscr{I}}(s)$. We call an event $e \in E_\zeta$ a cut-off event if there is another event $e' \in E_\zeta$ with $e' <_\zeta e$ and $\zeta[e^{\downarrow}]_{\mathscr{I}}(s) =$

[3]Alternatively, one can work with sets of pomsets pre-closed under $\leqslant$, see Theorem 2.

$$
\begin{array}{lcl}
G \parallel C;R,x_q=0 & \vdash_{\mathrm{rec}_3} & U_q^x \parallel G \parallel C_c;R,x_q=0\\
& \vdash_{\mathrm{rec}_2} & !\mathrm{r}_{q,1,c}.\{?\mathrm{g}_{q,1}.?\mathrm{p}_{q,1}.\mathrm{u}_{q,1}.!\mathrm{e}_{q,1}+?\mathrm{d}_{q,1}\}.\mathbf{0}_1 \parallel G \parallel ?\mathrm{r}_{q,1,c}.\{!\mathrm{g}_{q,1}.(V_q^{x\leftarrow 1} \parallel C_c)+!\mathrm{d}_{q,1}.C_c\};R\setminus\{1\},x_q=0\\
& \xrightarrow[\mathrm{asyn}_2]{!\mathrm{r}_{q,1,c}\circ_\Sigma ?\mathrm{r}_{q,1,c}} & \{?\mathrm{g}_{q,1}.?\mathrm{p}_{q,1}.\mathrm{u}_{q,1}.!\mathrm{e}_{q,1}+?\mathrm{d}_{q,1}\}.\mathbf{0}_1 \parallel G \parallel \{!\mathrm{g}_{q,1}.(V_q^{x\leftarrow 1} \parallel C_c)+!\mathrm{d}_{q,1}.C_c\};R\setminus\{1\},x_q=0\\
& \xrightarrow[\mathrm{choice,asyn}_2]{\cdots\circ_\Sigma !\mathrm{g}_{q,1}\circ_\Sigma ?\mathrm{g}_{q,1}} & ?\mathrm{p}_{q,1}.\mathrm{u}_{q,1}.!\mathrm{e}_{q,1}.\mathbf{0}_1 \parallel G \parallel V_q^{x\leftarrow 1} \parallel C_c;R\setminus\{1\},x_q=0\\
& \vdash_{\mathrm{rec}_1} & ?\mathrm{p}_{q,1}.\mathrm{u}_{q,1}.!\mathrm{e}_{q,1}.\mathbf{0}_1 \parallel G \parallel x_q \leftarrow x_q+1.!\mathrm{p}_{q,1}.?\mathrm{e}_{q,1}.x_q \leftarrow x_q-1.\mathbf{0} \parallel C_c;R\setminus\{1\},x_q=0\\
& \xrightarrow[\mathrm{ass}]{\cdots\circ x_q\leftarrow x_q+1} & ?\mathrm{p}_{q,1}.\mathrm{u}_{q,1}.!\mathrm{e}_{q,1}.\mathbf{0}_1 \parallel G \parallel !\mathrm{p}_{q,1}.?\mathrm{e}_{q,1}.x_q \leftarrow x_q-1.\mathbf{0} \parallel C_c;R\setminus\{1\},x_q=1 \quad \cdots
\end{array}
$$

Figure 2: Example execution.

$\zeta[e'^{\downarrow}]_{\mathscr{I}}(s)$. With other words, the state that is reached after executing e has already been reached by the execution of smaller event e', i. e. is already represented in the history of e. A pomtree $\zeta \in \mathbf{T}_{\mathscr{I}}(s)$ is called *locally complete* at s if

LC$_1$. each event $e \in \max_{\leqslant_\zeta} E_\zeta$ is either a cut-off event or for all $a \in A_\Sigma$ it holds that $\zeta[e^{\downarrow}]_{\mathscr{I}} \circ_\Sigma a \notin \mathbf{L}_{\mathscr{I}}(s)$, and

LC$_2$. if $e \in E_\zeta$ is not a cut-off event and $\zeta[e^{\downarrow}] \circ_\Sigma a \in \mathbf{L}_{\mathscr{I}}(s)$ for some $a \in A_\Sigma$, then $\zeta[e^{\downarrow}] \circ_\Sigma a \leqslant \zeta$.

The pomtree ζ is called *globally complete* at s if for each state $s' \in S_{\mathscr{I}}$ we have: If there is some $w \in \mathbf{L}_{\mathscr{I}}(s)$ such that $w_{\mathscr{I}}(s) = s'$, then there is some configuration $v \in \mathbf{Cf}(\zeta)$ with $v_{\mathscr{I}}(s) = s'$.

We now show that local completeness implies global completeness.

THEOREM 3. *Let $\mathscr{I} \in \mathbf{I}(\Sigma)$, $s \in S_{\mathscr{I}}$ and let $\zeta \in \mathbf{T}_{\mathscr{I}}(s)$. Assume that the set $R_s =_{df} \{s' \in S_{\mathscr{I}} : \exists u \in \mathbf{L}_{\mathscr{I}}(s).s' = u_{\mathscr{I}}(s)\}$ if finite. If ζ is locally complete at s than it is globally complete at s.*

PROOF. (Sketch) Suppose some state $s' \in R_s$ that is not represented in ζ, i. e. $u_{\mathscr{I}}(s) \neq s'$ for all configurations $u \in \mathbf{Cf}(\zeta)$. Since $s' \in R_s$, there is some $w \in \mathbf{L}_{\mathscr{I}}(s)$ such that $w_{\mathscr{I}}(s) = s'$. Let $v \in \mathbf{Cf}(\zeta)$ be a maximum pomset with $v \leqslant w$. Then there is a cut-off event $e \in E_v$, i. e. there is some $e' <_\zeta e$ such that $\zeta[e^{\downarrow}]_{\mathscr{I}}(s) = \zeta[e'^{\downarrow}]_{\mathscr{I}}(s)$. Furthermore, we have $w = \zeta[e^{\downarrow}] \circ_\Sigma u$ for some pomset u. If follows that $\zeta[e'^{\downarrow}] \circ_\Sigma u \in \mathbf{L}_{\mathscr{I}}(s)$. But since ζ is local complete, there must be non-empty u_1, u' such $u = u_1 \circ_\Sigma u'$ and $\zeta[e'^{\downarrow}] \circ_\Sigma u_1 \leqslant \zeta$. Let u_1 be the maximum pomset with this property. Again, u_1 must contain a cut-off event. Thus we can repeat the above construction until the part of u that does not belong to ζ reduces to ε. $\square$

THEOREM 4. *Let $\mathscr{I} \in \mathbf{I}(\Sigma)$, $s \in S_{\mathscr{I}}$ and let $\zeta \in \mathbf{T}_{\mathscr{I}}(s)$. Assume that the set $R_s =_{df} \{s' \in S_{\mathscr{I}} : \exists u \in \mathbf{L}_{\mathscr{I}}(s).s' = u_{\mathscr{I}}(s)\}$ if finite, moreover, assume that* $\mathrm{en}_{\mathscr{I}}(s)$ *is finite for each $s \in R_s$. Then there is a finite locally complete pomtree ζ at s.*

PROOF. (Sketch) Since R_s if finite, there is no infinite sequence of pomtrees $\zeta_0 \leqslant \zeta_1 \leqslant \cdots \leqslant \zeta_i \leqslant \cdots$ without replicating the states that are represented in ζ_i. The finiteness of $\mathrm{en}_{\mathscr{I}}(s)$ ensures that ζ_i does not contain an infinite co-set. $\square$

The following theorem explains how to compute locally (and thus globally) complete pomtrees as result of a fixed point iteration. Since the result of each step of this iteration depends on the actual choice of an action $a \in A_\Sigma$, the step function Φ_s is defined on sets of pomtrees rather than on pomtrees.

THEOREM 5. *With the assumptions of Theorem 4, a locally complete finite pomtree can be effectively computed as the least fixed point of the function $\Phi_s : \mathscr{P}(\mathbf{Pt}(\Sigma)) \to \mathscr{P}(\mathbf{Pt}(\Sigma))$:*

$$
\Phi_s(X) \quad =_{df} \quad \big\{\zeta \circ_\Sigma a : a \in A_\Sigma \;\&\; \zeta \circ_\Sigma a \in \mathbf{T}_{\mathscr{I}}(s) \;\& \quad \mathrm{c}(\zeta) \cap <^{-1}_{\zeta\circ_\Sigma a}(e_a) = \varnothing\big\} \cup X \qquad (1)
$$

where we assume that $E_{\zeta\circ_\Sigma a} = E_\zeta \cup \{e_a\}$, and $e_a \notin E_\zeta$ (i. e. e_a is the event that is added to ζ if composed with a), and $c(\zeta)$ denotes the set of cut-off events of ζ. More precisely, there is some $n < \omega$ such that $\Phi_s^n(\{\varepsilon\}) = \mathbf{lfp}[\Phi_s, \{\varepsilon\}] = \{\xi\}$, and ξ is local complete at s.

Since the result of the iteration process is unique, we write $\zeta = \mathbf{lfp}[\Phi_s, \varepsilon]$. Fig. 5 shows an algorithm which computes a local complete pomtree according to Theorem 5.

Remark 4. Local completeness has been elaborated in the context of *safe Petri nets* by McMillan [16]. McMillan uses a more efficient definition of cut-off events that replaces the ordering $\leqslant_\zeta$ that is used to relate a cut-off event to its predecessor. In [9] it has been shown that McMillans ordering can be further improved, and that the ordering we use is most inefficient (in terms of the size of the resulting complete pomtree) in a hierarchy of orderings. We use this ordering for the sake of simplicity; the results of [9] can easily be applied in the context of our work. $\square$

4. MORPHISMS

Effective planning for large scaled distributed systems requires (a) an abstraction mechanism that prevents from taking into account an overwhelming amount of details, and (b) a notion of system composition or, more general, the embedding of subsystems into composite systems.

Proofs of lemmas and theorems of this section (and Section 6) can be found in [8], if not stated otherwise.[4]

Abstractions. Let Σ_1 and Σ_2 be distributed alphabets, and $\mathscr{I}_1 \in \mathbf{I}(\Sigma_1)$, $\mathscr{I}_2 \in \mathbf{I}(\Sigma_2)$ An *abstraction* from $\mathscr{I}_1$ to $\mathscr{I}_2$ is a pair of mappings $\alpha : A_{\Sigma_1} \rightharpoonup A_{\Sigma_2}$ and $\beta : S_{\mathscr{I}_1} \to S_{\mathscr{I}_2}$ such that (a) $\alpha(a)\ I_{\Sigma_2}\ \alpha(b) \Rightarrow a\ I_{\Sigma_1}\ b$; (b) if $a_{\mathscr{I}_1}(s_1) = s_2$ then if $\alpha(a)$ is defined then $\alpha(a)_{\mathscr{I}_2}(\beta(s_1)) = \beta(s_2)$ and if $\alpha(a)$ is not defined then $\beta(s_1) = \beta(s_2)$, and (c) $\varphi_{\mathscr{I}_2} \subseteq \beta(\varphi_{\mathscr{I}_1})$ We denote the fact that α, β form an abstraction by $[\alpha, \beta] : \mathscr{I}_1 \to \mathscr{I}_2$.

Remark 5. One can show that the pair of identity mappings on A_Σ and $S_{\mathscr{I}}$ is an abstraction, and that the component-wise composition of abstraction is again an abstraction. Thus, the class of interpretations of d-alphabets together with abstraction arrows forms a category. $\square$

[4]Actually, [8] uses a slightly different version of the notions presented here. These differences do however not jeopardize the results presented here.

Example 5. To obtain an abstraction for our running Example 2, use abstract states of the form $\langle P, R, K\rangle$, with the abstraction mapping $\beta : \langle P, R, \sigma\rangle \mapsto \langle P, R, \sigma, \max_{q\in Q} \sigma(x_q)\rangle$. We furthermore have to redefine the state invariant to $\varphi =_{\mathrm{df}} \{\langle P, R, \sigma, n\rangle : n < \min_{q\in Q} N_q\}$. We suppose that the concrete and the abstract d-alphabet Σ coincides, and put $\alpha =_{\mathrm{df}} \mathrm{id}_{A_\Sigma}$.

The reader may ask whether the set of abstract states is indeed more "simple" than that of concrete states. The idea here is that we use the state component n as a "quality indicator" to assess the suitability of a state. We furthermore encode the "correlation process" of n (i. e. $n = \max_{q\in Q} \sigma(x_q)$) directly into the abstract states by keeping the variable state σ, because the abstract version of an interpretation based on quality indicators needs to have a way to estimate the effect of a particular (abstract) system action to the values of these indicators. The resulting complicatedness of abstract states is thus due to the simplicity of the running example. $\square$

Let $[\alpha, \beta] : \mathscr{I}_1 \longrightarrow \mathscr{I}_2$ be an abstraction for interpretations $\mathscr{I}_1 \in \mathbf{I}(\Sigma_1)$, $\mathscr{I}_2 \in \mathbf{I}(\Sigma_2)$. A *refinement* $[\alpha, \beta]^{-1} : \mathscr{I}_2 \longrightarrow \mathscr{J}$ for $\mathscr{J} \in \mathbf{I}(\Sigma_1)$ is defined by: (a) $S_{\mathscr{J}} =_{\mathrm{df}} \{\beta^{-1}(s) : s \in S_{\mathscr{I}_2} \;\&\; \beta^{-1}(s) \neq \varnothing\}$, (b) $a_{\mathscr{J}}(\beta^{-1}(s_1)) = \beta^{-1}(s_2) \Leftrightarrow_{\mathrm{df}} \exists a' \in \alpha^{-1}(a).a'_{\mathscr{I}_2}(s'_1) = s'_2$, and (c) $\varphi_{\mathscr{J}} =_{\mathrm{df}} \{\beta^{-1}(s) : s \in \varphi_{\mathscr{I}_2}\}$.

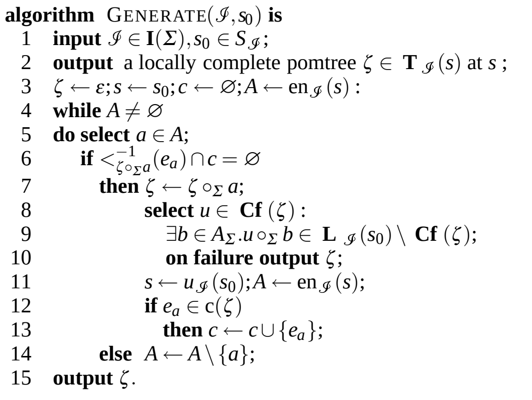
LEMMA 6. *Let $[\alpha, \beta] : \mathscr{I}_1 \longrightarrow \mathscr{I}_2$ be an abstraction and let $[\alpha, \beta]^{-1} : \mathscr{I}_2 \longrightarrow \mathscr{I}_3$ be the associated refinement. Then (a) there is an abstraction $[\hat{\alpha}, \hat{\beta}] : \mathscr{I}_1 \longrightarrow \mathscr{I}_3$; and moreover, (b) if $[\alpha_1, \beta_1] : \mathscr{I}_1 \longrightarrow \mathscr{J}$ and $[\alpha_2, \beta_2] : \mathscr{J} \longrightarrow \mathscr{I}_2$ be abstractions, then there is a unique abstraction $[\tilde{\alpha}, \tilde{\beta}] : \mathscr{I}_3 \longrightarrow \mathscr{J}$ that makes the diagram displayed right commute.*

$$\begin{tikzcd}
\mathscr{J} \arrow[dr, "{[\alpha_2,\beta_2]}"] & \\
\mathscr{I}_1 \arrow[u, "{[\alpha_1,\beta_1]}"] \arrow[r, "{[\alpha,\beta]}"] \arrow[d, "{[\hat{\alpha},\hat{\beta}]}"'] & \mathscr{I}_2 \arrow[dl, "{[\alpha,\beta]^{-1}}"] \\
\mathscr{I}_3 \arrow[uu, dashed, bend left=60, "{[\tilde{\alpha},\tilde{\beta}]}"] &
\end{tikzcd}$$

PROOF. Put $\hat{\alpha} =_{\mathrm{df}} \mathrm{id}_{A_{\Sigma_1}} \restriction \alpha^{-1}(A_{\Sigma_2})$ and $\hat{\beta}(s) =_{\mathrm{df}} \beta^{-1}(s)$. It is an easy exercise to show the required properties. $\square$

Thus the refined system $\mathscr{I}_2$ is more abstract than the original system $\mathscr{I}_1$, but it is nevertheless the "most concrete" system that can be obtained by any refinement process.

We now are going to investigate the relationship between abstractions and pomset languages of interpretations. For that, we first have to explain how a mapping on action sets of distributed alphabets can be extended to pomsets over these alphabets.



```
algorithm  GENERATE($\mathscr{I}, s_0$) is
input $\mathscr{I} \in \mathbf{I}(\Sigma), s_0 \in S_{\mathscr{I}}$;
output  a locally complete pomtree $\zeta \in \mathbf{T}_{\mathscr{I}}(s)$ at $s$ ;
$\zeta \leftarrow \varepsilon; s \leftarrow s_0; c \leftarrow \varnothing; A \leftarrow \mathrm{en}_{\mathscr{I}}(s)$ :
while $A \neq \varnothing$
do select $a \in A$;
    if $<^{-1}_{\zeta\circ_\Sigma a}(e_a) \cap c = \varnothing$
       then $\zeta \leftarrow \zeta \circ_\Sigma a$;
            select $u \in \mathbf{Cf}(\zeta)$ :
               $\exists b \in A_\Sigma . u \circ_\Sigma b \in \mathbf{L}_{\mathscr{I}}(s_0) \setminus \mathbf{Cf}(\zeta)$;
               on failure output $\zeta$;
            $s \leftarrow u_{\mathscr{I}}(s_0); A \leftarrow \mathrm{en}_{\mathscr{I}}(s)$;
            if $e_a \in \mathrm{c}(\zeta)$
               then $c \leftarrow c \cup \{e_a\}$;
       else  $A \leftarrow A \setminus \{a\}$;
output $\zeta$.
```

Figure 5: Pomset generation procedure

Let Σ_1, Σ_2 be distributed alphabets such that $\alpha : A_{\Sigma_1} \rightharpoonup A_{\Sigma_2}$ is a mapping. Then α is inductively extended to a mapping $\alpha^* : \mathbf{Ps}(\Sigma_1) \longrightarrow \mathbf{Ps}(\Sigma_2)$ as follows:

$$\begin{aligned}
\alpha^*(\varepsilon) &=_{\mathrm{df}} \varepsilon; \\
\alpha^*(u \circ_{\Sigma_1} a) &=_{\mathrm{df}} \begin{cases} \alpha^*(u) \circ_{\Sigma_2} \alpha(a), & \text{if } \alpha(a) \text{ is defined;} \\ \alpha^*(u), & \text{otherwise;} \end{cases} \\
&\qquad \text{for finite } u; \\
\alpha^*(u) &=_{\mathrm{df}} \bigvee \{\alpha^*(v) : v \leqslant u \;\&\; v \text{ is finite}\} \text{ for infinite } u.
\end{aligned}$$

for all $u \in \mathbf{Ps}(\Sigma_2)$ and $a \in A_{\Sigma_2}$.

The following lemma justifies the relationship of the notions of refinement and abstraction on distributed alphabets and interpretations by means of the pomset languages of interpretations.

LEMMA 7. *Assume $[\alpha, \beta] : \mathscr{I}_1 \longrightarrow \mathscr{I}_2$ is an abstraction. Then $\alpha^*(\mathbf{L}_{\mathscr{I}_1}(s)) \subseteq \mathbf{L}_{\mathscr{I}_2}(\beta(s))$ for all $s \in S_{\mathscr{I}_1}$.*

Embedding. The embedding operation provides us with a generic notion on how to understand a system component as part of a larger system, or the relationship of a system and its environment. Note that the composition of a number of system components can be understood as the embedding of each of these components into the composed system. Therefore, embeddings give us a generic notion of composition without assuming any specific composition operation such as synchronous or asynchronous communication.

Let $\mathscr{I}_1 \in \mathbf{I}(\Sigma_1)$ and $\mathscr{I}_2 \in \mathbf{I}(\Sigma_2)$ and let $f : A_{\Sigma_1} \longrightarrow A_{\Sigma_2}$ be an injective mapping and $g : S_{\mathscr{I}_2} \longrightarrow S_{\mathscr{I}_1}$ be a surjective mappings such that for all $a, b \in A_{\Sigma_1}$ (a) $f(a)\, I_{\Sigma_2}\, f(b) \Rightarrow a\, I_{\Sigma_1}\, b$, (b) $f(a)_{\mathscr{I}_2}(s_1) = s_2 \Rightarrow a_{\mathscr{I}_1}(g(s_1)) = g(s_2)$, and moreover (c) $\varphi_{\mathscr{I}_1} \subseteq \beta(\varphi_{\mathscr{I}_2})$. We write $(f, g) : \mathscr{I}_1 \longrightarrow \mathscr{I}_2$ in this case.

Remark 6. Since id_{A_Σ} and $\mathrm{id}_{S_{\mathscr{I}}}$ form an embedding of $\mathscr{I} \in \mathbf{I}(\Sigma)$ into itself, and moreover the component-wise composition of embeddings forms an embedding, the class of interpretations of d-alphabets together with embeddings as arrows is a category. $\square$

Example 6. In our example process algebra, $f = \mathrm{id}_{A_\Sigma}$ and $g : \langle U \parallel P, R, \sigma\rangle \mapsto \langle U, R, \sigma\rangle$ form an embedding, where U refers to all the terms "derivable" from U^x_q (compare Figure 2). $\square$

5. SUPERVISION

Supervision is based on observing the processing of some system, and on interacting if the system enters an undesired state (which could be an error state, a performance bottleneck, etc.). Interaction is done by restricting the possible behaviors of the system under supervision to those alternatives that will lead the system back into a desired state. Restricting behavioral alternatives bases on the disabling of actions of the system under supervision. But of course, not all actions can be controlled in this way. System actions like interrupts, timeouts, are beyond the control of a supervisor. If we consider a component of a larger environment, then from the point of the component, actions of the environment are not controllable (this may different if the environment is considered).

Hence, if Σ is a d-alphabet, we partition its alphabet set A_Σ into two disjoint sets $A_{\Sigma,\mathrm{c}}$ and $A_{\Sigma,\mathrm{uc}}$ of *controllable* and *uncontrollable* actions, respectively. Furthermore, we put $\Sigma_{\mathrm{c}} =_{\mathrm{df}} \langle A_{\Sigma,\mathrm{c}}, I_\Sigma \cap A_{\Sigma,\mathrm{c}} \times A_{\Sigma,\mathrm{c}}\rangle$, and $\Sigma_{\mathrm{uc}} =_{\mathrm{df}} \langle A_{\Sigma,\mathrm{uc}}, I_\Sigma \cap A_{\Sigma,\mathrm{uc}} \times A_{\Sigma,\mathrm{uc}}\rangle$.

Example 7. In our example, actions $!\mathrm{g}_{s,i}$, $!\mathrm{d}_{s,i}$, $!\mathrm{p}_{s,i}$ are controllable. The behavior of the user processes are not controllable, as well receiving events of messages from user processes. $\square$

Our approach is similar to that of *Supervisory Control Theory* (SCT) [23], with the difference that we do not assume "strong" control in the sense that a supervisor is active all the time, i. e. the the supervised system is not able to exhibit undesired alternatives at all. SCT has originally been presented using finite state machines as underlying system models. Recently, extensions to infinite state systems have been considered [3, 15, 10].

Controllability. A pomset $w \in \mathbf{Ps}^{\mathrm{w}}(\Sigma)$ is called *controllable* if $\{\tilde{e} : e \in \min_{\leqslant_w} E_w\} \cap A_{\Sigma,\mathrm{uc}} \neq \varnothing$. Note that if w is controllable, then it might have a suffix (i. e. some v such that th ere is an u with $w = u \circ_\Sigma v$) that is not controllable. On the other hand, uncontrollable pomsets may have controllable suffices.

Runs. A pomset $w \in \mathbf{Ps}^{\mathrm{w}}(\Sigma)$ is called a *run* of some $\mathscr{I} \in \mathbf{I}(\Sigma)$ at some $s \in S_{\mathscr{I}}$ if $w \in \mathbf{L}_{\mathscr{I}}(s)$ and either it is infinite or it is finite and moreover, $(w \circ_\Sigma a)_{\mathscr{I}}(s)$ is undefined for all $a \in A_\Sigma$.

Reactive Supervision Problem. Suppose some state s of a system $\mathscr{I} \in \mathbf{I}(\Sigma)$ that violates the state invariant $\varphi_{\mathscr{I}}$. The *reactive supervision problem* consists of the computation of maximum a pomtree ζ comprising of configurations u that, if executed by $\mathscr{I}$, will lead the system back into some state $s' \in \varphi_{\mathscr{I}}$, and moreover, if there is a pomset that is also executable at s but does not belong to the configurations of ζ (i. e. does not lead the system back into a desired state), then its execution can be prohibited by a supervisor.

More formally: Let $s \in S_{\mathscr{I}}$ such that $s \notin \varphi_{\mathscr{I}}$. A *reactive plan* for s is a finite pomtree $\zeta \in \mathbf{T}_{\mathscr{I}}(s)$ such that

R$_1$. $u_{\mathscr{I}}(s) \in \varphi_{\mathscr{I}}$ for all $u \in \mathbf{Cf}_\vee(\zeta)$;

R$_2$. $u_{\mathscr{I}}(s) \notin \varphi_{\mathscr{I}}$ for all $u \in \mathbf{Cf}(\zeta) \setminus \mathbf{Cf}_\vee(\zeta)$, and moreover,

R$_3$. for all $u \in \mathbf{Cf}(\zeta) \setminus \mathbf{Cf}_\vee(\zeta)$ it holds: If there is some pomset $v \in \mathbf{L}_{\mathscr{I}}(u_{\mathscr{I}}(s))$ such that $u \circ_\Sigma v \notin \mathbf{Cf}(\zeta)$, then v is controllable.

A valid plan ζ is called *most permissive* at s if for each valid plan $\xi \in \mathbf{T}_{\mathscr{I}}(s)$ we have $\zeta \leqslant \xi \Rightarrow \zeta = \xi$.

Example 8. In order to make the system of Example 2 "supervisable" we have to add some "management functions": We redefine the following equations:

$$\begin{aligned} U_q^x &=_{\mathrm{df}} \sum_{j \in J} !\mathrm{r}_{q,x,j}.\left\{?\mathrm{g}_{q,x}.(?\mathrm{p}_{q,x}.\mathrm{u}_{q,x}.!\mathrm{e}_{q,x} + ?\mathrm{d}_{q,x}) + ?\mathrm{d}_{q,x}\right\}.\mathbf{0}_x \\ V_q^x &=_{\mathrm{df}} x_q \leftarrow x_q + 1.(!\mathrm{p}_{q,x}.?\mathrm{e}_{q,x} + !\mathrm{d}_{q,x}).x_q \leftarrow x_q - 1.\mathbf{0}, \end{aligned}$$

Let $J = \{c\}$, $Q = \{q\}$, $N_q = 1$, and

$$\begin{aligned} U_i &=_{\mathrm{df}} \left\{?\mathrm{p}_{s,i}.\mathrm{u}_{s,i}.!\mathrm{e}_{s,i} + ?\mathrm{d}_{s,i}\right\}.\mathbf{0}_i \text{ and} \\ V_i &=_{\mathrm{df}} (!\mathrm{p}_{s,i}.?\mathrm{e}_{s,i} + !\mathrm{d}_{s,i}).x_s \leftarrow x_s - 1.\mathbf{0} \end{aligned}$$

for $i = 1,2,3$. Consider the concrete state $s = \left\langle P; R \setminus \{1,2,3\}, x_q \mapsto 3 \right\rangle$ with

$$P = \big\Vert_{i=1}^{3} U_i \parallel \big\Vert_{i=1}^{3} V_i \parallel C_c \parallel G$$

We have $\beta(s) = \left\langle P, R \setminus \{1,2,3\}, x_q \mapsto 3, 3 \right\rangle \notin \varphi$. A most permissive valid plan at $\beta(s)$ is

$$\bigvee_i \left(!\mathrm{d}_{s,i} \circ_\Sigma x_q \leftarrow x_q - 1 \circ_\Sigma \bigvee_{j \neq i} (!\mathrm{d}_{s,j} \circ_\Sigma x_q \leftarrow x_q - 1) \right)$$

for $i, j = 1,2,3$. $\square$

Pro-active Supervision Problem. Dually, we can define the *pro-active supervision problem* as the problem to determining all finite behavioral alternatives that lead to a state violating the state invariant and that cannot be prohibited by a supervisor. If formal terms: Let $s \in S_{\mathscr{I}}$ such that $s \in \varphi_{\mathscr{I}}$. A *violation prediction* for s is a finite pomtree $\zeta \in \mathbf{T}_{\mathscr{I}}(s)$ such that

P$_1$. $u_{\mathscr{I}}(s) \notin \varphi_{\mathscr{I}}$ for all $u \in \mathbf{Cf}_\vee(\zeta)$;

P$_2$. $u_{\mathscr{I}}(s) \in \varphi_{\mathscr{I}}$ for all $u \in \mathbf{Cf}(\zeta) \setminus \mathbf{Cf}_\vee(\zeta)$, and moreover,

P$_3$. for all $u \in \mathbf{Cf}(\zeta) \setminus \mathbf{Cf}_\vee(\zeta)$: If there is some pomset $v \in \mathbf{L}_{\mathscr{I}}(u_{\mathscr{I}}(s))$ such that $u \circ_\Sigma v \notin \mathbf{Cf}(\zeta)$, then v is controllable.

A violation prediction ζ is called *complete* at s if for each violation prediction $\xi \in \mathbf{T}_{\mathscr{I}}(s)$ we have $\zeta \leqslant \xi \Rightarrow \zeta = \xi$.

Solutions. To meet the assumptions of Theorem 5 assume that for a given state s the set $R_s =_{\mathrm{df}} \{s' \in \mathscr{I} : \exists u \in \mathbf{L}_{\mathscr{I}}(s).u_{\mathscr{I}}(s) = s'\}$ is finite, and moreover $\mathrm{en}_{\mathscr{I}}(s')$ is finite for all $s \in R_s$.

We now define—in addition to the functional Φ_s given by Equation (1)—a number of functionals acting on finite pomtrees:

$$\begin{aligned} \Psi_{s,\Sigma}(\zeta) &=_{\mathrm{df}} \bigvee \{u[E_u \setminus \leqslant_u(E)] : \\ &\qquad a \in A_\Sigma \mathbin{\&} u \in \mathbf{Cf}_\vee(\zeta) \mathbin{\&} u \circ_\Sigma a \notin \mathbf{Cf}_\vee(\zeta)\}, \\ &\qquad \text{where } E =_{\mathrm{df}} \max_{\leqslant_u} \lambda_u^{-1}(D_\Sigma(a)) \\ \Theta_{s,S}(\zeta) &=_{\mathrm{df}} \bigvee \min_{\leqslant} \{u \in \mathbf{Cf}(\zeta) : v_{\mathscr{I}}(s) \in S\} \\ \Omega_{\Sigma,S}(s) &=_{\mathrm{df}} \Theta_{s,S}\left(\mathbf{lfp}\left[\Psi_{s,\Sigma}, \mathbf{lfp}\left[\Phi_s, \varepsilon\right]\right]\right) \end{aligned}$$

THEOREM 6. *If $s \in S_{\mathscr{I}} \setminus \varphi_{\mathscr{I}}$ for some $\mathscr{I} \in \mathbf{I}(\Sigma)$, the $\Omega_{\Sigma_{\mathrm{uc}},\varphi_{\mathscr{I}}}(s)$ is a most permissive reactive plan for s.*

PROOF. To see that $\Omega_{\Sigma_{\mathrm{uc}},\varphi_{\mathscr{I}}}(s)$ is a reactive plan, recall from Theorem 4 that $\zeta = \mathbf{lfp}[\Phi_s, \varepsilon]$ is globally complete. $\Psi_{s,\Sigma_{\mathrm{uc}}}(\zeta)$ deletes all maximum uncontrollable events e from ζ including all immediate predecessors of e. Thus a situation in which e is executed cannot longer occur. Thus $\mathbf{lfp}[\Psi_{s,\Sigma_{\mathrm{uc}}}, \zeta]$ fulfills property (R$_3$). The final application of $\Theta_{s,\varphi_{\mathscr{I}}}$ ensures properties (R$_1$) and (R$_2$).

To prove maximum permissiveness of $\zeta = \Omega_{\Sigma_{\mathrm{uc}},\varphi_{\mathscr{I}}}(s)$ assume there is a valid plan ξ such that $\zeta < \xi$. Since $\mathbf{lfp}[\Phi_s, \varepsilon]$ is state complete, we can assume that $\xi \leqslant \mathbf{lfp}[\Phi_s, \varepsilon]$ is true. Then we have a configuration $u \in \mathbf{Cf}_\vee(\zeta)$ such that one of the following is true: (a) There is some $a \in A_\Sigma$ such that $u \circ_\Sigma a \in \mathbf{Cf}(\xi)$. This however violates property (R$_2$), and ξ is not a reactive plan. (b) So assume $u \circ_\Sigma a \notin \mathbf{Cf}(\xi)$. Then there is some pomset v such that there are $e \in \min_{\leqslant_v}(E_v)$, $e' \in E_v$, $\tilde{e} = a$, and $e \lessdot_v e'$, such that both a and $\tilde{e}'$ are controllable. But in this case a (more precisely, the event corresponding to a) would not have been removed from $\mathbf{lfp}[\Phi_s, \varepsilon]$ in the computation process of $\mathbf{lfp}\left[\Psi_{s,\Sigma_{\mathrm{uc}}}, \mathbf{lfp}[\Phi_s, \varepsilon]\right]$. $\square$

THEOREM 7. *If $s \in \varphi_{\mathscr{I}}$ for some $\mathscr{I} \in \mathbf{I}(\Sigma)$, the $\Omega_{\Sigma_{\mathrm{c}},S_{\mathscr{I}} \setminus \varphi_{\mathscr{I}}}(s)$ is a complete violation prediction for s.*

PROOF. Dualize the proof of Theorem 6. $\square$

6. PLANNING WITH ZOOMS

A Galois Connection. In order to employ the the Cousot & Cousot framework of AI, we need to find an appropriate Galois connection:

THEOREM 8. *Let $[\alpha, \beta] : \mathscr{I}_1 \to \mathscr{I}_2$ be an abstraction. Then $\alpha^*, \gamma : \mathscr{P}(\mathbf{L}_{\mathscr{I}_1}(s)) \rightleftharpoons \mathscr{P}(\mathbf{L}_{\mathscr{I}_2}(\beta(s)))$ is a Galois connection, where*

$\gamma : \mathbf{L}_{\mathscr{I}_2}(\beta(s)) \to \mathscr{P}(\mathbf{L}_{\mathscr{I}_1}(s))$, *and* $\gamma(u)$ *is defined to be the smallest set satisfying the following conditions:*

$$\varepsilon \in \gamma(\varepsilon)$$
$$u \in \gamma(v) \;\&\; a \in [a']_\alpha \Rightarrow u \circ_{\Sigma_1} a \in \gamma(v \circ_{\Sigma_2} a') \text{ for finite } u, v;$$
$$\bigvee_{C \in \mathrm{Mc}(X)} C \in \gamma(v) \text{ for infinite } u;$$

where $X =_{df} \bigcup \{\gamma(w) : w < v \;\&\; w \text{ is finite}\}$, *and* $\mathrm{Mc}(X)$ *denotes the set of all maximum chains in* X*, that is the set of all maximum subsets* $C \subseteq X$ *such that* $u, v \in C \Rightarrow u \leqslant v \vee v \leqslant u$.

PROOF. Assume $\alpha^*(X) \subseteq Y$. By Lemma 6 we have $\gamma(\alpha^*(X)) \subseteq \mathbf{L}_{[\alpha,\beta]^{-1}(\mathscr{I}_2)}(\beta(s))$, hence $X \subseteq \gamma(\alpha^*(X)) \subseteq \gamma(Y)$. Assume $X \subseteq \gamma(Y)$. Then $\alpha^*(X) \subseteq Y$ by Lemma 7. $\square$

Remark 7. Theorem 8 is formulated in terms of chains because $\gamma(v)$ yields a set of possible concretization of v that are not necessarily comparable by $\leqslant$. Thus in "jumping" from finite to infinite pomsets, the sets to build infinite joins need to be made explicit.

Zooms. We now consider the following situation. Let $(f,g) : \mathscr{I}_1 \to \mathscr{I}_2$ be an embedding and let $[\beta,\alpha] : \mathscr{I}_3 \to \mathscr{I}_1$ be an abstraction. The question arises whether we can use the embedding process (described by (f,g)) also to embed the refined version of $\mathscr{I}_1$ into (a refined version) of $\mathscr{I}_2$? Such an operation would provide us with a *magnifying glass*, a notion of local refinement or *zooming*. Of course, we are not looking for an arbitrary zooming operation but for a universal one in the sense that the resulting refined system is the most abstract one that embeds the interpretation $\mathscr{I}_3$.

THEOREM 9. *Let* $\mathscr{I}_i \in \mathbf{I}(\Sigma_i)$ *interpretations for* $i = 1,2,3$ *such that* $(f,g) : \mathscr{I}_1 \to \mathscr{I}_2$ *is an embedding and* $[\alpha,\beta] : \mathscr{I}_3 \to \mathscr{I}_1$ *is an abstraction. Then there is an interpretation* $\mathscr{I}_4 \in \mathbf{I}(\Sigma_4)$*, an embedding* $(\hat{f},\hat{g}) : \mathscr{I}_3 \to \mathscr{I}_4$*, and an abstraction* $[\hat{\alpha},\hat{\beta}] : \mathscr{I}_4 \to \mathscr{I}_2$ *such that for all interpretations* $\mathscr{I}_5$ *and all embedings* $(f',g') : \mathscr{I}_3 \to \mathscr{I}_5$ *and abstractions* $[\alpha',\beta'] : \mathscr{I}_5 \to \mathscr{I}_2$ *there is a uniquely defined abstraction* $[\tilde{\alpha},\tilde{\beta}] : \mathscr{I}_5 \to \mathscr{I}_4$*. In other words,* $[\tilde{\alpha},\tilde{\beta}]$ *is the only abstraction arrow that makes the diagram displayed right commute.*

$$\begin{tikzcd}
\mathscr{I}_1 \arrow[r, "(f,g)"] & \mathscr{I}_2 & \\
\mathscr{I}_3 \arrow[u, "{[\alpha,\beta]}"'] \arrow[r, "(\hat{f},\hat{g})"] \arrow[rrd, bend right, "(f',g')"'] & \mathscr{I}_4 \arrow[u, "{[\hat{\alpha},\hat{\beta}]}"] & \\
& & \mathscr{I}_5 \arrow[lu, dashed, "{[\tilde{\alpha},\tilde{\beta}]}"'] \arrow[luu, bend right, "{[\alpha',\beta']}"']
\end{tikzcd}$$

The following theorem states that zooms can be composed.

$$\begin{tikzcd}
\mathscr{I}_1 \arrow[r, "(f_1,g_1)"] & \mathscr{I}_2 \arrow[r, "(f_2,g_2)"] & \mathscr{I}_3 \\
\mathscr{I}_4 \arrow[u, "{[\alpha_1,\beta_1]}"'] \arrow[r, "(f_3,g_3)"] & \mathscr{I}_5 \arrow[u, "{[\alpha_2,\beta_2]}"'] \arrow[r, "(f_4,g_4)"] & \mathscr{I}_6 \arrow[u, "{[\alpha_3,\beta_3]}"']
\end{tikzcd}$$

THEOREM 10. *Consider the diagram show left. If the inner squares form two zooms, then the outer rectangle is also a zoom. If the right hand side inner squares and the outer rectangle are zooms, then the left hand side inner square is also.*

Planning with zooms. To prepare the results presented in this paragraph, we have to ensure that controllability remains invariant under abstractions and embeddings. Therefore, if $[\alpha,\beta] : \mathscr{I}_1 \to \mathscr{I}_2$ is an abstraction for $\mathscr{I}_1 \in \mathbf{I}(\Sigma_1)$ and $\mathscr{I}_2 \in \mathbf{I}(\Sigma_2)$, than we stipulate that $\alpha(A_{\Sigma_1,\mathrm{uc}}) \subseteq A_{\Sigma_2,\mathrm{uc}}$ does hold, since a more concrete event which in not controllable cannot made controllable by abstraction. If $(f,g) : \mathscr{I}_1 \to \mathscr{I}_2$. is an embedding, we have to assume

1. If a state $s \notin S_{\mathscr{E}}$ is detected, compute a most permissive valid plan ξ for s.

2. Stop if step (1) does not yield a result.

3. For each localized environment $\mathscr{E}_i$, compute a prefix $\zeta_i \leqslant \gamma(\xi)$ such that $\zeta_i \in \mathbf{T}_{\mathscr{E}_i}(s')$, where $s' \in \hat{\beta}^{-1}(s)$ is the state that is detected on the level of $\mathscr{E}_i$. Note that $s' \notin \varphi_{\mathscr{E}_i}$. We know that $u_{\mathscr{E}_i}(s') \in \varphi_{\mathscr{E}_i}$ for each configuration $u \in \mathbf{Cf}_\vee(\zeta_i)$, but ζ_i is not necessarily a valid plan, or most permissive.

4. Stop if step (3) does not yield a result.

Figure 6: Planning algorithm

$f(A_{\Sigma_1,\mathrm{c}}) \subseteq A_{\Sigma_2,\mathrm{c}}$, since actions that are controllable in $\mathscr{I}_2$ are not necessarily controllable in $\mathscr{I}_1$.

Consider again the zoom diagram displayed in Theorem 9. Let $\zeta \in \mathbf{T}_{\mathscr{I}_2}(s)$ be a finite pomtree for some state $s \in S_{\mathscr{I}_2}$. Since the converse of Lemma 7 does not hold, $\gamma_i(\zeta)$ is not necessary an element of $\mathbf{T}_{\mathscr{I}_4}(s')$ (where γ_i is the adjoint in the Galois connection $\hat{\alpha}_i^*, \gamma_i : \mathbf{L}_{\mathscr{I}_2}(s) \rightleftharpoons \mathbf{L}_{\mathscr{I}_4}(s)$ and $s' \in [s]_{\hat{\beta}}$). We however have:

THEOREM 11. *Assume the notions of Theorem 9. Let* $s \in S_{\mathscr{I}_4}$ *be a state such that* $s \notin \varphi_{\mathscr{I}_4}$ *and let* $\zeta \in \mathbf{T}_{\mathscr{I}_4}(s)$ *be a most permissive valid plan for* s*. If* $\xi \in \mathbf{T}_{\mathscr{I}_2}(\beta(s))$ *is a most permissive valid plan for* $\beta(s)$*, then* $\hat{\alpha}^*(\zeta) \leqslant \xi$.

The proof of the theorem is rather tedious, also the theorem itself is rather obvious. It bases on the idea that if ζ resolves a problem on a concrete level (i. e. in $\mathscr{I}_4$), than the problem on a abstract level (i. e. in $\mathscr{I}_2$). But since ξ is most permissive we obtain $\hat{\alpha}^*(\zeta) \leqslant \xi$.

In practice, we will apply zooms on a number of embedded components, i. e. we will use a families $\{[\alpha_i,\beta_i] : \mathscr{I}_i \to \mathscr{J}_i\}_{i \in I}$ and $\{(f_i,g_i) : \mathscr{J}_i \to \mathscr{E}\}_{i \in I}$, obtaining families of completions $\{[\hat{\alpha}_i,\hat{\beta}_i] : \mathscr{E}_i \to \mathscr{E}\}_{i \in I}$ and $\{(\hat{f}_i,\hat{g}_i) : \mathscr{I}_i \to \mathscr{E}_i\}_{i \in I}$ (compare figure right). In this case, we call the interpretation $\mathscr{E}$ a *global environment*, and the interpretations $\mathscr{E}_i$ *localizations* of $\mathscr{E}$. This situation can be iterated forming a tree structure of interpretations that are increasingly more concrete—and more localized—towards the leaves of the tree by letting each of the interpretations $\mathscr{I}_i$ the environment of another set of zoom diagrams.

$$\begin{tikzcd}
& & \mathscr{E} & \\
& \mathscr{J}_2 \arrow[ru] & \mathscr{J}_2 \arrow[u] & \cdots \\
& \mathscr{I}_2 \arrow[u] & \mathscr{I}_2 \arrow[u] & \cdots \\
\mathscr{J}'_1 \arrow[ru] & \mathscr{J}'_2 \arrow[u] & \cdots & \\
\mathscr{I}'_1 \arrow[u] & \mathscr{I}'_2 \arrow[u] & \cdots &
\end{tikzcd}$$

Theorem 11 provides us now (indirectly) with a planning algorithm utilizing zooms as shown in Figure 6. Again, an algorithm for violation prediction with zooms can be obtained by dualizing.

Planning with zooms is done top-down, which has the following implications: The choice of the abstractions $\{[\alpha_i,\beta_i] : \mathscr{I}_i \to \mathscr{J}_i\}_{i \in I}$ is very sensible. Choosing a version of $\mathscr{I}_i$ that is "too abstract" might render to many abstract action as uncontrollable because some of their concrete counterparts are not controllable, leading to the situation that a valid plan cannot be computed at all. On the other hand, using abstractions that are "to concrete" leads to complexity problems and to non-feasible planning algorithms.

What happens if the execution of the algorithm is terminated in Step (2) because of the fact that too many uncontrollable events exist? If $\mathscr{E}$ is in fact the root of out model tree, we have to conclude that we cannot resolve the problem situation. But an action that is uncontrollable in one component may be controllable in another one. Thus if there is a model level available *above* $\mathcal{E}$—say $\mathcal{E}'$—we can try to repeat the planning algorithm on this level. Since $\mathcal{E}'$ embeds other components besides $\mathcal{E}$, controllable actions of these components are available to resolve the problem situation on the higher level.

We conclude that a hierarchical structure of the system models that are used for supervision allows to resolve problems on a level that is as local as possible.

For pro-active planing, the situation is similar. Suppose there is a complete violation prediction ζ for some state $s \in S_{\mathcal{E}}$. Again, we have to "concretize" ζ into the localized environments $\mathcal{E}_i$ to determine whether ζ can be executed locally. We omit the details.

7. SUMMARY AND FURTHER WORK

In this paper, we have presented an approach to perform reactive as well as pro-active planning for distributed autonomic systems comprising components that exhibit operational models of themselves. Using the embedding and abstraction morphisms, it had been shown that there is a "most abstract" concretization of an abstracted system component under the embedding into some environment, or—using a more colorful language—it is possible to use component abstractions as "magnification glasses" to zoom into models without inventing details.

Reactive supervision has been defined as the problem to lead a system from an undesired state back into a desired one under the assumption of partial controllability. Pro-active supervision is about the prediction of problem states that cannot be prevented because of the lack of controllability.

We presented algorithms based on the Cousot & Cousot framework of Abstract Interpretation to solve both problems. We furthermore elaborated how these solutions can be applied in the context of a hierarchical system model leading to an iterative planning algorithm that uses the most local view to the system that is possible (under a given set of abstractions/embeddings).

A number of open questions remain, the most immanent are:

1. How to incorporate widening/narrowing operators for Abstract Interpretation (Remark 1)?

2. How to define suitable abstractions (embeddings are usually given by the structure of the SUS)? Is there a way to "adjust" abstractions that are to coarse or fine gained?

3. Predictions that incorporate additional probabilities (expressing how likely it is that an uncontrollable actions occurs) would be much more useful.

8. REFERENCES

[1] M. Arbib and E. Manes. *Arrows, Structures, and Functors: The Categorical Imperative*. Academic Press, New York, 1975.

[2] G. Birkhoff. *Lattice Theory*. American Mathematical Society, Providence, Rhode Island, 3rd edition, 1967.

[3] Y.-L. Chen and F. Lin. Safety control of discrete event systems using finite state machines with parameters. In *Proc. of American Control Conference 2001*, pages 975–980, Arlington, VA, June 2001.

[4] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, California, 1977. ACM Press, New York, NY.

[5] P. Cousot and R. Cousot. Abstract interpretation frameworks. *Journal of Logic and Computation*, 2(4):511–547, 1992.

[6] P. Cousot and R. Cousot. Comparing the Galois connection and widening/narrowing approaches to abstract interpretation, invited paper. In M. Bruynooghe and M. Wirsing, editors, *Proceedings of the International Workshop Programming Language Implementation and Logic Programming, PLILP '92,*, Leuven, Belgium, 13–17 August 1992, Lecture Notes in Computer Science 631, pages 269–295. Springer-Verlag, Berlin, Germany, 1992.

[7] P. H. Deussen. *Analyse verteilter Systeme mit Hilfe von Prozessautomaten*. PhD thesis, Brandenburg Technical Univ. of Cottbus, 2001. In German.

[8] P. H. Deussen. A mathematical framework for pervasive supervision. Cascadas project deliverable D2.1, Fraunhofer Institute for Open Communication Systems, 2006.

[9] J. Esparza, S. Römer, and W. Vogler. An improvement of McMillan's unfolding algorithm. In *TACAS*, pages 87–106, 1996.

[10] B. Gaudin and P. H. Deussen. Supervisory control on concurrent discrete event systems with variables. In *Proc. 2007 American Control Conference*, July 2007.

[11] R. Goldblatt. *Topoi, the Categorial Analysis of Logic*. North Holland Publishing Company, 1979.

[12] J. Grabowski. On partial languages. *Fund. Inform*, 4(2):427–498, 1981.

[13] J. Kephart and D. Chess. The vision of autonomic computing. *IEEE Computer*, 36(1):41–52, 2003.

[14] S. M. Lane. *Categories for the Working Mathematics*, volume 5 of *Graduate Texts in Mathematics*. Springer Verlag, New York, 1977.

[15] T. Le Gall, B. Jeannet, and H. Marchand. Supervisory control of infinite symbolic systems using abstract interpretation. In *Proc. of the 44th IEEE Conference on Decision and Control and Control (CDC'05) and European Control Conference (ECC 2005)*, Sevilla (Spain), December 2005.

[16] K. L. McMillan. A technique of state space search based on unfolding. *Formal Methods in System Design: An International Journal*, 6(1):45–65, January 1995.

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

[18] M. Nielsen, G. Plotkin, and G. Winskel. Petri nets, event structures and domains, Part I. *Theoretical Computer Science*, 13:85–108, 1981.

[19] V. Pratt. Modelling concurrency with partial orders. *International Journal of Parallel Programming*, 15(1):33–71, 1986.

[20] M. Smirnov. Autonomic communication—research agenda for a new communication paradigm. Company whitepaper, Fraunhofer Institute for Open Communication Systems, Berlin, Germany, 2004.

[21] P. H. Starke. Multiprocessor systems and their concurrency. *J. Inf. Process. Cybern. EIK*, 20(4):207–427, 1984.

[22] A. Tarski. A lattice.theoretical fixedpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:522–587, 1955.

[23] W. M. Wonham. Notes on control of discrete-event systems. Tech. rep. ECE 1636F/1637S, Department of Electrical and Computer Engineering, Univertsity of Toronto, Toronto, July 2003.