Towards a concurrency theory for supervisory control

Baeten, J.C.M.; van Hulst, A.C.; van Beek, D.A.; Markovski, J.

Published: 01/01/2012

Document Version
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

• A submitted manuscript is the author's version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

Citation for published version (APA):

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal?

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 05. Dec. 2018
Towards a Concurrency Theory for Supervisory Control

J.C.M. Baeten       A.C. van Hulst
D.A. van Beek       J. Markovski

ISSN: 1872-1567
In this paper we propose a process-theoretic concurrency model to express supervisory control properties. In light of the present importance of reliable control software, the current workflow of direct conversion from informal specification documents to control software implementations can be improved. A separate modeling step in terms of controllable and uncontrollable behavior of the device under control is desired. We consider the control loop as a feedback model for supervisory control, in terms of the three distinct components of plant, requirements and supervisor. With respect to the control flow, we consider event-based models as well as state-based ones. We study the process theory TCP$^*$ as a convenient modeling formalism that includes parallelism, iteration, communication features and non-determinism. Via structural operational semantics, we relate the terms in TCP$^*$ to labeled transition systems. We consider the partial bisimulation preorder to express controllability that is better suited to handle non-determinism, compared to bisimulation-based models. It is shown how precongruence of partial bisimulation can be derived from the format of the deduction rules. The theory of TCP$^*$ is studied under finite axiomatization for which soundness and ground-completeness (modulo iteration) is proved with respect to partial bisimulation. Language-based controllability, as the necessary condition for event-based supervisory control is expressed in terms of partial bisimulation and we discuss several drawbacks of the strict event-based approach. State-based control is considered under partial bisimulation as a dependable solution to address non-determinism. An appropriate renaming operator is introduced to address an issue in parallel communication. A case for automated guided vehicles (AGV) is modeled using the theory TCP$^*$. The latter theory is henceforth extended to include state-based valuations for which partial bisimulation and an axiomatization are defined. We consider an extended case on industrial printers to show the modeling abilities of this extended theory. In our concluding remarks, we sketch a future research path in terms of a new formal language for concurrent control modeling.
1 Introduction

Control software is nowadays a product of enormous complexity, while for the most part being engineered by human labor. Besides its complexity, it has various other difficulties. It is error-prone, has to meet sharp time-to-market deadlines, and has to improve continuously on quality and function. Another critical aspect is that control software often interacts with, and controls, all sorts of hardware and actuators. Therefore, any failure at this control level might have dangerous consequences. It is clear that all these responsibilities put high demands on the development of control software and, therefore, it is worthwhile to analyze the qualitative aspects of ongoing development cycles.

In the most common development workflows, software engineers write control software based on specification documents that contain informal requirements. This is a time-consuming and error-prone process, since the requirements are often ambiguous and, moreover they constantly change during the product development [16]. This issue in control software design gave rise to supervisory control theory of discrete-event systems [8, 27] where high-level supervisory controllers are synthesized automatically based upon formal models of the hardware and the control requirements.

1.1 Supervisory Control

In supervisory control we distinguish between two main components: 1) The machine or hardware device under control, and 2) The controller implemented as a software component. These components are connected via a feedback loop for supervisory control, to be referred to as the control loop in the remainder of this paper. It is illustrated in Figure 1. The supervisory controller observes the discrete-event behavior of the device under control by observing signals from ongoing activities. In practice, one may think of these signals as either being sent by sensors inside the device or as operations performed by actuators in the hardware itself. Based upon these signals, it decides which activities the machine is allowed to carry out, and sends back control signals to the actuators, which control the hardware. We work under the assumption that the supervisory controller can react sufficiently fast on every input from the machine, to provide the required control signal.

![Figure 1: The feedback loop for supervisory control. Behavior in the hardware device is observed by the software controller. Allowed behavior is communicated back to the hardware device as control signals.](image)

1.2 Modeling the Control Loop

We define the events as the set of operations carried out by the hardware device that is subject to control. The set of events is split into controllable and uncontrollable events. In practice, controllable events correspond to operations performed by hardware actuators inside the device under control. The operation to start up production in a particular
machine, for instance, is a controllable event. Uncontrollable events are sensor readings occurring inside the hardware device. An example of an uncontrollable event is a sensor detecting that an object has been stuck on a conveyor belt. A supervisory controller might disallow a controllable event, but it can never prevent an uncontrollable event from happening.

We further detail our model of the control loop and its underlying components by identifying three distinctive parts. The plant is a model of the observable behavior of the hardware device. A second key component are the requirements which specify the required behavior of the plant model. The third part is the supervisor. It models the controller which restricts the plant to behave according to the requirements. The control loop as shown in Figure 1 is transferred to this model to establish communication between the plant and the supervisor. We refer to the pair of plant and supervisor models, acting as a synchronized unit under control, as the supervised plant.

1. Event-based Supervisory Control

   The plant is modeled as a set of observable sequences, or traces, of events. Events that occur in the plant are observed by the supervisor as shown in Figure 1. The supervisor uses this information to detect the state of the plant by keeping track of a history of event execution. It then communicates allowed events back to the plant. As noted before, the supervisor must always allow an uncontrollable event. Therefore, the only way for the supervisor to prevent an uncontrollable event from happening, is to disallow a preceding controllable event that occurs earlier on in an observed trace. This approach is referred to as event-based supervisory control [2, 8, 27]. It is illustrated in Figure 2.

   ![Figure 2: The control loop for event-based supervisory control.](image)

In event-based supervisory control, the three components of plant, requirements and supervisor are expressed in terms of sequences of events. The plant consists of all observable event-sequences that can occur in the hardware device. The requirements state the sequences of events that model the required behavior of the device under control. The supervisor contains all event-sequences that are specified in the requirements, as well as all uncontrolled behavior that might occur in the plant. The supervised plant then models the control loop as well as its underlying components. For each event that is observed in the plant by supervisor, a decision is made to either allow or disallow it. If the event is part of a sequence of required behavior, or if it is uncontrollable, it is allowed by the supervisor. If it is controllable and not part of required behavior, it is disallowed. This implies that the only way for the supervisor to prevent an uncontrollable event from happening, is to disallow a controllable event that occurs earlier in a sequence of plant behavior.

2. State-based Supervisory Control

   In this approach the event-based model of the control loop is extended with state observation [10, 19] by the supervisory controller. A state-based controller observes the current state that is communicated, or emitted by the plant. The supervisor then communicates the allowed events that can occur in the observed state back
to the plant. The model of the control loop for state-based supervisory control is illustrated in Figure 3.

![Figure 3: Control loop for state-based supervisory control.](image)

In state-based supervisory control, the plant, requirements and supervisor are similar models compared to the event-based situation, but extended by state-observation. The plant consists of event sequences together with information on states in which events occur. The requirements list the desired events on a per-state basis. The supervisor includes the formulated requirements as well as uncontrollable behavior, which should be allowed regardless of state. A supervised plant then models the following. The current state of the plant is observed by the supervisor. Information from the requirements determines the list of allowed events in this state, which is then communicated back to the plant. This models the control loop for the state-based situation.

1.3 Motivation

The purpose of this paper is twofold. First, we would like to address the issue of informal specifications for control software. The creation of a formal model of the supervised plant is a valuable step towards the improvement of control software quality. It allows the informal requirements to be formulated in an abstract, yet precise way. This model can then be used as a verified design specification for the control software, that adheres to the required control properties.

The second contribution of this paper is on supervisory control for non-deterministic plants. We argue that the trace-based approach essentially impedes progress in supervisory control theory. If a certain state in the plant has two outgoing transitions that are labeled by the same event, the supervisor cannot disallow a future event that occurs in only one of these traces. To address this issue, we define a theory that is suitable to model the control problem for non-deterministic plant-specifications. We apply this formalism to model an industrial case in this paper.

1.4 Process Theoretic Approach

We argue that a process-theoretic [1] specification of plant, requirements and supervisor provides the necessary levels of abstraction as well as formal precision. Process theories as described in [1] provide a formal description of discrete-event behavior, including termination. Furthermore, communication of state/event observations and control signals, as well as restrictions on behavior can all be expressed in such theories. The use of a refinement relation that is similar to bisimulation equivalence [1] can be used to formally state the control problem. To address the issue of non-deterministic plant-specifications, we extend the aforementioned process theory by constructs for state-observation as well as state-based control. Process-theoretic expressions are paired with Boolean valuations that contain information regarding the current state of the plant as propositions. For
the specification of the supervisor, conditional expressions are employed to (dis)allow events, based upon the evaluation of a guarding formula within the Boolean valuation that contains the state-information.

1.5 Previous Work

This article is an extension of previous work by the authors. In [6] and [5] the employed process theories are defined as well as the partial bisimulation refinement. This work contains a more in-depth investigation into these process theories, partial bisimulation and the interplay between these two concepts. We extend the work on event-based supervisory control theory to a sound and complete axiomatization for the underlying process theory. Aspects of a state-based theory from [5] are studied in-depth in this work. This leads to a better understanding of the foundations of the industrial case from [23] that we have included in this paper within the new context of the state-based process theory.

1.6 Overview

The remainder of this paper is set up as follows. We give an in-depth overview of related work in the supervisory control field. This encompasses work that relies on similar, process-theoretic approaches as well as work that studies the supervisory control problem from a different point of view. This is followed by a section on notations and language theory which defines the necessary preliminaries upon which the process-theoretic formalism is based. We continue with a section on language-based controllability that details how the controllability property can be expressed in terms of the previously defined notions of language theory. The next section defines an event-based process theory, which is one of the two central formalisms in this paper for modeling the plant, requirements and supervisor. Syntax and semantics are defined for this theory in a comparable way to [1]. We continue with the definition of the partial bisimulation preorder and prove several of its properties. An important contribution among these is a proof concerning derivation of precongruence properties, with respect to partial bisimulation, based upon the format of the deduction rules. This preorder is used to state the control problem in terms of process-theoretic expressions. The event-based process theory is further studied by means of a finite axiomatization for which proofs of soundness and completeness are provided. We extend this theory to include data and states. A new state-based definition of partial bisimulation is given and its properties are studied. The next section defines controllability for the two process-theories in a formal way. We show how the controller and the supervised plant can be defined in terms of our theories. Two case studies are provided in the following sections. In a smaller study on event-based supervisory control, we model an automated guided vehicle. The second case, which is considerably larger, models an industrial printer using state-based process theory. We end the article with our concluding remarks as well as a look at future developments.

2 Related Work

An introduction to supervisory control theory for discrete event systems can be found in the book by Cassandras [8]. A standard work on supervisory control is the paper by Ramadge and Wonham [27], which introduces the distinction between controllable and uncontrollable events to identify the sufficient and necessary conditions for existence of a supervisor. The notion of partial bisimulation as a behavioral relation suitable to define controllability was introduced in a coalgebraic approach to supervisory control [28].
essentially states that controllable events should be simulated, while uncontrollable events should be bisimulated. This spawned new investigations into process algebraic approaches for supervisory control theory [6, 2].

Non-deterministic automata are not disallowed in [27], but the semantics remains in terms of accepted languages. Although non-determinism provides greater modeling convenience by enabling abstract specifications [1], it introduces complications when controllability is defined as a language-based property. Several attempts have been made to address the issue of supervisory control for non-deterministic systems. A trace-based approach under bisimulation equivalence can be found in [32] and [10], while supervisory control is studied in [14] and [25] in terms of failure-based semantics.

Approaching supervisory control via failure trajectories and refusal sets, which are an extension of the concept of failure semantics, has been studied before [15]. In this work, a prioritized synchronization operator is employed to define controllability on process-theoretic specifications. This allows for a clear specification of plant-supervisor communication and ensures non-blockingness. The failure trajectories model and the operator for compositional prioritized synchronization are further studied in [14, 18]. A different approach is taken in [25], where a refinement relation based on failure semantics characterizes non-deterministic behavior.

Even though it is argued in [9] that refinements for failure and bisimulation semantics have similar properties, we consider (bi)simulation to be an elegant notion to capture non-determinism [31, 1]. The use of (bi)simulation as a refinement is also proposed in [21, 30], where partial observation induces non-determinism. Moreover, there exist efficient partitioning algorithms for minimization modulo (bi)simulation [12], already applied in the deterministic setting to optimize supervisor synthesis [7]. A different approach to address supervisory control for non-deterministic systems is to consider sets of deterministic systems as non-deterministic specifications. In this case, controllability of all underlying deterministic systems induces controllability of the non-deterministic system [26].

State-controllability [10, 32] induces language controllability for deterministic plant-specifications. This notion is, however, very restrictive as a plant may not be state-controllable with respect to itself, even though a supervisor enabling all events always exists [10]. If non-determinism is modeled as a choice between unobservable events [14], a definition of state-controllability can be build upon the notion of partial observability. A related approach to handle non-deterministic plant-specifications is to use state-tree representations as in [20]. However, this approach is not suitable when only partial observations of states are available for control. The work in [29] discusses the usage of abstractions for the definition of non-blocking supervisors, where partial observation may be present.

3 The Process Theory TCP∗

In this section we present the process theory TCP∗, which has a rich syntax, enabling it to model a variety of problems in a clear way. Terms in this theory can be used to model the different components in the supervisory control setup. While communicating actions represent the information flow between components, thereby completing the model of the control loop. Synchronizing actions are used to model allowance or denial of plant behaviour by the supervisory controller.

We introduce a number of preliminary notions in language theory and process algebra that are required to lay the foundations of TCP∗. We define a finite data alphabet $\mathcal{D}$
and a finite set $\mathcal{H}$ of communication channels. For each $c \in \mathcal{H}$, we define $\mathcal{A}_c = \{c^m?_nd \mid m, n \in \mathbb{N}, m + n > 0, d \in \mathcal{D}\}$, where $c^m?_nd$ is a generic communication action \cite{1} consisting of $m$ send actions and $n$ receive actions. We use the following, abbreviated notations: $c^1?_1d$, $c^1d$, $c?_1d$ and $c?d$ for $c^1?_1d$, $c^1d$, $c?_1d$ and $c?d$. Intuitively, these events denote respectively that datum $d$ is received, sent or communicated along channel $c$. We further define $\mathcal{A} = \cup \mathcal{A}_c \in \mathcal{H}$. We will use the following definition to denote all actions relying on the same communication channel:

**Definition 3.1.** We use $B \subseteq \mathcal{H}$ to indicate that there exists an $\mathcal{H}' \subseteq \mathcal{H}$ such that $B = \cup_{c \in \mathcal{H}'\mathcal{H}_c}$

This notation will be convenient when we have to handle arbitrary subsets of $\mathcal{A}$ that have to contain all communications sent over a number of channels.

We form traces of events $(a_0, a_1, \ldots, a_n) \in \mathcal{A}^*$ in a standard manner (see for instance \cite{17}), where $\mathcal{A}! = \{(a_0, a_1, \ldots, a_n) \mid n \in \mathbb{N}\}$. We use $\epsilon = (a_0, a_1, \ldots, a_n)$ for $a_i \in \mathcal{A}$ and $n = 0$ to denote the unique empty trace. If $t_a = (a_0, a_1, \ldots, a_m)$ and $t_b = (b_0, b_1, \ldots, b_n)$, then we use $t_a \cdot t_b = (a_0, a_1, \ldots, a_m, b_0, b_1, \ldots, b_n)$ to denote the concatenation of traces $t_a$ and $t_b$.

We partition the set of events $\mathcal{A} = \mathcal{C} \cup \mathcal{U}$ into controllable events $\mathcal{C}$ and uncontrollable events $\mathcal{U}$. This is a strict partition, events cannot be both controllable and uncontrollable. We only consider the situation in which the controllability aspect of an event does not change.

**Definition 3.2.** The set of terms $\mathcal{T}$ of the process theory TCP* is generated by the following grammar:

\[
P \Rightarrow 0 \mid 1 \mid a.P \mid P + P \mid P \parallel P \mid P^* \mid \partial E(P) \mid P.P \mid P \parallel P
\]

where $a \in \mathcal{A}$ and $E \subseteq \{c^m?_n \mid c \in \mathcal{C}, m, n \in \mathbb{N}\}$. We give short and intuitive explanations of the constants and operators in $\mathcal{T}$.

The constant process $0$ denotes inaction or deadlock. The constant process $1$ denotes successful termination. For each action $a \in \mathcal{A}$, the process corresponding to the term $a.P$ executes the action $a$ and it continues behaving as $P$. The binary operator $+\!+$ denotes alternative composition. The process term $p + q$ denotes a non-deterministic choice for a process that can either behave as $p$ or as $q$. The sequential composition operator $\cdot\!\cdot$ first executes the left-hand side process and then, upon successful termination of this operand, executes the right-hand side process. The binary operator $\parallel\!\parallel$ denotes a parallel composition of two terms, that is able to perform interleaving as well as synchronous communication. The term $p \parallel q$ can behave as 1) a unilateral step of either $p$ or $q$, while the other operand remains unchanged or, 2) a synchronous communication step in both $p$ and $q$, upon which data is communicated over a specified channel. The operator $(\_)^*$ or Kleene star is used to express iteration. It unfolds with respect to sequential composition. The term $p^*$ either terminates or behaves as $p$, followed by $p^*$. The unary operator $\partial E(p)$ encapsulates the process $p$ in such a way that all (incomplete) communication actions (i.e. $c^1d$ and $c^1d$), are blocked for all data, so that bilateral communication is enforced. An example might be illustrative in this regard.

**Example 3.3.** Suppose that we want to enforce communication between $k$ processes on channel $c$, then we have to choose $E$ in the following way:

\[E = \{c^m?_n \mid 0 < m + n < k, c \in \mathcal{C}\}\]

If $E$ is chosen in such a way, it includes all generic communication actions (excluding data) in such a way that it becomes possible to communicate between at most $k$ process terms.
The two last operators of \( \cdot | \cdot \) (synchronous parallel composition) and \( \cdot \parallel \cdot \) (left merge) are only required for the ground-completeness proof for the given axiomatization. The operator \( \cdot | \cdot \) only behaves synchronously between parallel terms. It cannot interleave as the operator \( \cdot \parallel \cdot \) can do. This is illustrated in the operational semantics in Figure 4.

The left merge operator behaves as a non-commutative, unilateral parallel composition. The only allowed behavior of \( \cdot \parallel \cdot \) is a transition in \( p \).

### 3.1 Operational Semantics

We give structural operational semantics for each process term \( p \in T \). The semantics is given in terms of labeled graphs with successful termination, labeled transition systems for short, modulo a behavioral equivalence. A labeled transition system, defined by \( G = (N \subseteq T, \mathcal{A}, p \in N, \downarrow, \rightarrow) \), has a set of nodes \( N \), which are connected by transitions labeled by \( \mathcal{A} \) and defined by the relation \( \rightarrow \subseteq T \times \mathcal{A} \times T \). The root node is \( p \in N \) and some nodes are marked by the predicate \( \downarrow \subseteq T \) as having the successful termination option. For a process term \( p \in T \) we have a labeled graph of the form \((T, \mathcal{A}, \downarrow, \rightarrow)\), where \( \downarrow \) and \( \rightarrow \) are defined using by the structural operational rules given in Figure 4. We will use infix notation and write \( p \downarrow \) for \( p \in N \) and \( p \rightarrow q \) for \( (p, a, p') \in \rightarrow \). For \( p \in T \) we construct a labeled transition graph \( G \) with \( p \) as a root node in the following way. If \( q \in T \) is a node in \( G \) and if \( q' \in T \) and \( a \in \mathcal{A} \) exist such that \( q \rightarrow a q' \) then we add \( q' \) as a node in \( G \) and an edge from \( q \) to \( q' \) labeled by \( a \).

![Figure 4: Operational semantics for TCP*](image)

These operational rules define termination options as well as steps in the transition relation.

We briefly comment on the operational rules. We first discuss rules regarding termination and then continue with the operational rules that define the transition relation. Rule 1 states that the constant process 1 always terminates. Rules 2 and 3 show that if one component of the alternative composition has a termination option, then the alternative composition has a termination option as well. Rule 4 states that termination of sequential composition depends on termination of both operands. Rule 5 shows that any term under iteration can always terminate. Intuitively this corresponds to iterating zero times. Rules 6 and 7 state that termination of synchronous as well as asynchronous parallel composition depends on termination of both operands. Rule 8 shows how termination of any encapsulated term depends on termination of the term itself.
Rule 9 states that action prefixes induce outgoing transitions with the same label. The rules 10 and 11 enable a nondeterministic choice between the alternatives of the alternative composition. Rules 12 and 13 semantically define how the sequential composition behaves: 1) The left operand may terminate in which case a continuation is expected from the right operand, or 2) the left side may take an independent step. Rule 14 shows how the Kleene-star unfolds with respect to sequential composition. The parallel composition operator relies on the rules 15, 16, and 17 for unilateral behavior as well as parallel communication. The rule 18 defines how the left merge operator unfolds in one step into the parallel composition and the rule 19 states that the operator \( \_ | \_ \) unfolds into parallel composition after one synchronous step. We define the last rule 20 for encapsulation where semantics of this operator depends on the set of encapsulated actions. In [1] more information is available about these operators.

### 3.2 Partial Bisimilarity

We previously highlighted one of the fundamental requirements of the control loop: a supervisor cannot disallow an uncontrollable event [27]. If process theoretic terms are considered modulo an equivalence on structural behavior, such as bisimulation, [1], we lose the ability to express this requirement. A solution to this problem was proposed in [28] as the partial bisimulation preorder, which restricts full bisimulation to a specified set of events. This is illustrated in Figure 5 for the terms \( a.b.c.1 \) and \( a.(b.c.1 + d.e.0) \). The partial bisimulation \( R \) enforces bisimulation of the events \( a, b, c \), while the the events \( d \) and \( e \) are only simulated. This corresponds precisely to the distinction in supervisory control between controllable and uncontrollable events [27]. We formally define partial bisimulation for the process theory TCP\(^*\) in Definition 3.4.

![Figure 5: Partial bisimulation. Transitions with labels in \( B \) are bisimulated, while transitions labeled by \( A \setminus B \) are only simulated.](image)

**Definition 3.4.** Let \( R \) be a relation on \( T \). Then \( R \) is a partial bisimulation with respect to the bisimulation action set \( B \subseteq_H A \) if for all \( p, q \in T \) such that \( (p, q) \in R \) the following holds:

1. \( p \downarrow \) if and only if \( q \downarrow \);

---

9 The Process Theory TCP\(^*\)
2. for all $p' \in \mathcal{T}$ and $a \in \mathcal{A}$ such that $p \trans{a} p'$, there exists $q' \in \mathcal{T}$ such that $q \trans{a} q'$ and $(p', q') \in R$; and

3. for all $q' \in \mathcal{T}$ and $b \in \mathcal{B}$ such that $q \trans{b} q'$, there exists $p' \in \mathcal{T}$ such that $p \trans{b} p'$ and $(p', q') \in R$.

We say that the process term $p$ is partially bisimilar to $q$ with respect to the bisimulation set $\mathcal{B}$, notation $p \bisim_B q$, if there exists a partial bisimulation $R$ with respect to $\mathcal{B}$ such that $(p, q) \in R$. If $p \bisim_B q$ and $q \bisim_B p$, then we say that $p$ and $q$ are mutually partially bisimilar (with respect to $\mathcal{B}$) and we write $p \leftrightarrow_B q$. When clear from the context, we will omit $\mathcal{B}$.

It can be easily shown that partial bisimilarity is a preorder relation \[^{28}\]. Also, it is not difficult to prove that mutual partial bisimilarity is an equivalence relation \[^{28}\]. Note that if the bisimulation set $\mathcal{B}$ is empty, then the partial bisimilarity preorder coincides with the standard (strong) similarity preorder and the partial bisimilarity equivalence coincides with standard similarity equivalence \[^{11}\]. When $\mathcal{B} = \mathcal{A}$, the partial bisimilarity preorder becomes strong bisimilarity. We have the following property that characterizes the dependence on the bisimulation set $\mathcal{B}$.

**Lemma 3.5.** If $p \bisim_B q$, then $p \bisim_C q$ for every $C \subseteq \mathcal{B}$.

**Proof.** Let $p \bisim_B q$ given by $R$. If, for arbitrary $(p, q) \in R$ it holds that $q \trans{c} q'$ for $c \in C \subseteq \mathcal{B}$ and $q' \in \mathcal{T}$ then there exists a $p' \in \mathcal{T}$ such that $p \trans{b} p'$ and $(p', q') \in R$. The remaining conditions for partial bisimulation remain unaltered and we may therefore conclude that $R$ is also a partial bisimulation such that $p \bisim_C q$. \(\square\)

### 3.3 Derivation of Precongruence

We show that partial bisimilarity $\bisim_B$ for $B \subseteq \mathcal{H} \mathcal{A}$ is a precongruence with respect to the operators of $\text{TCP}^*$. In \[^{24}\], it is shown that for operational rules in the $\text{tyft}$ format, congruence with respect to bisimulation can be automatically derived. We give a precongruence proof in a relatively general way. We use $C(T)$ to denote the closed terms in $\mathcal{T}$.

We need two definitions (see \[^{24}\]).

**Definition 3.6.** An operational rule is in process-$\text{tyft}$ format if it is of the form:

\[
\begin{array}{l}
\{ t_i \trans{a} y_i \mid i \in I \} \\
\quad \quad f(x_0, \ldots, x_{n-1}) \trans{a} t'
\end{array}
\]

Where $I$ is a set of indices, $a \in \mathcal{A}$, $f$ is an $n$-ary operator in the process theory $\text{TCP}^*$, $t' \in \mathcal{T}$ and $x_0, \ldots, x_{n-1}, y_i$ are all distinct process variables. For all $i \in I$, we have $a_i \in \mathcal{A}$ and $t_i \in \mathcal{T}$. We define $X_p = \{x_0, \ldots, x_{n-1}\}$, the set of process variables in the source of the conclusion. Additionally, we have $Y_p = \{y_i \mid y \in I\}$, the set of variables in the target of the premises. We have as an additional requirement that the sets $X_p$ and $Y_p$ are disjoint.

We say that a transition system specification is in process-$\text{tyft}$ format if all its deduction rules are in process-$\text{tyft}$ format.

We define the *acyclicity* of the variable dependency graph:
Definition 3.7. For every deduction rule depending on premises $P_1, \ldots, P_N$ with their respective sets of process variables $S_i$ in the source of the premise $i$ and $T_i$ in the target of the premise $i$, we define a variable dependency graph in the following way:

1. Every variable in $\cup_i (S_i \cup T_i)$ is a node.
2. There exists an edge $(v_i, v_j)$ if there exists an $i$ such that $v_i \in S_i$ and $v_j \in T_i$.

We define this graph to be *acyclic* if it does not contain any cycles. We define the *rank*($x$) for each process variable $x$ as the maximum length of a backward chain starting in $x$ in the variable dependency graph. The rank of a premise is the rank of its target variable.

We need the definition of closure of a relation under precongruence:

**Definition 3.8.** Let $R \subseteq C(T) \times C(T)$. We define the relation $\bar{R} \subseteq C(T) \times C(T)$ to be the smallest reflexive precongruence on $C(T)$ such that the relation $R$ is contained in $\bar{R}$. We formally define $\bar{R}$ as follows:

1. $\bar{R}$ is reflexive;
2. $R \subseteq \bar{R}$;
3. $(f(p_0, \ldots, p_{n-1}), f(q_0, \ldots, q_{n-1})) \in \bar{R}$ for every $n$-ary $f \in T$, and all $p_0, \ldots, p_{n-1}, q_0, \ldots, q_{n-1} \in C(T)$ such that $(p_i, q_i) \in \bar{R}$ for $0 \leq i < n$.

**Lemma 3.9.** Let $R \subseteq C(T) \times C(T)$ and $t \in T$. For any two process substitutions $\sigma$ and $\sigma'$ such that $(\sigma(x), \sigma'(x)) \in \bar{R}$, where $x$ is a process variable in $t$, we have that $(\sigma(t), \sigma'(t)) \in \bar{R}$.

**Proof.** By induction on the structure of the process term $t$. See [13].

**Theorem 3.10.** Partial bisimulation is a precongruence for each of the operators in $T$.

**Proof.** Let $f$ be an $n$-ary process function and let $p_i, q_i$ be closed process terms for $0 \leq i < n$. Suppose that $B \subseteq \bar{R}$ $A$ and $p_i \not\preceq B q_i$ for $0 \leq i < n$. This means that for every $0 \leq i < n$ there exists a partially bisimilar relation $R_i$ that witnesses these partial bisimulations. Let $\bar{R} = \cup_{i=0}^n R_i$ be the union of these relations. It is quite obvious that $\bar{R}$ is a partial bisimulation, with respect to $B$, as well. We show that the relation $\bar{R}$ contains the pair $(f(p_0, \ldots, p_{n-1}), f(q_0, \ldots, q_{n-1}))$ and that it adheres to the partial bisimulation property as well. The first claim follows directly from the definition of $\bar{R}$.

In [24] it is shown for $(p, q) \in \bar{R}$ that for $a \in A, p' \in C(T)$ such that $p \xrightarrow{a} p'$, there exists a $q' \in C(T)$ such that $q \xrightarrow{a} q'$ and $(p', q') \in \bar{R}$. We complete this proof here for partial bisimulation by showing that for $b \in B$ such that $q \xrightarrow{b} q'$, there exists a $p' \in C(T)$ such that $p \xrightarrow{b} p'$ and $(p', q') \in \bar{R}$.

We show this by induction on the depth of the proof of a transition. The proof for the induction base is omitted because it is a direct instance of the proof of the induction step where there are no premises.

For the induction step we distinguish between three cases based on the definition of $\bar{R}$. In case $(p, q) \in \bar{R}$ due to reflexivity or due to $(p, q) \in R \subseteq \bar{R}$ we directly have...
the result and do not require any inductive reasoning. For the remaining case, we have 
\( p = f(p_0, \ldots, p_{n-1}) \) and \( q = f(q_0, \ldots, q_{n-1}) \) for some \( p_0, \ldots, p_{n-1}, q_0, \ldots, q_{n-1} \) such that 
\( (p_i, q_i) \in \tilde{R} \) for all \( 0 \leq i < n \). The last step in the deduction tree for the transition of \( q \)
is due to the application of a deduction rule of the following form:

\[
\frac{\{t_i \overset{b_i}{\rightarrow} y_i \mid i \in I\}}{f(x_0, \ldots, x_{n-1}) \overset{b}{\rightarrow} t'}
\]

To prove the result for partial bisimulation, we require an additional condition for every deduction rule: If \( b \in B \) then \( \forall i \in I \, b_i \in B \).

This means that there exists a process substitution \( \sigma \) such that \( \sigma(x_i) = q_i \) for all \( 0 \leq i < n \) and \( \sigma'(t') = q' \). Furthermore, for each \( i \in I \) there exists a proof of \( \sigma(t_i) \overset{b_i}{\rightarrow} \sigma(y_i) \) with smaller depth. For each process variable \( x \) in the variables of \( t_i \) of each premise \( t_i \overset{a_i}{\rightarrow} y_i \), it holds that \( \text{rank}(x) < \text{rank}(y_i) \). We define the process substitution \( \sigma' \) as follows:

\[
\sigma'(x) = \begin{cases} 
q_i & \text{if } x = x_i, \\
\sigma(x) & \text{if } x \notin X_p \cup Y_p.
\end{cases}
\]

Note that this process substitution remains to be define for variables in \( Y_p \). This definition will be extended in the remainder of this proof. For all \( i \in I \) such that \( r_i = \text{rank}(P_i) = \text{rank}(y_i) \) of premise \( P_i \), we show three essential properties:

(A) \( (\sigma(t_i), \sigma'(t_i)) \in \tilde{R}; \)

(B) \( \sigma'(t_i) \overset{b_i}{\rightarrow} \sigma'(y_i); \)

(C) \( (\sigma(y_i), \sigma'(y_i)) \in \tilde{R}. \)

Again, we do not show the proof of the induction base \( (r_i = 0) \) as it is an instance of the proof of the induction step. For the inductive part, assume that \( r_i \geq 1 \). Let \( t_i \overset{b_i}{\rightarrow} y_i \) for some \( i \in I \) be a premise of rank \( r_i \). We first show property (A). Let \( x \) be a variable in \( t_i \), and distinguish between the following cases:

1. If \( x \in X_p \) then \( x = x_i \) for some \( 0 \leq i < n \). From the definition of \( \sigma' \) we have that \( \sigma(x) = \sigma(x_i) = q_i \) and \( \sigma'(x_i) = p_i \) and, as \( (p_i, q_i) \in \tilde{R} \), we have that \( (\sigma(x), \sigma'(x)) \in \tilde{R}. \)

2. If \( x \notin X_p \) and \( x \notin Y_p \) then we have \( \sigma(x) = \sigma'(x) \) and because inclusion of identity in \( \tilde{R} \) we directly have \( (\sigma(x), \sigma'(x)) \in \tilde{R}. \)

3. If \( x \in Y_p \) then \( x = y_j \) for some \( j \in I \). Because in this case \( \text{rank}(y_j) < \text{rank}(y_i) \) we have by the induction hypothesis \( (\sigma(y_j), \sigma'(y_j)) \in \tilde{R}. \) However, as \( x = y_j \), we also have \( (\sigma(x), \sigma'(x)) \in \tilde{R}. \)

Because of the fact that \( (\sigma(x), \sigma'(x)) \in \tilde{R} \) for all variables \( x \) in \( t_i \), we have that \( (\sigma(t_i), \sigma'(t_i)) \in \tilde{R} \) by Lemma 3.9 which proves property (A).
As we have a proof of smaller depth for $\sigma(t_i) \xrightarrow{b} \sigma(y_i)$, by the induction hypothesis, we have the existence of a process term $p'_i$ such that $\sigma'(t_i) \xrightarrow{b} p'_i$ and $(\sigma(y_i), p'_i) \in \bar{R}$. We define $\sigma'(y_i) = p'_i$. Observe that this shows existence of an appropriate process term $\sigma'(y_i)$. This gives us $\sigma'(t_i) \xrightarrow{b} \sigma'(y_i)$ and $(\sigma(y_i), \sigma'(y_i)) \in \bar{R}$, which proves properties (B) and (C).

The proof is completed using process substitution $\sigma'$ for the aforementioned deduction rule. Observe that we have $\sigma'(f(x_0, \ldots, x_{n-1})) = f(y_0, \ldots, y_{n-1}) = p$. In property (B) we have shown that there are proofs for all premises using the process substitution $\sigma'$. Then, according to the same deduction rule and using $\sigma'$ instead of $\sigma$, we have $\sigma'(f(x_0, \ldots, x_{n-1})) \xrightarrow{b} \sigma'(t)$. Since $\sigma'(f(x_0, \ldots, x_{n-1})) = f(y_0, \ldots, y_{n-1}) = p$ we have $p \xrightarrow{b} \sigma'(t')$.

It remains to be shown that $(\sigma(t'), \sigma'(t')) \in \bar{R}$. By Lemma 3.9, we only have to show that $(\sigma(x), \sigma'(x)) \in \bar{R}$ for variables $x$ in $t'$. The proof is completed by considering the following three cases:

1. If $x \in X_p$ then $x = x_i$ for some $0 \leq i < n$. We have that $\sigma(x_i) = p_i$ and $\sigma'(x_i) = p_i$, and we already know that $(p_i, q_i) \in \bar{R}$ and $x_i = x$. Therefore, we have that $(\sigma(x), \sigma'(x)) \in \bar{R}$.

2. If $x \not\in X_p$ and $x \not\in Y_p$ then we have $(\sigma(x), \sigma'(x)) \in \bar{R}$ because $\sigma(x) = \sigma'(x)$ and because of inclusion of identity in $\bar{R}$.

3. If $x \in Y_p$ then $x = y_j$ for some $j \in I$ then we have $(\sigma(y_j), \sigma'(y_j)) \in \bar{R}$ by property (C). However, as $x = y_j$ we also have $(\sigma(x), \sigma'(x)) \in \bar{R}$.

\[\square\]

**Corollary 3.11.** Partial bisimulation is a precongruence with respect to the operators in $TCP^*$ because of Theorem 3.10 and by observing that the deduction rules in Figure 4 adhere to the tyft format.

Now, we can build the standard algebraic term model [1] for the process theory $TCP^*$ by using partial bisimilarity as the underlying behavioral congruence.

**Definition 3.12.** The term algebra $\mathbb{P}(TCP^*)$ is given by

\[\mathbb{P}(TCP^*) = (T, 0, 1, \alpha, \ldots, \cdot, \cdot \cdot, \cdot \cdot \cdot, \partial_F(\cdot), \cdot^*, \cdot | \cdot, \cdot \parallel \cdot).\]

The term model of $TCP^*$ is given by the quotient algebra $\mathbb{P}(TCP^*)/_{\sim_IT}$.

### 3.4 Axiomatization

We give a finite equational axiomatization of the process algebra $TCP^*$ with respect to partial bisimulation. This provides insight into the theory and could serve as a first step towards automated tools for equational reasoning and rewriting of terms.

The parallel composition operator $\cdot \parallel \cdot$ is expressive enough to model a large variety of problems, and therefore the need for additional parallel operators is not immediately clear. However, to obtain a finite axiomatization for a theory that includes $\cdot \parallel \cdot$, we include the operators $\cdot | \cdot$ and $\cdot \parallel \cdot$ as well [1].

13 The Process Theory $TCP^*$
Let $x, y, z$ be terms in $T$, $a \in A$ and $B \subseteq H A$:

| $x \leq_B a.x + y$ if $a \notin B$ | $(PB)$ | $x \parallel y =_B x \parallel y + y \parallel x + x \parallel y$ | $(M)$ |
| $x + y =_B y + x$ | $(A1)$ | $0 \parallel x =_B 0$ | $(LM1)$ |
| $x + (y + z) =_B (x + y) + z$ | $(A2)$ | $1 \parallel x =_B 0$ | $(LM2)$ |
| $x + x =_B x$ | $(A3)$ | $a.x \parallel y =_B a.(x \parallel y)$ | $(LM3)$ |
| $(x + y) \cdot z =_B x \cdot z + y \cdot z$ | $(A4)$ | $(x + y) \parallel z =_B x \parallel z + y \parallel z$ | $(LM4)$ |
| $x \cdot (y + z) =_B (x \cdot y) \cdot z$ | $(A5)$ | | |
| $x + 0 =_B x$ | $(A6)$ | $0 \parallel x =_B 0$ | $(CM1)$ |
| $0 \cdot x =_B 0$ | $(A7)$ | $(x + y) \parallel z =_B x \parallel z + y \parallel z$ | $(CM2)$ |
| $x \cdot 1 =_B x$ | $(A8)$ | $1 \parallel 1 =_B 1$ | $(CM3)$ |
| $1 \cdot x =_B x$ | $(A9)$ | $a.x \parallel 1$ | $(CM4)$ |
| $(a.x) \cdot y =_B a.(x \parallel y)$ | $(A10)$ | $(c_{1+m}^t_1 d).x \parallel (c_{1+m}^0 n d).y =_B$ | |
| | | $(c_{1+m}^t_1 k_{k+n} d).x \parallel (c_{1+m}^0 n d).y =_B 0$ | |
| | | if $c \neq c' \lor d \neq d'$ | |
| $x^* =_B x \cdot x^* + 1$ | $(IT1)$ | | |
| $(x + 1)^* =_B x^*$ | $(IT2)$ | $x \parallel y =_B y \parallel x$ | $(SC1)$ |
| $(x + y)^* =_B x^*. (y \cdot (x + y)^* + 1)$ | $(IT3)$ | $x \parallel 1 =_B x$ | $(SC2)$ |
| $x =_B y \cdot x + z \land y \Rightarrow x =_B y^* \cdot z$ | $(RSP^*)$ | $1 \parallel x + 1 =_B 1$ | $(SC3)$ |
| $\partial_E(1) =_B 1$ | $(D1)$ | $(x \parallel y) \parallel z =_B x \parallel (y \parallel z)$ | $(SC4)$ |
| $\partial_E(0) =_B 0$ | $(D2)$ | $(x \parallel y) \parallel z =_B x \parallel (y \parallel z)$ | $(SC5)$ |
| $\partial_E(c_{1+m}^t_1 n d).x =_B 0$ for $c_{1+m}^t_1 n \in E$ | $(D3)$ | $(x \parallel y) \parallel z =_B x \parallel (y \parallel z)$ | $(SC6)$ |
| $\partial_E(c_{1+m}^0 n d).x =_B (c_{1+m}^t_1 n d).\partial_E(x)$ | $(D4)$ | $(x \parallel 0 =_B x \parallel 0$ | $(SC7)$ |
| for $c_{1+m}^t_1 n \notin E$ | | | |
| $\partial_E(x + y) =_B \partial_E(x) + \partial_E(y)$ | $(D5)$ | | |

Table 1: Axioms for TCP*: A sound equational axiomatization with respect to partial bisimulation.

As mentioned in [4], it is not known whether the addition of axiom RSP* provides a complete axiomatization of a process algebra that includes deadlock and successful termination. According to the best knowledge of the authors, this can be considered a hard problem. Therefore, we avoid this issue and only take into account the subset of terms in which iteration does not appear when we consider the completeness proof.

Most of the axioms in table 1 are given for strict bisimulation, besides the axiom (PB), which is specific for partial bisimulation. It is important to note that if an axiom is sound with respect to strict bisimulation, we have soundness for $\leq_B$ for any $B \subseteq H A$ due to Lemma 3.5. If we have $\leq_B$ and $\geq_B$ we denote this by $=_B$. For the names of the various axioms, we will mostly use the same abbreviations as in [1].

### 3.5 Soundness and Ground-Completeness

We prove soundness with respect to partial bisimulation for the axioms in Table 1. Due to Lemma 3.5, we directly obtain soundness for partial bisimulation if we have a proof for soundness with respect to strict bisimulation. In [1], soundness for bisimulation is shown for all axioms in Table 1, except for the axiom (PB), which we will show here.
Lemma 3.13. For \( x, y \in T \), \( a \in A \) and \( B \subseteq \mathcal{H} \ A \) we have that \( x \leq a \cdot y + x \) implies that \( x \leq_B a \cdot y + x \), provided that \( a \not\in B \).

Proof. We choose \( R = \{ (x, a \cdot y + x) \} \cup \{ (z, z) \mid z \in T \} \) and observe that \( x \downarrow \iff a \cdot y + x \downarrow \). If \( x \xrightarrow{c} x' \) for \( c \in A \) then \( a \cdot y + x \xrightarrow{c} x' \) with \( (x', x') \in R \). If \( a \cdot y + x \xrightarrow{c} x' \) for \( c \in B \) then it must be the case that \( x \xrightarrow{c} x' \) because the only outgoing transition of \( a \cdot y \) is labeled by \( a \not\in B \). We again conclude that \((x', x') \in R\). \(\square\)

We prove ground-completeness for the given set of axioms with respect to partial bisimulation. This proof is given for the set \( T \) without the terms containing iteration. To a certain extent, we follow the proof given in [1] for strong bisimulation. We show that all terms (excluding those containing iteration) can be rewritten to strictly bisimilar terms generated by the following grammar:

\[
P \Rightarrow 0 \mid 1 \mid a.P \mid P + P \quad \text{for } a \in A
\]

We will denote the set generated by this grammar as \( T_{(+)} \). In the next three lemmas and their corresponding proofs, we use the appropriate elimination axioms to rewrite the non-iterative terms in \( T \) to equal terms in \( T_{(+)} \). Note that the operational semantics in Figure 4 and the previous soundness proofs still apply for \( T_{(+)} \).

Lemma 3.14. For \( p, q \in T_{(+)} \), there exists an \( r \in T_{(+)} \) such that \( p \cdot q = r \).

Proof. The proof is by structural induction on \( p \). If \( p \equiv 0 \) then we choose \( r \equiv 0 \) such that \( 0 \cdot q = 0 \) by \( A7 \). If \( p \equiv 1 \) then we pick \( r \equiv q \) such that \( 1 \cdot q = q \) by \( A9 \). For the action prefix case, let \( p \equiv a.p' \) for some \( a \in A \) and \( p' \cdot q = r' \). We then have \((a.p') \cdot q = a.(p' \cdot q) = a.r' \equiv r \). For the last case we have \( p \equiv p_1 + p_2 \) with \( p_1 \cdot q = r_1 \) and \( p_2 \cdot q = r_2 \). We rewrite using axiom \( A4 \): \((p_1 + p_2) \cdot q = p_1 \cdot q + p_2 \cdot q = r_1 + r_2 \equiv r \) with \( r_1, r_2 \in T \). \(\square\)

We have introduced two additional operators for the parallel composition in order to have proper elimination rules. This proof relies on mutually dependent induction and therefore we prove this property for the three operators \( \parallel \parallel \parallel \) and \( \parallel \parallel \parallel \) in parallel:

Lemma 3.15. We prove the following lemmas:

- For \( p, q \in T_{(+)} \), there exists an \( r \in T_{(+)} \) such that \( p \parallel q = r \).
- For \( p, q \in T_{(+)} \), there exists an \( r \in T_{(+)} \) such that \( p \mid q = r \).
- For \( p, q \in T_{(+)} \), there exists an \( r \in T_{(+)} \) such that \( p \parallel q = r \).

Proof. We start with an inductive proof for the left merge operator. If \( p \equiv 0 \) or \( p \equiv 1 \) then we choose \( r \equiv 0 \) or \( r \equiv 1 \) respectively (axioms LM1 and LM2). For the inductive case where \( p \equiv a.p' \), assume that we have an \( r' \in T_{(+)} \) such that \( p' \parallel q = r' \). According to axiom LM3, we choose \( r \equiv a.r' \) such that \( a.p' \parallel q = a.(p' \parallel q) = a.r' \). For the alternative composition, assume that \( p_1 \parallel q = r_1 \) and \( p_2 \parallel q = r_2 \). We have \((p_1 + p_2) \parallel q = p_1 \parallel q + p_2 \parallel q = r_1 + r_2 \) by induction.

We proceed with the synchronous parallel composition operator \( p \parallel q \). If \( p \equiv 0 \) then we may choose \( r \equiv 0 \) because of axiom CM1. For the case where \( p \equiv 1 \), we proceed by
induction towards \( q \). For \( q \equiv 0 \) we have \( 1 \mid 0 = 0 \mid 1 = 0 \) by symmetry and CM1. If \( q \equiv 1 \) then directly \( r \equiv 1 \) by CM3. For the case \( q \equiv a.q' \), we have \( 1 \mid a.q' = a.q' \mid 1 = 0 \) so we have to choose \( r \equiv 0 \). If \( q \equiv a.p + q_1 + q_2 \) such that \( 1 \mid q_1 = r_1 \) and \( 1 \mid q_2 = r_2 \) then, by induction and CM1, we have \( 1 \mid (q_1 + q_2) = 1 \mid q_2 + 1 \mid q_1 = r_1 + r_2 \). We continue with the case where \( p \equiv a.p' \) for some \( p' \in \mathcal{T}^{(+)}. \) The cases where \( q \equiv 0 \) and \( q \equiv 1 \) have been discussed previously in the symmetric argument. If \( q \equiv b.q' \) for some \( b \in \mathcal{A} \), we have to distinguish between the two cases that correspond to the axioms CM5 and CM6. If \( a \equiv c_{l,m} r d \) and \( b \equiv c_{m,n} d \) then we proceed inductively as \( a.p' \mid b.q' = (c_{l+m,n} d).\langle p' \parallel q' \rangle. \) In the other case, we choose \( r \equiv 0 \) and apply axiom CM6. If \( q \equiv q_1 + q_2 \) such that \( a.p' \mid q_1 = r_1 \) and \( a.p' \mid q_2 = r_2 \), then we have \( a.p' \mid (q_1 + q_2) = a.p' \mid q_1 + a.p' \mid q_2 = r_1 + r_2 \). The next case to consider is when \( p \equiv p_1 + p_2 \). Due to symmetry, we have previously shown that for \( q \equiv 0 \), \( q \equiv 1 \) and \( q \equiv a.q' \), an appropriate \( r \) can be found. The remaining case where \( q \equiv q_1 + q_2 \) can be solved by using the axiom CM2 twice via a simple induction.

The last proof to consider is that of the parallel composition. Let \( p \parallel q = p \parallel q \parallel p + p \mid q \) such that \( p \parallel q = r_1 \), \( q \parallel p = r_2 \), and \( p \mid q = r_3 \), then \( p \parallel q = r_1 + r_2 + r_3 \) by induction using the previous proofs.

**Lemma 3.16.** For \( p \in \mathcal{T}^{(+)}, \) there exists for each \( E \subseteq \{ c_{l,m} r d \mid c \in \mathcal{C}, m, n \in \mathbb{N} \} \) an \( r \in \mathcal{T}^{(+) \mathcal{A}} \) such that \( \partial_E(p) = r. \)

**Proof.** We consider the various following cases inductively. Let \( p \equiv 0 \), then choose \( r \equiv 0 \) (axiom D2) and if \( p \equiv 1 \), then we pick \( r \equiv 1 \) (axiom D1). If \( c_{l,m} r d \in E \) then we may choose \( r \equiv 0 \) according to axiom D3. On the other hand, of \( c_{m,n} r d \not\in E \) then we have \( \partial_E(c_{l,m} r d.p) = (c_{l,m} r d).\partial_E(r) \) by induction and axiom D4. For the alternative composition we assume the existence of \( r_1 \) and \( r_2 \) such that \( \partial_E(p_1) = r_1 \) and \( \partial_E(p_2) = r_2 \). We rewrite using axiom D5: \( \partial_E(p_1 + p_2) = \partial_E(p_1) + \partial_E(p_2) = r_1 + r_2. \) 

**Lemma 3.17.** The set of axioms in Table 1 is ground-complete with respect to partial bisimulation.

**Proof.** We prove ground-completeness using normal forms as explained in [1] and using Lemma 3.14, 3.15, and 3.16. Using these lemmas, we are able to write every term \( p' \in \mathcal{T} \) as a term \( p \in \mathcal{T}^{(+) \mathcal{A}} \) such that \( p' \equiv_{\mathcal{A}} p. \) In [1] it is outlined how this \( p \in \mathcal{T}^{(+) \mathcal{A}} \) can be rewritten as the normal form:

\[
p = \sum_{i \in M} a_i p_i[+1] \equiv p_{\mathcal{A}} \text{ where } p_i \in \mathcal{T}^{(+) \mathcal{A}}, a_i \in \mathcal{A}\backslash \mathcal{B}, \text{ and } M \subseteq \mathbb{N}
\]

Where \([+1]\) is present if and only if \( p \) contains the optional summand 1 for successful termination. We write the normal form of \( q \in \mathcal{T} \) in an equivalent way:

\[
q = \sum_{j \in N} c_j q_j[+1] \equiv q_{\mathcal{A}} \text{ where } q_j \in \mathcal{T}^{(+) \mathcal{A}}, c_j \in \mathcal{A}\backslash \mathcal{B}, \text{ and } N \subseteq \mathbb{N}
\]

We have \( p \leftrightarrow_{\mathcal{A}} p_{\mathcal{A}} \) and \( q \leftrightarrow_{\mathcal{A}} q_{\mathcal{A}}. \) Because of Lemma 3.5 it now follows that \( p \leftrightarrow_{\mathcal{B}} p_{\mathcal{A}} \) and \( q \leftrightarrow_{\mathcal{B}} q_{\mathcal{A}}. \) This allows us to conclude that \( p_{\mathcal{A}} \leq_{\mathcal{B}} q_{\mathcal{A}} \) if and only if \( p \leq_{\mathcal{B}} q. \) We proceed by induction on the total number of constants and action prefixes on these normal forms. We have the following base cases:

- \( p_{\mathcal{A}} \equiv 0 \leq_{\mathcal{B}} 0 \equiv q_{\mathcal{A}} (A3 \leftrightarrow, A3). \)
- \( p_{\mathcal{A}} \equiv 1 \leq_{\mathcal{B}} 1 \equiv q_{\mathcal{A}} (A3 \leftrightarrow, A3). \)

Because \( p_{\mathcal{A}} \leq_{\mathcal{B}} q_{\mathcal{A}}, \) there exists an \( R \subseteq \mathcal{T} \times \mathcal{T} \) such that \( R \) is a partial bisimulation and \((p_{\mathcal{A}}, q_{\mathcal{A}}) \in R. \) If \( p_{\mathcal{A}} \) contains a 1 summand then \( q_{\mathcal{A}} \) contains it as well, and vice versa,
because of the termination requirements in Definition 3.4. Assume that \( p_{nf} \xrightarrow{a} p'_{nf} \) for some \( a \in A \) and \( p'_{nf} \in \mathcal{T}_i \). Then, according to the operational rules in Table 4 there exists a summand \( a_i, p_i \) of \( p_{nf} \) for some \( i \in M \) such that \( a \equiv a_i \) and \( p_i \equiv p'_{nf} \). By Definition 3.4 there exists a summand \( c_j, q_j \) of \( q_{nf} \) such that \( c_j \equiv a \) and \( (p_i, q_j) \in R \) for a \( j \in N \).

We now have that \( p_i \preceq B q_j \) and by the induction hypothesis \( p_i \preceq B q_j \). Thus, there exists \( N' \subseteq N \) such that for every \( i \in M \) there exists a \( j \in N \) such that \( a_i, p_i \preceq B c_j, q_j \). Vice versa, for every \( j \in N \) such that \( c_j \in B \) there exists an \( i \in M \) such that \( a_i, p_i \preceq B c_j, q_j \).

Let \( K = N' \cup \{ j \mid c_j \in B, j \in N' \} \) and divide \( q_{nf} \) into \( q_{nf} = q_{nf}' + q_{nf}'' \) in such a way that \( q_{nf}' \) contains the summands that are prefixed by an action in \( B \) or that have an index in \( N' \), with \( q_{nf}'' \) containing the remaining summands. That is,

\[
q_{nf}' = \sum_{k \in K} c_k \cdot q_k \quad \text{and} \quad q_{nf}'' = \sum_{n \in N' \setminus K} c_n \cdot q_n
\]

Note that \( q_{nf}'' \) contains only summands prefixed by actions that are not in \( B \). Now we may conclude that \( p = B p_{nf} \preceq B q_{nf} \). We apply axiom PB for the summands \( c_n, q_n \) of \( q_{nf}'' \). This results in \( q_{nf}' \preceq B q_{nf}' + q_{nf}'' = B q_{nf} = B q \). Hence, \( p \preceq B q \), which completes the proof.

\[\square\]

## 4 Controllability

In this section we define controllability in the context of the control loop of the plant-supervisor model. The intuitive idea behind controllability is the following: if we observe a desired trace in the plant followed by an uncontrollable event then the control requirements cannot request that this uncontrollable event should be disabled after allowing that trace. Several constructs have been devised to express controllability for discrete-event systems. We define language-based controllability [27] and state-based controllability [32] in this section and point out inherent problems in previous attempts to express the notion of controllability in these ways. We show how partial bisimulation is better suited to handle non-deterministic plant models and we show how this definition can be used together with the process theory TCP∗.

In the context of supervisory control synthesis, extended definitions of this notion of controllability have been studied. For instance, types of controllability that prevent deadlock and livelock. To this end, marked states are added to the automata to specify non-blocking behavior [27].

We extend our previous definitions of language theory in section 3 in a standard manner [17]. We define the reflexive transitive closure \( \rightarrow^* \) of the step-relation \( \rightarrow \) as follows: For \( p, p' \in \mathcal{T} \) we have \( p \xrightarrow{t} p \) and if \( t, t' \in A^* \) with \( t = a \cdot t' \) then \( p \xrightarrow{t} p' \) if and only if there exists a \( p'' \in \mathcal{T} \) such that \( p \xrightarrow{a} p'' \) and \( p'' \xrightarrow{t'} p' \). If \( G \) is a labeled transition system with starting node \( p \in \mathcal{T} \) then we define the language \( L(G) = \{ t \in A^* \mid \exists p' \in \mathcal{T} \left( p \xrightarrow{b} p' \right) \} \).

We say that a language is \textit{prefix closed} if \( L = \overline{L} \) where \( \overline{L} = \{ t \mid \exists t' \in A^* \left( t \cdot t' \in L \right) \} \). By \( L \cdot L' \) we denote the concatenation of the two languages \( L \) and \( L' \). That is \( L \cdot L' = \{ t \cdot t' \mid t \in L, t' \in L' \} \).

17 Controllability
4.1 Language-Based Controllability

In this section, we work under the assumption that the plant, requirements and supervisory are modeled as the respective labeled transition systems $P$, $R$, and $S$. We will use the notation $S/P$ to denote the supervised plant, which acts as a model of the effective realization of the plant under supervisory control. We ensure that $S/P$ does not disable uncontrollable events by requesting that $P$ is controllable with respect to $S/P$. This is formalized in the following definition:

**Definition 4.1.** Let $A = U \cup C$ and let $P$ be a model of the plant and $S/P$ be a model of the supervised plant, then $P$ is controllable with respect to $S/P$ if and only if:

$$L(S/P) \cdot U \cap L(P) \subseteq L(S/P)$$

Definition 4.1 shows that any trace in the supervised plant, followed by an uncontrollable action, should be a trace in the supervised plant, if this trace also occurs in the language of the plant itself. In any realistic model of the supervised plant, additional conditions need to be stated to relate $S/P$ to the requirements. A useful and straightforward option is to require that $L(R) = L(S/P)$ but a more complex condition might be appropriate in a specific setting.

Language-based controllability according to Definition 4.1 poses a problem in the non-deterministic setting, as shown in the following example:

**Example 4.2.** Let $L(S/P) = \{a, ac\}$ and $L(P) = \{a, au, ac\}$ for a non-deterministic plant-model as shown in Figure 6. Assume that $a, c \in C$ and $u \in U$. We now have that $au \in L(S/P) \cdot U \cap L(P)$ but $au \not\in L(S/P)$. Therefore, Definition 4.1 is not satisfied in the non-deterministic case.

![Figure 6: Language-based controllability is not suitable for non-determinism; the condition $L(S/P) \cdot U \cap L(P) \subseteq L(S/P)$ is not satisfied.](image)

If a decision about event allowance can be made on a per-state basis, problems regarding non-determinism can usually be avoided [32]. An appropriate construct to capture controllability for the non-deterministic situation is the subject of the remainder of this section.

4.2 State-Based Controllability

Intuitively, state-based control has to be understood in the following way: the plant communicates its current state to the supervisor upon which the latter decides the set of actions that can be taken in this state. To define a state-based notion of controllability, states therefore need to be taken into account, and therefore we choose to work under state-observation on labeled transition systems in the remainder of this section. We
discuss the approach taken in [32] towards an applicable definition of state-based controllability and therefore we need the definition of reachable states \( \text{reach} \) under transitive closure \( \rightarrow^* \) of the transition relation \( \rightarrow \). Let \( X \) be the set of states of a labeled transition graph. For each \( x \in X \), we define \( \text{act}(x) \) as the set of actions originating in state \( x \).

\[
\text{reach}(Y, \epsilon) = Y \text{ with } Y \subseteq X \\
\text{reach}(Y, sa) = \cup\{ q \in X \mid p \xrightarrow{a} q, p \in \text{reach}(Y, s), a \in \text{act}(q) \} \text{ with } s \in A^*, a \in A
\]

We use the definition of \( \text{reach} \) in the following definition of state-controllability.

**Definition 4.3.** Let \( P \) and \( S/P \) be labeled transition graphs of respectively the plant and the supervised plant. We say that \( P \) is state-controllable with respect to \( S/P \) and \( U \) if the following condition is satisfied for the set of starting states \( S/P_0 \) of the supervised plant.

For \( s \in L(S/P), a \in U \) such that \( sa \in L(P) \) it holds that: \( \forall q \in \text{reach}(S/P_0, s) : a \in \text{act}(q) \)

From this definition it is clear that state-controllability implies language controllability [32]. However, there remains an intrinsic problem in this definition of state-controllability, as will be made clear in the following example:

**Example 4.4.** As shown in [10], for this definition of state-controllability, no relation can be defined under bisimulation such that this relation is a preorder. Let \( P = S/P \) be a single labeled transition graph representing both a plant and supervised plant model, as shown in Figure 7. For \( a \in L(S/P) \) and \( b \in U \) it is clear that \( sa \in L(P) \) and there exists a \( q \in \text{reach}(P_0, a) \) such that \( b \notin \text{act}(q) \). This clearly contradicts the definition of state-controllability [32] and shows that it is not reflexive and therefore not a preorder.

**Figure 7:** Counterexample to show that a plant model is not state-controllable with respect to itself. This shows that state-controllability is not a preorder because it is not reflexive, as shown in [10]

### 4.3 Partial Bisimulation as a Means for Controllability

The aforementioned definition of state-controllability does indeed embrace the concept of language-controllability and is suited for non-determinism as well, because event-allowance is determined on a per-state basis. However, as we have shown that state-controllability does not satisfy the condition of preorder, we have to search for a more suitable definition.

We will show that partial bisimulation according to Definition 3.4 does indeed satisfy the required conditions. We will work directly under the assumption that plant, requirements and supervisor are provided as process-theoretic specifications because this avoids confusing two different definitions of synchronization. We rely on two conditions as formulated in the following definition:
Definition 4.5. Let \( p \in \mathcal{T} \) and \( r \in \mathcal{T} \) be process-theoretic specifications of the plant and requirements respectively. If \( \mathcal{A} = \mathcal{U} \cup \mathcal{C} \) then we say that \( s \in \mathcal{T} \) is a supervisor for \( p \) that satisfies \( r \) if:

\[
p \parallel s \preceq u \quad \text{and} \quad p \parallel s \preceq \emptyset \quad r
\]

Where \( p \parallel s \) is a model of the supervised plant, i.e. the model of the plant under supervisory control.

From the first condition in Definition 4.5 it is clear that no uncontrollable actions are disabled in the supervised plant because \( \mathcal{U} \) is included in the bisimulation action set. It is therefore immediately clear that the aforementioned condition of \( L(p \parallel s) \cdot \mathcal{U} \cap L(p) \subseteq L(p \parallel s) \) is satisfied. The second condition in Definition 4.5 states that the supervised plant is an actual realization of the behavior as formulated in the requirements.

We have the following theorem which shows that partial bisimulation is a finer notion compared to state-controllability. It will be shown that the existence of a partial bisimulation implies state-controllability, but as we have shown in Example 4.4, the reverse is not true.

**Theorem 4.6.** If \( p, q \in \mathcal{T} \) such that \( q \preceq u p \) then \( p \) is state-controllable with respect to \( q \).

**Proof.** Let \( p, q \in \mathcal{T} \) such that \( q \preceq u p \) for the partial bisimulation \( R \subseteq \mathcal{T} \times \mathcal{T} \). We have to show that the following condition is satisfied: for \( s \in L(q) \), \( a \in \mathcal{U} \) such that \( sa \in L(p) \) and for all \( x \in \text{reach}(\{q\}, s) \) it holds that \( a \in \text{act}(x) \).

Let \( s \in L(q) \) and \( q' \in \mathcal{T} \) such that \( q \xrightarrow{s} q' \) and let \( a \in \mathcal{U} \) such that \( sa \in L(p) \) with \( p \xrightarrow{s} p' \) for \( p' \in \mathcal{T} \). It is clear that \( q' \in \text{reach}(\{q\}, s) \) and \( (q', p') \in R \) and therefore \( a \in \text{act}(q') \), as follows directly from partial bisimulation according to Definition 3.4.

### 4.4 Controllability in a Process-Theoretic Context

Using the process theory TCP* in conjunction with partial bisimulation, we are able to give precise formulations of the elements and functionality within the control loop. In general, we may say that the plant and requirements can be modeled appropriately using TCP* as there are appropriate constructions such as non-deterministic choice, sequentiality, iteration and communication available. This allows even complicated plants and requirements to be modeled at the right abstraction level. Allowance of uncontrollable behavior and prevention of controllable events is modeled by parameterizing the partial bisimulation preorder with the set of uncontrollable events, as shown in Definition 4.5. Encapsulation in conjunction with communication can be used to enforce occurrence of only complete communication actions. An important remark has to be made with respect to these communication actions as shown in the following example:

**Example 4.7.** Consider a simple plant \( P \) in which two parallel machines \( M_1 \) and \( M_2 \) are signalled by local controller \( C \) that a product is ready for further processing. This process repeats itself indefinitely and is modeled by the following definitions. Let \( P, M_1, M_2, C \in \mathcal{T} \) such that:
We further assume that in this example all actions are controllable. If we have as a requirement that first $M_1$ and then $M_2$ needs to be executed (repeatedly) then we may formulate this as a requirement in the following way:

$$R \overset{\text{def}}{=} (c!\,2\,\text{ready}.\,\text{process}M1.\,\text{process}M2.\,1)^*$$

Assume that we want to create a supervisor $S$ that satisfies the condition of $P \parallel S \preceq R$. This means that the allowed controllable traces from $P \parallel S$ should be simulated by $R$. Because we assume that $U = \emptyset$, it might seem straightforward to choose $S = R$. However, using the operational semantics of the parallel composition operator, we observe that synchronisations are essentially ’multiplied’ in $P \parallel S$. For instance, if $c!\,2\,\text{ready}$ occurs in a trace of $P$, then $c!\,4\,\text{ready}$ occurs in a trace of $P \parallel S$, which is clearly not simulated in $R$.

To solve the issue raised in the previous example, we introduce the renaming operator $\xi : T \rightarrow T$ which traverses the term to which it is applied inductively. This renaming operator has to be partially redefined in each instance it is used to list the exact renamed actions. It is defined according to the following pattern. Let $p, q \in T$ and $\xi : T \rightarrow T$:

- $\xi(0) = 0$
- $\xi(1) = 1$
- $\xi(c!\,m\,?\,n\,d) = \text{model-specific}$
- $\xi(p + q) = \xi(p) + \xi(q)$
- $\xi(p \cdot q) = \xi(p) \cdot \xi(q)$
- $\xi(p^\ast) = \xi(p)^\ast$
- $\xi(p \parallel q) = \xi(p) \parallel \xi(q)$
- $\xi(c!\,m\,?\,n\,d,p) = \xi(c!\,m\,?\,n\,d)\,\xi(p)$ for $c \in \mathcal{H}, d \in \mathcal{D}, m, n \in \mathbb{N}$

This renaming function is applied in the next example in which we use TCP* to model a case of automated guided vehicles.

## 5 An Event-Based Supervision Example

In this section, we illustrate our approach to supervisory control and the model of the control loop. We discuss a simple example concerning coordination of an automated guided vehicle (AGV) in an automated production line, depicted in figure 5. The AGV is responsible for transferring the preproduct made by Workstation M to Workstation N and transferring the finished product from Workstation N to the Delivery station. These are two phases that need to be executed in a sequential fashion.

The workstations and the AGV are coordinated by a supervisor, which sends the corresponding control signals. We will show how to model the automated production system
We use the process terms $M, N, A$ and $S$ to model Workstation M, Workstation N, the AGV and the supervisor. Note that in this model, we abstract from the delivery station (depicted by a single event $deliver$, as it does not contribute to any interesting behavior. We use the same communication channel names as used in figure 5. The data elements are $D = \{make, move2N, preproduct, product\}$. As uncontrollable channels we have $U = \{m, n, produce, process, move, deliver\}$ and as controllable channel we have $C = \{s\}$.

**Example 5.1.** We use the following definitions for this example:

\[
\begin{align*}
M &\seteq (s?make.produce(preproduct).m!preproduct.1)^* \\
N &\seteq (n?preproduct.process(preproduct).n!product.1)^* \\
A &\seteq (m?preproduct.s!move2N.move(preproduct).n!preproduct.1+n?product.deliver(product).1)^* \\
S &\seteq (s!make.s!move2N.1)^* \\
\end{align*}
\]

Workstation $M$ repeatedly waits for a command from the supervisor to make a preproduct, which is offered to the AGV once it is made. Workstation $N$ waits for a preproduct from the AGV, which is thereafter processed and offered back to the AGV. The AGV can either pick up a preproduct at workstation $M$, after which it asks for permission to move the preproduct to Workstation $N$, or pick up a finished product at Workstation $N$, and deliver it. Now, the unsupervised plant is given by the process:

\[
U \seteq \partial_F(M \parallel N \parallel A), \text{ where } F = \{m?, m!, n?, n!\}
\]

At this point, we note that we enforce meaningful communication of uncontrollable channels within the plant by encapsulation and this does not restrict the behavior of the unsupervised plant, but only ensures its meaningful behavior. Following the framework outlined above, it can be readily observed that the plant $U \in \mathcal{P}$ follows the outlined syntax.

In this first modeling instance, we assume that the AGV is responsible for delivering the final product and we propose a supervisor as given by the process $S$. Note that the supervisor $S \in S$ follows the outlined syntax and it does not make use of any observed control signals. Figure 8: AGV case

---

**Figure 8: AGV case**

from figure 5 using TCP*. We use the following definitions for this example:

\[
\begin{align*}
M &\seteq (s?make.produce(preproduct).m!preproduct.1)^* \\
N &\seteq (n?preproduct.process(preproduct).n!product.1)^* \\
A &\seteq (m!preproduct.s!move2N.move(preproduct).n!preproduct.1+n?product.deliver(product).1)^* \\
S &\seteq (s!make.s!move2N.1)^* \\
\end{align*}
\]
information. Supervisor $S$ repeatedly gives orders to Workstation $M$ for new products to be made, followed by orders to the AGV to transfer the preproduct to Workstation $N$. Thus, the automated production system is modeled as:

$$U/S \overset{\text{def}}{=} \partial_E(S \parallel U), \text{ where } E = \{s?, s!\}$$

which enforces communication of control signals and transfer of the (pre)products. One can directly check that $S$ is a valid supervisor by establishing that the supervised plant is partially bisimulated by the original plant with respect to the uncontrollable events. To this end, we must employ renaming of events, as the original plant has open communication actions that wait for synchronization with the supervisor. This renaming function $\xi$ traverses the process terms and renames all open communication actions to succeeded communication actions. We note that we overload the name of the renaming function of the process terms and apply it to the communication action names as well. Also, we only specify the communication actions that are actually renamed.

Now, in order to verify that the supervisor does not disable uncontrollable events, it is sufficient to verify that it holds that

$$U/S \leq_{\mathcal{A}_U} \xi(U), \text{ where } \xi : s?d \mapsto s!d \text{ for } d \in D$$

which can be directly checked. We note that there was no restriction imposed on the control requirements, which in case coincide with the plant and are, therefore, trivially satisfied.

Unfortunately, our automated production has a deadlock. The main reason for the deadlock is that a second preproduct can come too early, before the first product is completely finished and delivered, which is set off by sending a $s!$ make command too early, that is, before the processed product has left Workstation $N$. Then, the AGV picks up the preproduct from Workstation $M$, but it cannot deliver it to Workstation $N$, as the latter also waits for a finished product to be picked. A trace that leads to deadlock is the following one:

Such form of blocking behavior appears often, so in many cases the supervisor is additionally required to prevent situations in which blocking behavior such as deadlock and livelock occurs. To this end, special marked states are introduced to automata in supervisory control. We note that these states roughly correspond to successful termination in our setting. The correspondence is not strict, mainly due to the absence of sequential composition and the Kleene star operator in the supervisory control literature and the role of the successful termination in these contexts. Note that the marked states do not contribute to the formation of the recognized language of an automaton, which is different from its marked language.

So, besides the control requirements, we impose an additional deadlock freedom requirement on the supervisor, stated formally as: there exists no trace $t \in \mathcal{A}^*$ such that $U/S \overset{\text{t}}{\Rightarrow}^* 0$. To ensure this additional nonblocking requirement, we have to modify the supervisor to accept requests for making a new preproduct only after the finished product has been loaded on the AGV, to be transferred to the delivery station.

To this end, the supervisor should allows for a new product to be made only after the finished product has been loaded to the AGV at Workstation $N$, which can be achieved by
observing this additional information on channel $n$. To this end, we modify the supervisor to $S'$ as follows:

$$S' \overset{\text{def}}{=} (s! \cdot s! \cdot \text{move2N} \cdot n? \cdot \text{product} \cdot 1)^*$$

At this point, we note that communication on the channel $n$ now must occur between three parties. This situation occurs when Workstation $N$ that sends information and the AGV and the supervisor that receive it. In order to enforce this communication, we employ the generic communication actions. We encapsulate all (incomplete) communication actions $N$, except for $n!_1?_2 \cdot \text{product}$. The definition of the deadlock-free supervised plant now becomes:

$$U/S' \overset{\text{def}}{=} \partial_{E'}(S' \parallel U), \text{ where } E' = \{s?, s!, n?, n!\}$$

Again, one directly verifies that the supervisor is valid by establishing partial bisimilarity between the supervised and the original plant following an appropriate renaming of the incomplete communication actions, given by $\xi : s?d \mapsto s!_d, n?_d, n!\mapsto n!_1?_2d$ for $d \in D$.

6 The process theory TCP$^*_\perp$

We propose TCP$^*_\perp$, an extension of TCP$^*$, with propositional signals and guarded commands in order to support the modeling of a control loop with state-based observations. The end purpose of this setup is to model assymmetric supervised plants, tailored towards the specific needs for plants as well as supervisors. The asymmetric nature of this construction becomes clear from the following intuitive explanation. The plant communicates its current state to the supervisor, upon which a list of enabled signal is sent back to the plant by the supervisor. Therefore, besides the standard process terms as discussed before, the plant only has to be able to perform outward communication of its state. On the other hand, the supervisor has to be able to receive this information and to either allow or disallow it. Thereby obviously taking into account the fact that uncontrollable transition should never be disallowed. In this section, we define the concurrency theory TCP$^*_\perp$ to achieve this. We employ a different grammar for terms that are designated to model plant components compared to supervisor terms. However, they both use the same operational semantics.

$$P \Rightarrow 0 \mid 1 \mid c?_dP \mid n!_1?_kd.P \mid P + P \mid P \cdot P \mid P^* \mid P \parallel P \parallel P \parallel \partial_{E'}(P) \mid \phi \rightarrow P \mid \phi \cdot P$$

$$S \Rightarrow 1 \mid c!d.S \mid S + S \mid \phi \rightarrow S \mid S^*$$

for $c \in C$, $u \in U$, $l, k \in \{0, 1\} \subset N$, $d \in D$, $\phi \in B$, and $E \subseteq \{f!_m?_n | f \in H, m, n \in N\}$.

On of the other foundations of TCP$^*_\perp$ is the Boolean algebra:

$$\mathbb{B} = (\mathcal{P}, \text{false}, \text{true}, \neg, \wedge, \lor, \subset)$$

where $\mathcal{P} = \{P_1, \ldots, P_n\}$ is the set of propositional symbols. Other members of this algebra should be self-explanatory. This includes the constants true and false, negation, conjunction, disjunction and implication. Note that we use the symbol $\subset$ instead of $\rightarrow$ or $\Rightarrow$ for implication to avoid confusion because of too many arrows. We use $\mathcal{B}$ to denote that standard Boolean formulas of $\mathbb{B}$, which are evaluated with respect to a valuation $v : \mathcal{B} \rightarrow \{\text{true}, \text{false}\}$. The set of valuations is denoted by $\mathcal{V}$. 24
We enrich the syntax of TCP* and the set of process terms $\mathcal{T}$ with the inaccessible process, guarded commands and signal emission [1]. The inaccessible process, notation $\bot$, specifies the process in which there are inconsistencies between the valuation of the propositional variables and the emitted propositional signals. A guarded command, notation $\phi \rightarrow p$, specifies a formula $\phi \in \mathcal{B}$ that functions as a delimiter of a process $p \in \mathcal{T}$. If the guard $\phi$ evaluates to $true$, then the process $p$ is allowed to continue as usual. If $\phi$ evaluates to $false$ then the guarded process $p$ deadlocks. The root signal emission operator $\phi \cdot ! p$ emits the propositional signal $\phi \in \mathcal{B}$ until the process $p \in \mathcal{T}$ takes an outgoing transition. A prerequisite is that the propositional signal is consistent with the valuation. To be able to evaluate the Boolean formulas, we couple the process terms with valuations, notation $(p, v) \in \mathcal{T} \times \mathcal{V}$. The dynamics of the valuations, with respect to outgoing labeled transitions, is captured by the predefined valuation effect function. This function has the signature: $effect : \mathcal{A} \times \mathcal{V} \rightarrow 2^\mathcal{V}$. With respect to the valuation we have to extend the successful termination predicate to $\triangleright \in \mathcal{T} \times \mathcal{V}$ and the labeled transition relation to $\rightarrow \in \mathcal{T} \times \mathcal{V} \times \mathcal{A} \times \mathcal{T} \times \mathcal{V}$. We introduce an additional consistency predicate $\triangleright \in \mathcal{T} \times \mathcal{V}$ that checks whether the state is consistent. The operational rules in table 6.1 give the semantics of the new predicate and the transition relation with respect to the new operators.

6.1 Operational Semantics

We briefly comment on the rules in Table 6.1. Rules 1 and 2 state that any valuation is consistent with respect to the constant terms 0 and 1. Rule 3 is equivalent to the one in TCP* where the constant 1 always terminates. Rule 4 shows that the action prefix does not influence consistency. The next rule 5 details how the action prefix is allowed to take a step based on the result of the effect function. Rules 6 and 7 show how the alternative composition behaves as in TCP*, provided that consistency is given. The rules 8 and 9 show how consistency is required for termination of alternative composition as well. Rule 10 states that consistency of the operands is transferred to consistency of a sum. In rules 11 and 12 it is detailed how termination of sequential composition and right-operand transition behaves as in TCP*. In the case of a left-operand transition, consistency is required for the resulting term $p' \cdot q$ as is shown in rule 13. In rules 14 and 15 it is detailed how termination in the left operand transfers consistency for sequential composition. Consistency for the left side $(p, v)$ only transfers to the product $(p \cdot q, v)$ on the condition that the right side $(q, v)$ does not terminate, as shown in 15. Rules 19 and 20 state that termination and consistency for parallel composition depend on both operands. In rules 21 and 22 it is detailed how unilateral steps in the parallel operator $|\|$ takes place. Note that this depends on remaining consistency of the term that remains constant. The rule 23 shows how a bilateral parallel step can be enabled by the effect function, and is further similar to that of TCP*. The rules 24, 25, and 26 show the behavior of the encapsulation operator in this new setting. The rules 27 and 28 show how a true valuation is required for termination as well as consistency of a guarded term. The somewhat remarkable rule 29 show that a guarded term is consistent, even if its corresponding valuation evaluates to $false$. Rule 30 states the effective behavior of a guarded process, enabling the underlying term only if the valuation $\phi$ is true. The last three rules consider the signal emission operator. Rules 31 and 32 show how a true valuation is required for the signal emitting term to terminate and to be consistent. The last rule 33 details how a true valuation is required for a transition step as well.

We require an additional property of the effect function in order for it to be well-defined [1]. Let $c \in \mathcal{H}$, $d \in \mathcal{D}$, and $l, k, m, n \in \mathcal{B}$ with $l + k > 0$ and $m + n > 0$:

$$effect(c_{l+m}^{l+m+k}\cdot l+k+n d) \subseteq effect(c_{m+n}^{l+m+k}, effect(c_{l+n}^{l+n}d, v)) \cap effect(c_{l+n}^{l+n}d, effect(c_{l+n}^{l+n}d, v))$$
6.2 Partial Bisimulation

We have to adopt Definition 3.4 in such a way that it correctly handles valuations. We base this approach on the work in [1] and [3] where this extension is investigated for (strict) bisimulation.

**Definition 6.1.** We define a relation \( R \subseteq T \times T \) to be a partial bisimulation with respect to the bisimulation action set \( B \subseteq A \) if for all \( (p, q) \in R \) it holds that:

1. If \( \langle p, v \rangle \Downarrow \) for some \( v \in V \) then \( \langle q, v \rangle \Downarrow \).
2. If \( \langle q, v \rangle \Downarrow \) for some \( v \in V \) then \( \langle p, v \rangle \Downarrow \).
3. If \( \langle p, v \rangle \xrightarrow{a} \langle p', v' \rangle \) for some \( v, v' \in V \) and \( a \in A \) then there exists a \( q' \in T \) such that \( \langle q, v \rangle \xrightarrow{c} \langle q', v' \rangle \) and \( (p', q') \in R \).
4. If \( \langle q, v \rangle \xrightarrow{b} \langle q', v' \rangle \) for some \( v, v' \in V \) and \( b \in B \) then there exists a \( p' \in T \) such that \( \langle p, v \rangle \xrightarrow{a} \langle p', v' \rangle \) and \( (p', q') \in R \).

Just as in Definition 3.4, we say that \( p \) is partially bisimilar to \( q \) with respect to \( B \), and we use the notation \( p \preccurlyeq_B q \). If \( q \preccurlyeq_B p \) holds as well, we write \( p \equiv_B q \).

6.3 Axiomatization

We extend the axiomatization for the process theory TCP\(^*\) as given in Table 1 to a finite axiomatization for TCP\(^*_\perp\). The axioms in Table 6.3 are given with respect to (strict) bisimulation, which allows us to obtain a corresponding axiom for partial bisimulation with respect to an arbitrary bisimulation action set \( B \subseteq A \), as shown in Lemma 3.5. In Table 6.3 we only state the additional axioms that are required for TCP\(^*_\perp\) and this table should therefore be considered as an extension of Table 1. These axioms are considered in detail in [1], where the same abbreviations are used. We prove soundness of the axiom (PBS), because it is specific for partial bisimulation. In [1] it is shown how proofs of soundness of the axioms in Table 1 can be converted to the state-based situation and how the axioms in Table 6.3 can be shown to be sound.

**Lemma 6.2.** For \( p, q \in T \) we have that \( p \preccurlyeq_B a.q + p \) implies that \( p \preccurlyeq_B a.q + p \) provided that \( a \not\in B \).

**Proof.** Choose \( R = \{(p, a.q + p)\} \cup \{(r, r) \mid r \in T\} \). Assume that \( \langle p, v \rangle \Downarrow \) for some \( v \in V \), then we have \( (a.q + p) \Downarrow \) according to rule 9, and vice versa. If \( \langle p, v \rangle \xrightarrow{c} \langle p', v' \rangle \) for some \( v, v' \in V \) and \( c \not\in B \), then according to rule 7 we have that \( \langle a.q + p, v \rangle \xrightarrow{c} \langle q', v' \rangle \) and \( (p', q') \in R \). If \( \langle a.q + p, v \rangle \xrightarrow{b} \langle q', v' \rangle \) for some \( b \in B \) then this step has to come from \( p \) because \( a \not\in B \). Again using rule 7, we have that \( \langle p, v \rangle \xrightarrow{b} \langle q', v' \rangle \) and \( (q', q') \in R \).

7 Oce Printer Case: An Industrial Example

We employ the process theory TCP\(^*_\perp\) to model the coordination of maintenance procedures of a printing process of a high-end Océ printer [22]. The printing process consists of
Let $x, y \in T$ and $\phi, \psi \in B$:

$p \leq (\phi \rightarrow a.q) + p$ if $\phi \neq$ false \hspace{1cm} PBS

| \text{true} : \rightarrow x = x | IR1  \\| \text{false} : \rightarrow x = 0 | IR2  \\| \phi : \rightarrow 0 = 0 | IR3  \\| \phi : \rightarrow (x + y) = (\phi : \rightarrow x) + (\phi : \rightarrow y) | IR4  \\| (\phi \lor \psi) : \rightarrow x = (\phi : \rightarrow x) + (\psi : \rightarrow y) | IR5  \\| \phi : \rightarrow (\psi : \rightarrow x) = (\phi \land \psi) : \rightarrow x | IR6  \\| \phi : \rightarrow (x \cdot y) = (\phi : \rightarrow x) \cdot y | IR7  \\| \phi : \rightarrow (x \mid y) = (\phi : \rightarrow x) \mid y | IR8  \\| (\phi : \rightarrow x) \mid y = rs(y) + \phi : \rightarrow (x \mid y) | IR9  \\| \phi : \rightarrow \partial_E(x) = \partial_E(\phi : \rightarrow x) | IR10 |

| true.$^\ast x$ = x | RSE1  \\| $\phi.$$^\ast (\psi.$$^\ast x$) = $\phi.$$^\ast (\phi \land \psi).$$^\ast x$ | RSE4  \\| $\phi.$$^\ast (\psi.$$^\ast x$) = $\phi.$$^\ast (\phi \lor \psi) \land (\phi : \rightarrow x)$ | RSE5  \\| $\phi.$$^\ast (\phi : \rightarrow x) = \phi.$$^\ast x$ | RSE6  \\| $\phi.$$^\ast (x \cdot y) = (\phi.$$^\ast x$) \cdot y | RSE7  \\| $\phi.$$^\ast (x \mid y) = (\phi.$$^\ast x$) \mid y | RSE8  \\| $\phi.$$^\ast (\psi.$$^\ast x$) \mid y = $\phi.$$^\ast (x \mid y)$ | RSE9  \\| $\phi.$$^\ast (\psi.$$^\ast x$) \mid y = $\phi.$$^\ast x \mid y | RSE10  \\| $\partial_E(\phi.$$^\ast x$) = $\phi.$$^\ast \partial_E(x)$ | RSE11 |

rs(1) = 0 \hspace{1cm} RS2  \\rs(0) = 0 \hspace{1cm} RS3  \\rs(a.x) = 0 \hspace{1cm} RS4  \\rs(x + y) = rs(x) + rs(y) \hspace{1cm} RS5  \\rs(\phi : \rightarrow x) = \phi : \rightarrow rs(x) \hspace{1cm} RS6  \\rs(\phi.$$^\ast x$) = $\phi.$$^\ast rs(x)$ \hspace{1cm} RS7

**Table 3:** Axioms for TCP$^*_\perp$ as an extension of the axioms for TCP$^*$ in Table 1.
several distributed independent components as depicted in Fig. 9a). The process applies the toner image onto the toner transfuse belt and fuses it onto the paper sheet. To maintain high printing quality, several maintenance operations have to be carried out, some of which are: (1) toner transfuse belt jittering, which displaces the transfuse belt to prolong its lifespan due to wearing by paper edges; (2) black image operation, which removes paper dust by occasionally printing completely black pages; (3) coarse toner particles removal operation. Most maintenance operations are scheduled after a given number of prints, but must be carried out after a given strict threshold.

For example, after 500 prints there is need to perform a black image operation, unless there is an active print job, but after 700 prints it must be executed to maintain the printing quality, even if an ongoing print job needs to be interrupted. To perform a maintenance operation, the printing process has to change its power mode, from Run mode, used for printing, to Standby mode, required for maintenance. However, this change can actually trigger pending maintenance operations, which may unnecessary prolong the user waiting time.

As an illustration, in Fig. 9b) we depict the situation, where due to inevitable execution of maintenance operation A, the ongoing print job is suspended and the power mode of the printer is changed to Standby. However, an unwanted situation occurs, i.e., the power mode change triggers a longer, yet postponable maintenance operation B as depicted in Fig. 9c). For instance, a black image operation (A) must be performed, which takes the time needed to print one page and is activated often, but the switching of the power mode triggers the much longer toner transfuse belt jittering (B), thus making the user wait unnecessarily.

The goal of the research performed for this use case was to eliminate undesired emergent behavior due to interactions of otherwise correctly-functioning distributed components, with primary focus at coordinating maintenance operations. Our approach was to synthesize a supervisory coordinator for the maintenance procedures [22], which here we model in the proposed process theory.

An abstract view of the control architecture of a high-end printer is depicted in Fig. 10. Print jobs are sent to the printer by means of the user interface. The printer controller communicates with the user and assigns print jobs to the embedded software, which actuates the hardware to realize print jobs. The embedded software is organized in a distributed way, per functional aspect, such as, paper path, printing process, etc. Several managers communicate with the printer controller and each other to assign tasks to functions, which take care of the functional aspects.

We depict a printing process function comprising one maintenance operation in Fig. 10. We abstract from all timing behavior, which can be present in some control signals, e.g., execute a maintenance procedure after a given delay. Each function is hierarchically organized as follows: (1) controllers: Target Power Mode and Maintenance Scheduling, which
receive control and scheduling tasks from the managers; (2) procedures: Status Procedure, Current Power Mode, Maintenance Operation, and Page Counter, which handle specific tasks and actuate devices, and (3) devices as hardware interface.

We describe the procedures. The Status Procedure is responsible for coordinating the other procedures given the input from the controllers. It will be implemented as a supervisory coordinator. We give the coordination rules below. The Current Power Mode procedure sets the power mode to Run or Standby depending on the enabling signals from the Status Procedure \( Stb2Run \) and \( Run2Stb \), respectively. The confirmation is sent back via the signals \( InRun \) and \( InStb \), respectively. Maintenance Operation either carries out maintenance operation or it is idle. The triggering signal is \( OperStart \) and the confirmation is sent back by \( OperFinished \). The Page Counter procedure counts the printed pages since the last maintenance and sends signals when soft and hard deadlines are reached using \( ToSoftDln \) and \( ToHardDln \), respectively. The counter is reset each time the maintenance is finished, by receiving the confirmation signal \( OperFinished \) from Maintenance Operation. The controller Target Power Mode defines which mode is requested by the manager by sending the control signals \( TargetStb \) and \( TargetRun \) to the Status Procedure. Maintenance Scheduling receives a request for maintenance from Status Procedure via the signal \( SchedOper \), which it forwards to a manager. The manager confirms the scheduling with the other functions and sends a response back to the Status Procedure via the control signal \( ExecOperNow \). It also receives feedback from Maintenance Operation that the maintenance is finished in order to reset the scheduling.

We model the procedures by means of processes. We will retain the names of the control signals, turning them into communication actions where appropriate. The controllable communicating channels are the given by \( C = \{ Run2Stb, Stb2Run, SchedOper, OperStart \} \), modeled as receive communication actions in the plant. We note that we abstract from data elements as communication should only enforce ordering of events. The other actions are uncontrollable, also prefixed by \( _{} \), where only \( OperFinished \) is modeled as a communication action, as the procedure Maintenance operation must send signals and reset Page Counter and Maintenance Scheduling. The signals emitted from the plant uniquely identify the state of the plant. Page Counter is modeled by the process \( C \), where \( OperFinished \) is modeled as a receive action, to be synchronized with Maintenance Operation:
Maintenance Operation is specified by the process $O$, where \(_{\text{OperFinished}}\) broadcasts that the maintenance operation has finished:

$$O \triangleq (\text{in(OperIdle)} \cdot_{\text{OperStart}} \cdot_{\text{in(OperInProg)}} \cdot_{\text{OperFinished}.1})^\ast.$$  

Target Power Mode is modeled by $T$:

$$T \triangleq (\text{in(TargetStandby)} \cdot_{\text{TargetRun}} \cdot_{\text{in(TargetRun)}} \cdot_{\text{TargetStandby}.1})^\ast,$$
where as Current Power Mode is given by $P$:

$$P \triangleq (\text{in(Standby)} \cdot_{\text{Stb2Run.in(Starting)}} \cdot_{\text{InRun}} \cdot_{\text{Run2Stb.in(Stopping)}} \cdot_{\text{InStb}})^\ast.$$  

Finally, Maintenance Scheduling is specified as $M$:

$$M \triangleq (\text{in(NotScheduled)} \cdot_{\text{SchedOp.in(Scheduled)}} \cdot_{\text{ExecOpNow}} \cdot_{\text{in(ExecuteNow)}} \cdot_{\text{OperFinished}.1})^\ast.$$  

Now, the unsupervised plant can be specified as $U \in \mathcal{P}$ given by:

$$U \triangleq \partial_F(C \parallel O \parallel T \parallel P \parallel M),$$

For clarity, we depict the process in Fig. 11, where the signal names are given next to the states that emit them. We synthesized a coordinator that implements Status Procedure, which coordinates the maintenance procedures with the rest of the printing process. The following coordination requests describe the behavior of the Status Procedure:

1. Maintenance operations can be performed only when the printing process is in standby;
2. Maintenance operations can be scheduled only if soft deadline has been reached and there are no print jobs in progress or a hard deadline is passed;
3. Maintenance operations can be started only after being scheduled;
4. The power mode of the printing process must follow the power mode dictated by the managers, unless overridden by a pending maintenance operation.

We formalize the control requirements as follows:

1. The maintenance procedure is performed if the process emits the signal \(_{\text{OperInProg}}\), while emitting the signal \(_{\text{Standby}}\) as well:
   $$R_1 = \text{OperInProg} \supset \text{Standby}$$
2. For the control signal $SchedOper!$ to be sent to Maintenance Scheduling, either one of the following must hold: (1) A soft deadline has been passed, identified by emission of the signal $SoftDeadline$, and there are no print jobs waiting, meaning that the target power mode is not in run, identified by the signal $TargetRun$; or (2) A hard deadline has been passed, indicated by the signal $HardDeadline$. This is captured by the following control requirement:

$$R_2 = SchedOper! \supset (SoftDeadline \land \neg TargetRun) \lor HardDeadline.$$ 

3. The maintenance operation can be started by sending the control signal $OperStart!$ only if it has been scheduled, prompted by the emission of the signal $ExecOperNow$:

$$R_3 = OperStart! \supset ExecOperNow.$$

4. If we want to switch from standby to run power mode, indicated by sending the control signal $Stb2Run!$, then this has been requested by the target power mode manager by emitting the signal $TargetRun$, provided that there are no maintenance operations scheduled, for which the signal $ExecOperNow$ should be checked:

$$R_{41} = Stb2Run \supset TargetRun \land \neg ExecOperNow.$$ 

When switching from run to standby power mode, indicated by sending the control signal $Run2Stb!$, the target power mode should be in the standby, given by emission of the signal $TargetStandby$. An exception is made when a maintenance operation is scheduled to be executed, given by emission of the signal $ExecOperNow$:

$$R_{42} = Run2Stb \supset TargetStandby \lor ExecOperNow.$$

With respect to the control requirements we synthesized a deadlock- free supervisor \[22\]. The supervisor sends the control signals upon observation of certain signal combinations,
which are given in the form of guards. The indices of the guards correspond to the indices of the control requirements that concern the control signal:

\[ g_2 = (\text{in(SoftDeadline)} \land \text{in(TargetStandby)}) \lor \text{in(HardDeadline)} \]

\[ g_3 = \text{in(Standby)} \land \text{in(ExecuteNow)} \]

\[ g_{41} = \neg \text{in(ExecuteNow)} \land \text{in(TargetRun)} \land \neg \text{in(OperInProg)} \]

\[ g_{42} = (\neg \text{in(ExecuteNow)} \land \text{in(TargetStandby)}) \lor \text{in(ExecuteNow)}. \]

The supervisor is given by

\[ S = (g_2 : \rightarrow \text{SchedOper!}.1 + g_3 : \rightarrow \text{OperStart!}.1 + \\
    g_{41} : \rightarrow \text{Run2Stb!}.1 + g_{42} : \rightarrow \text{Stb2Run!}.1)^*. \]

Now, the supervised plant \( U/S \) is given by:

\[ U/S \equiv \partial_E(S \parallel U), \text{ where } E = \{ c!, c? \mid c \in \text{CCH} \}. \]

Again, we can show that the supervised plant is partially bisimilar to the original plant with respect to the uncontrollable events by showing that

\[ U/S \preceq \xi(U), \text{ where } \xi : c? \mapsto c!! \text{ for } c \in \text{CCH}. \]

The above form of the supervisor does not provide much information regarding the choices made. For example, it is not difficult to deduce that since the initial signal is \text{Standby}, the event \text{Run2Stb} is not possible. Also, \text{StartOper} is unavailable as the signal \text{ExecuteNow} is not emitted. In order to better understand the control choices made by the supervisor, we depict an alternative supervisor in Fig. 12. We note that both variants produce equivalent supervised behavior (note that the guards remain the same), the difference being that the supervisor depicted in Fig. 12 reveals the consequences of choosing a particular controllable action. We can now observe, that if the operation is scheduled while the printing process is in standby power mode, then it can be directly executed, returning the supervisor to the initial state. If the power mode is changed to run, then the operation can still be scheduled, but the system has to switch to standby power mode for it to be executed.

8 Conclusion

In the introduction we mentioned an important problem that persists in the development of control software nowadays: the gap between informal specification and concrete realization of control software. However, we do see the theoretical foundations we laid out in this article as more than just a modelling aid that acts as a stepping stone between...

33 Conclusion
informal requirements and the creation of an actual controller. Instead, we argue that
a process-theoretic approach to supervisory control theory effectively bridges this gap.
The feedback loop for supervisory control is advantageous in the sense that it abstractly
models the realistic distinction between applicable control and uncontrollable behavior.
This abstract model is given a concrete refinement by providing a convenient formalism to
model the plant and supervisor using the process theory TCP*, which has been described
extensively in this article. Our framework is further streamlined by allowing requirements
to be stated using the same formalism.

The distinct parts of the process theory TCP* correspond to elementary modeling needs in
concrete situations. Parallellism allows the specification of multiple adjoined components
in the plant that function interactively. Where interactions are modeled using parallel
communication functionality as clearly present in the theory TCP*. Encapsulation is used
to model effective restrictions on communicating processes. Thereby providing flexibility
in the specification of individual components while retaining the ability to restrict com-
munications if processes are combined. Iteration allows the specification of continuous
plant-components that have the option to terminate while allowance of non-determinism
provides the ability to integrate all of the aforementioned features in a model at the de-
sired level of abstraction. By means of an operational semantics, we connect elements of
the theory TCP* to labeled transition system realizations.

We defined partial bisimulation as a means to consider process theoretic terms under
a preorder modulo structural behavior. This connects to the concept of bisimulation
of uncontrollable events, while enabling restrictions on control by means of unilateral
simulation. We related the precongruence property of partial bisimulation to the format
of the operational rules, thereby allowing an easy generalization towards extensions of
the theory. The theory TCP* is further studied under finite axiomatization to provide
insight into its validity, which is then extended in the provided soundness and ground-
completeness proofs.

We specified controllability as avoiding disallowance of uncontrollable behavior. For trace-
based control this is realized in the definition of event-based controllability, which is
unsuitable for non-deterministic systems. State-based control can be an effective means
to capture non-determinism and we discuss this in the light of an earlier approach that
does not satisfy reflexivity in its behavioral preorder between plant and supervisor. As
we require the plant to be state-controllable to itself, we adapted the definition of partial
bisimulation for state-based control and showed that it satisfies the required properties.
We introduced a renaming operator to enable a parallel and communicating construction
of plant and supervisor in the control loop. This feature is used in an example case
of automated guided vehicles where we use TCP* to model parallel components under
event-based supervisory control.

We extend the process theory TCP* to include state-based Boolean valuations, guarded
commands and signal emission. This enables the specification of outward communicating
plant models that adhere to state-based control signals. We showed how this setup retains
the required controllability property by redefining a suitable partial bisimulation preorder
in terms of state-based valuations. The extended theory of TCP*⊥ is again considered
under finite axiomatization. This specification of TCP*⊥ is followed by an extensive
case study into the supervisory control of an industrial printer where five parallel plant
components are modelled as communicating processes that operate under state-based
control. This final case concludes the definition, investigation and application of a formal
specification theory for supervisory control.
8.1 Future work

Our future research activities will focus on a new formal language for control modeling called SEAL. In this specification language we intend to combine a convenient set of constructs to model discrete concurrent event systems and logical expressions which can be used in a descriptive, generative way as well as for verification purposes. The foundation for SEAL consists of labeled transition system formalism that has the ability to express parallelism, communication, iteration, and various other features to model discrete event systems at a desired level of abstraction. Transition systems described in SEAL are then refined using logical expression that form an intrinsic part of the SEAL formalism. This two-level approach allows inclusion or exclusion of states and transitions under conditions that are more conveniently expressed in a logical language, compared to the coarser automata formalism. Our intention is to provide a formal semantics in which the generative effect of logical rules is expressed in terms of alterations to the labeled transition systems described in SEAL. A unified logical language will be play a descriptive role in SEAL models as well as for the verification of properties on these models. This streamlined approach allows to establish an optimized connection to existing tools for simulation and verification of discrete event systems. In conclusion, we intend to accommodate the two-tier SEAL formalism in such a way that it encompasses the following four areas: 1) Convenient modeling of discrete event systems, 2) Optimized simulation of these systems, 3) Synthesis of systems, expressed as transformations in terms of logical rules, for instance supervisory controllers, and 4) Verification of dependent properties on discrete event systems.


