Protocol based workflow management system

van 't Klooster, G.T.

Award date:
2014

Master Thesis Embedded Systems

Title:	Protocol Based Workflow Management System
Student:	G.T. van 't Klooster
Department:	Mathematics and Computer Science
E-mail:	g.t.v.t.klooster@student.tue.nl
Student#:	0767701
Date:	18 August 2014

Graduation supervisor:
prof.dr.ir. J.F. Groote,
University of Technology, Eindhoven.

Graduation tutor:
dr. H.A.F. Gratama van Andel,
Elekta, Veenendaal.

Contents

1 Introduction

The aim of this report is to explain and to provide the formal semantics for a workflow management system, which will be used by the adaptive brachytherapy platform. This workflow management system is developed to control, schedule and monitor the workflow of a medical treatment in the field of brachytherapy in order to increase the quality of medical intervention.

The adaptive brachytherapy platform is designed to support user-centric workflows. Hence, the platform supports hospitals of scalable sizes. Moreover, users who have skills in the range from basic to advanced, should be able to manage and control the brachytherapy treatment of a patient. The user-interface of the platform is adjustable with use of the hospital clinical protocols and preferences.

A clinical workflow for brachytherapy can be separated into small demarcated steps, *i.e., workflow steps*. The workflow management system will schedule the workflow steps.

1.1 Project description

The aim of this project is to develop a protocol based workflow management system. This workflow management system will interpret the protocol in order to schedule the workflow steps of a clinical workflow. This scheduling approach is called: Protocol based workflow scheduling.

The main advantage of a protocol based workflow is the reproducibility of a workflow. Besides that, the user can only choose workflow steps that are allowed to be executed according to the protocol. Hence, the chance of mistakes is reduced. The user will not be hampered by starting feasible workflow steps, because the system disallows only the workflow steps that are not allowed to execute according to the protocol.

Moreover, the user has the ability to achieve a certain quality of a medical intervention, because he is able to get insight in the execution and risks of the performed workflows. With use of this insight the quality of the treatment can be optimized. This is a reason why the medical field can benefit from this methodology.

A protocol based workflow management system is based on the semantics that all workflow steps that do not violate the constraints of the protocol are allowed to be executed. This semantics contributes to the ability to create the highest flexible workflow that is allowed by the protocol. A protocol based workflow management system provides more flexibility than a traditional workflow management systems [13].

A medical protocol describes: the equipment that is allowed to be used; the execution order of workflow steps; constraints on for example the height of a dose rate; the necessary authorization of the medical staff; and conditions where the patient must reside in. The protocol is used to determine which workflow steps are allowed to be executed each moment in time. Therefore, the protocol describes the dependencies between the workflow steps.

The constraint based workflow model depends on the inter-task dependency. This means that the restrictions are based on the dependencies between the workflow steps. The inter-task dependency can be specified into value dependency and external dependency.

Khemuka [11] distinguished these dependencies by:

- Value dependency: A value dependency specifies task dependencies based on the output value generated by certain tasks.
- External dependency: These dependencies are due to some external factors.

The external dependencies are described in the protocol rules of the treatment. The protocol rules describe the behavior in a declarative style [13], e.g., *"eventually workflow step A is followed by workflow step B"*.

1.2 Architecture description

The adaptive brachytherapy platform aims to integrate multiple devices and algorithms into a single platform. Therefore, a five layer reference model architecture is developed at Elekta. The five layers are based on the separation of concerns of the platform. Figure 1.2.1 depicts the layering architecture.

- The bottom layer realizes the interfaces between the platform and the medical devices, which can be in contact with the patient. An example of a medical device is for instance a CT-scan or an afterloader.
- The second bottom layer is the functional integration layer. This layer is responsible for combining the medical devices and other components into qualified workflow elements. A single workflow element can use multiple devices and a single device can be part of multiple workflow elements.
- The 3^{th} layer of the model is called the treatment supervisor layer. This layer is responsible to determine the feasible workflow steps. This supervisor layer contains a workflow controller, which deals with the concerns of interpreting the protocol, the current state, available workflow elements, authorization, and the commissioning state of the workflow elements.
- The 4^{th} and 5^{th} layers are outside the scope of this report. These layers deal with the concerns of multiple patients, where the lower layers deal with the concerns of a single patient treatment.

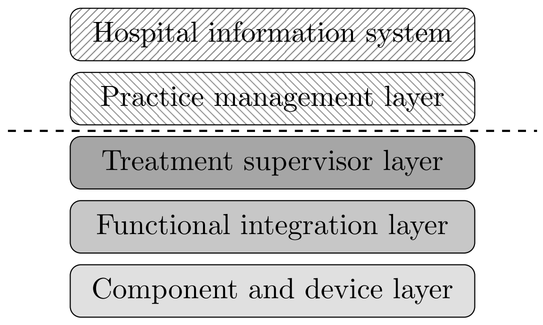
Hospital information system
Practice management layer
Treatment supervisor layer
Functional integration layer
Component and device layer

Figure 1.2.1: System layering, Elekta

The perception of working with multiple devices is changing in the medical field. Previously, medical equipment mostly worked stand alone. Each medical device had its own task and user-interface. However, the need to work adaptively is increasing. This means that the inter-operability between the devices increases. Therefore, it becomes more appropriate to combine multiple devices into a single workflow element.

The American Society for Testing and Materials (ASTM), developed an architecture reference [10], which can be mapped to the layering software architecture of Elekta. The architecture of ASTM is developed to improve the inter-operability of multiple medical devices. Figure 1.2.2 depicts the reference architecture. The gray layers map to the Elekta architecture of Figure 1.2.1.

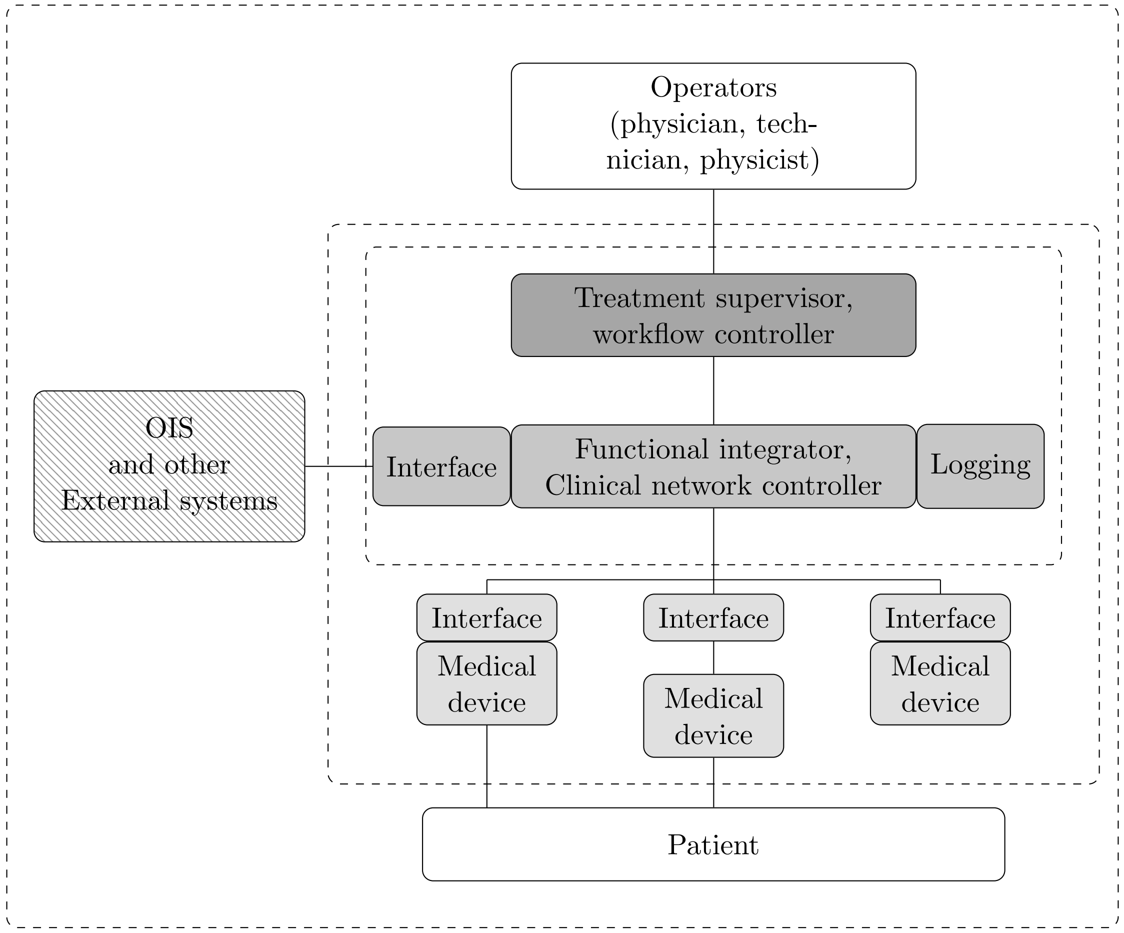

Figure 1.2.2: Architecture overview based on the ASTM F2761-09 ICE architecture

1.3 Related work

Workflow management systems are currently being used in several businesses processes in order to monitor, control, and to schedule tasks to participants. The concept of workflow management emerged in the late eighties, begin nineties.

Before the eighties, the processes were implemented hard-coded into the software applications. The consequence of this approach were increased costs in modifying a workflow or a process. Therefore, it became too expensive to maintain the software applications, because lifetime of processes decreased and the complexity of processes increased. This is the reason for development of a generic system with the purpose to support the definition, execution, registration and control of processes [3].

The Workflow Management Coalition, which is founded in 1993, defined the formal definition of Workflow Management System as: *"A system that defines, creates and manages the execution of workflows through the use of software, running on one or more workflow engines, which is able to interpret the process definition, interact with workflow participants and, where required, invoke the use of IT tools and applications."* [5].

Web service based workflow management systems

Many vendors have developed workflow management systems. A cooperation of companies has developed a Web Service based workflow management language, abbreviated WSBPEL. This language emerged from the cooperation of the two companies IBM and Microsoft. The latest version of WSBPEL, version 2.0, is developed by several companies, *e.g.,* Adobe, SAP, and BEA systems. This version is standardized by the consortium OASIS.

WSBPEL is based on web standards and it makes use of a Extensible Markup Language messaging protocol (XML). The disadvantage of this language is that it is not based on any formal semantics [5]. Several research groups have been working on a formal semantics for WSBPEL. The tools that are developed in order to provide a framework, are based on formal model languages. For instance, Petri nets, guarded automata and labeled transitions systems [12].

Petri net based workflow management systems

A Petri net is a well-founded formal model language, which is represented by a directed bipartite graph. The graph is composed of nodes and connections called places and transitions, which are depicted by rectangles and circles, respectively. A place can contain tokens, which are depicted by black dots. The transitions and places are connected via the arrows in the graph.

The current state of the Petri net is defined by all tokens of the net. Therefore, the Petri net is a state based rather than an event based model [3, 6, 7, 8]. Figure 1.3.1 depicts a model of a simple postorder process, represented by a Petri net.

The Dutch government started a project called Sagitta-2000, which led to an eponymous software platform. The platform is being used in order to manage the customs declarations. This platform is based on mathematical modeling language called Petri net. The benefit of a workflow language based on Petri nets is the formal semantics of the language [8].

T_1 = Receive order
T_2 = Write bill
T_3 = Pack order
T_4 = Ship order

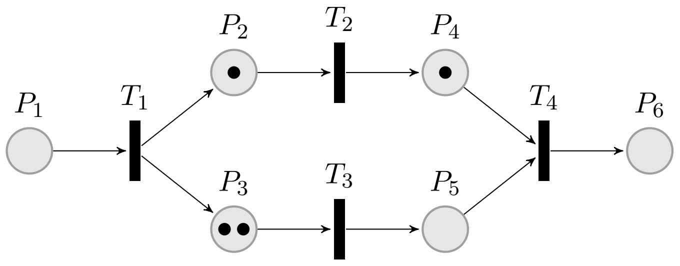

Figure 1.3.1: Petri net of a postorder process

Another Petri net based management system is YAWL, which is the abbreviation for Yet Another Workflow Language. This language is used in a theoretically proven management system, which is developed in corporation of several universities and companies [4].

There are various reasons whether workflow management systems, *e.g.,* YAWL and Sagitta, are based on Petri nets. In the paper of van der Aalst [8] the three main reasons are summarized to choose for a Petri net based workflow system for the Sagitta-2000 platform. The three reasons are:

1. Formal semantics despite the graphical nature.
2. State-based instead of event-based.
3. Abundance of analysis techniques.

The argumentation of the author [8] is sustained by the mathematical analysis and theorems of among others W. Reisig [9]. With use of the formal semantics it is possible to prove the safety properties of a workflow. Hence, with use of a well-founded workflow language, it is possible to increase the confidence of the correctness of a workflow [11].
The graphical nature of the definition of a Petri net is sustained by the unambiguous representation with tokens, places and transitions. The main benefits of the formal semantics and graphical nature are the unambiguous, tool independent, and available analysis techniques.
Moreover, with use of a state based description the workflow management system is able to interpret the enabling conditions of a task. The execution of a task can be enabled automatically, with user input or with a time based event. Secondly, the state-based description is able to handle competitive tasks.

Two tasks are competitive if both tasks are enabled to execute, although only one of the two is allowed to be executed. Hence, if one of the two tasks executes, then the token is removed from the place. Therefore, the other task is not able to execute anymore. Figure 1.3.2 depicts a situation where the two tasks T_1 and T_2 are competitive, because both tasks are enabled, although only one task is allowed to execute. In this situation task

T_1 is executed and the token moves from P_1 to P_2, task T_1 and T_2 are not enabled any more. See Figure 1.3.3.

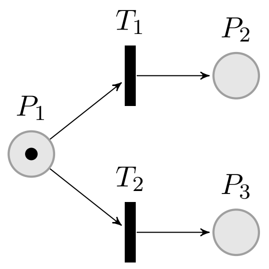

Figure 1.3.2: Tasks T_1 and T_2 are enabled

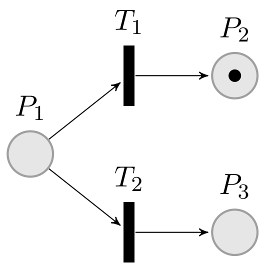

Figure 1.3.3: Task T_1 is executed

The author extends his arguments with the possibility to analyze the performances of the workflows. This leads to the benefit that it is possible to calculate the response times, waiting times, and occupation rates of a specified workflow [8].

Automata based management systems

The semantics of the workflow management system in this report is based on automata theory. As far as we know there is no commercial language that sustains a constraint based workflow management system based on automata theory.

However, there is a thesis [11] that describes a constraint based management system based on the automata theory.

The workflow principle models a task as a finite automata, these automata are shuffled to combine the languages of the tasks. The inter-tasks dependencies are then defined by so called illegal states. The illegal states are disabled in the result of the shuffled automaton [11].

1.4 Brachytherapy

This section describes briefly the aim of brachytherapy in order to provide the reader some background information to understand the medical example, which is discussed later on in this report.

Brachytherapy is an important way to treat patients with cancer. It is a subclass of radiotherapy, which works by radiating the cancer cells in order to destroy the cells of a tumor. Brachytherapy radiates the patient from inside out. This means that the radiation source is placed directly inside or closely next to the tumor. The benefit of brachytherapy in comparison with external radiotherapy is lower exposure of radiation onto the healthy organs, because the radiation has not first to pass healthy tissue before reaching the tumor. [14].

During the brachytherapy treatment it is important that the physician knows where the radiation source is located with respect to the target and the organs at risks (OAR). Therefore, the physician creates contours of the target and OAR on an image acquired of the patient. With the knowledge of the contours he is able to plan and optimize the total dose distribution on the target and OAR. This information is described in a plan. When the plan is approved, the patient can be radiated. In some cases the patient needs multiple doses. This means that a part of the treatment needs to be iterated.

1.5 Structure of report

This report speaks of semantics in order to give meaning to a workflow system suitable for Elekta. The semantics are described by the definitions, theorems and lemmas and substantiated by the proofs in this report.

This report is composed of six sections. Section 2 starts with the common definitions in the field of automata theory. Some definitions are adapted to be applicable to the workflow management system. In order to verify the properties of the definitions, the necessary proofs are provided.

Section 3 describes the concepts of the protocol, these concepts are not common in the field of automata theory. Hence, the concepts are based on the needs of the medical purpose.

The concepts of the protocol are used by the workflow controller, which is explained in section 4. The workflow controller interprets the protocol and determines the feasible workflow steps.

Until section 5 most examples are at a high of abstraction level and these examples are not directly related to the medical purpose. Section 5 describes a practical medical workflow used in the field of brachytherapy. The protocol based on the semantics of this report is evaluate to the intention of the original workflow.

The report ends with a conclusion and some comments to future research.

2 General definitions

This section describes the general definitions, which are used throughout the report. These definitions are common in the field of automata theory, although they are not directly related to the medical field. The ubiquitously known definitions, *e.g.,* intersection, list manipulation, etcetera are excluded from this section. The definitions in this section are adapted in such a manner that they can be applied to the adaptive brachytherapy platform reference architecture.

2.1 Finite automaton

A finite automaton described by formal model language can be expressed as a directed graph. The graph is composed of states and transitions. A state represents the statename or some parameters. They are depicted by circles. The states are connected via transitions. Each transition is composed of a begin and an end state. Furthermore, the transition is labeled with a workflow step. We label the workflow steps of the set W in this section with $\langle p, e\rangle$ to be consequent with the rest of the report. For the definition of the workflow step see section 3.2.

Definition 2.1.1. ***(Automaton)*** *Given a set of workflow steps W. We call a quadruple in the form $A = (S, \rightarrow, s_i, S_g)$ an automaton, where:*

- *S is a finite set of states.*
- *$\rightarrow \subseteq S \times W \times S$ is a set of transition relations.*
- *s_i is the initial state.*
- *S_g is a set of final states.*

An automaton accepts only a predefined alphabet, *i.e.,* the set of workflow steps. An accepting path is an execution order of workflow steps, which lead from the initial state to a final state. The minimal set of all possible accepting paths is called the language of the automaton.

Definition 2.1.2. ***(Automaton language)*** [1]. *Given an automaton $A = (S, \rightarrow, s_i, S_g)$. We define the language $\mathfrak{L}(s)$ of a state $s \in S$ as the minimal set satisfying:*

- *if $s \in S_g$ then $\emptyset \in \mathfrak{L}(s)$.*
- *if $\langle s, \langle p, e\rangle, s'\rangle \in \rightarrow$ and $\sigma \in \mathfrak{L}(s')$ then $\langle p, e\rangle\sigma \in \mathfrak{L}(s)$.*

Definition 2.1.3. ***(Language equivalence)*** [1]. *Given two automata $k = (S_k, \rightarrow_k, s_{i_k}, S_{g_k})$ and $l = (S_l, \rightarrow_l, s_{i_l}, S_{g_l})$. We call the automata k and l language equivalent iff $\mathfrak{L}(s_{i_k}) = \mathfrak{L}(s_{i_l})$.*

2.2 Bisimulation

The definition of bisimulation is included to prove some equivalence properties [2]. Using bisimulation it is possible to express the behavioural equivalence between two states of an automaton.

Bisimulation is based on the following principle, if an action can be performed in a state of an automaton it is also possible to perform the action from a state, which is bisimilar to the first mentioned state. If two states are bisimilar, then all the resulting states must be bisimilar as well [1].

Definition 2.2.1. ***(Bisimulation)*** [1]. *Given two automata $k = (S_k, \rightarrow_k, s_{i_k}, S_{g_k})$ and $l = (S_l, \rightarrow_l, s_{i_l}, S_{g_l})$. A binary relation $R \subseteq S \times S$ is called a strong bisimulation relation iff for all $s \in S_k$ and $t \in S_l$ such that sRt holds, it also holds that:*

1. *if $\langle s, \langle p, e\rangle, s'\rangle \in\rightarrow_k$, then there is a $t' \in S_l$ such that $\langle t, \langle p, e\rangle, t'\rangle \in\rightarrow_l$ with $s'Rt'$.*
2. *if $\langle t, \langle p, e\rangle, t'\rangle \in\rightarrow_l$, then there is a $s' \in S_k$ such that $\langle s, \langle p, e\rangle, s'\rangle \in\rightarrow_k$ with $s'Rt'$.*
3. *$s \in S_{g_k}$ if and only if $t \in S_{g_l}$.*

Two bisimilar states are denoted by $s \underline{\leftrightarrow} t$. Two automata are called strongly bisimilar if the initial states of the automata are bisimilar. We write $k \underline{\leftrightarrow} l$ if and only if $s_{i_k} \underline{\leftrightarrow} s_{i_l}$.

2.3 Determinism

A deterministic automaton does not contain states that have multiple identical outgoing transitions to different states. Figure 2.3.1 and Figure 2.3.2 depict two automata. The right automaton is non deterministic, because it contains two outgoing transitions to two different states with the same workflow step.

Definition 2.3.1. ***(Determinism)*** [1]. *We call an automaton $A = (S, \rightarrow, s_i, S_g)$ deterministic if an only if for all states $s, s', s'' \in S$ it holds that if $\langle s, \langle p, e\rangle, s'\rangle \in\rightarrow$ and $\langle s, \langle p, e\rangle, s''\rangle \in\rightarrow$ then $s' = s''$.*

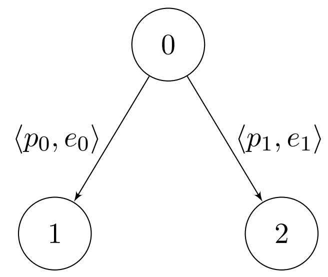

Figure 2.3.1: Deterministic automaton

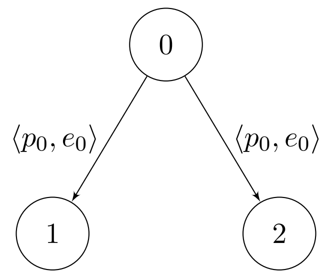

Figure 2.3.2: Non deterministic automaton

2.4 Synchronous product

A synchronous product automaton is based on two automata and the language of this automaton is the intersection of these two automata. The states of the synchronous product automaton is the cartesian product of the two input automata. The set of transitions is defined by the transitions of both automata that contains the same workflow steps. The synchronous product automaton contains a single initial state and goal state, which is again the cartesian product of the initial and goal states.

Definition 2.4.1. ***(Synchronous product)*** *Given two automata $k = (S_k, \rightarrow_k, s_{i_k}, S_{g_k})$ and $l = (S_l, \rightarrow_l, s_{i_l}, S_{g_l})$. We define the synchronous product as a quadruple $k \times l = (S_{k\times l}, \rightarrow_{k\times l}, s_{i_{k\times l}}, S_{g_{k\times l}})$, where:*

- $S_{k\times l} = S_k \times S_l$.
- $\rightarrow_{k\times l} = \{\langle\langle s_k, s_l\rangle, \langle p, e\rangle, \langle s'_k, s'_l\rangle\rangle \mid \langle s_k, \langle p, e\rangle, s'_k\rangle \in \rightarrow_k \wedge \langle s_l, \langle p, e\rangle, s'_l\rangle \in \rightarrow_l\}$.
- $s_{i_{k\times l}} = \langle s_{i_k}, s_{i_l}\rangle$.
- $S_{g_{k\times l}} = S_{g_k} \times S_{g_l}$.

Example 2.4.2. This example describes the synchronous product of two automata k and l and is depicted in Figure 2.4.3. The automaton k contains a self loop with the workflow step $\langle\{a\}, \{a\}\rangle$. Automaton l contains a selfloop with the workflow step $\langle\{a\}, \{b\}\rangle$.
Both self loops will be canceled out by the synchronous product automaton, because they are not accepted by both input automata.

The synchronous automaton contains a considerable number of unreachable states, e.g., $\langle 0, 0\rangle$, $\langle 0, 1\rangle$ and $\langle 1, 3\rangle$. State $\langle 1, 3\rangle$ is unreachable, although the state has an outgoing transition. However, it is not an initial state and it has no incoming transitions.

Given the two automata:
$k = (\{0,1,2,3\}, \{\langle\{a\}, \langle\emptyset, a\rangle, \{a\}\rangle\langle\{a\}, \langle a, b\rangle, \{a,b\}\rangle, \langle\{a,b\}, \langle\emptyset, a\rangle, \{a,b\}\rangle\}, 1, \{3\})$ *and*
$l = (\{0,1,2,3\}, \{\langle\{a\}, \langle a, b\rangle, \{a,b\}\rangle, \langle\{a,b\}, \langle a, b\rangle, \{a,b\}\rangle\}, 1, \{3\})$.

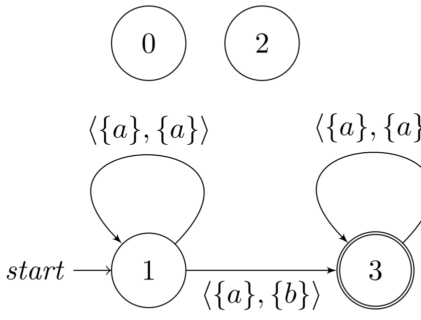

Figure 2.4.1: Direct graph of automaton k

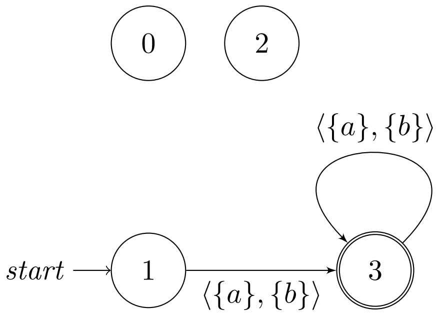

Figure 2.4.2: Direct graph of automaton l

We can construct the synchronous product automaton depicted in Figure 2.4.3 from the two automata of Figure 2.4.1 and 2.4.2.

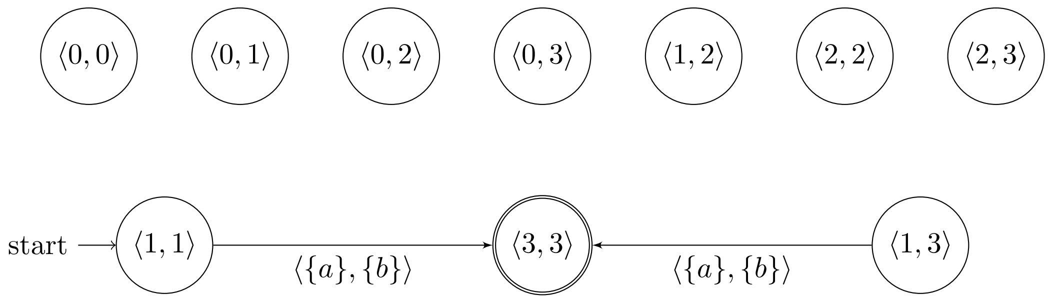

Figure 2.4.3: Direct graph of the automaton $k \times l$

The synchronous product construction is idempotent, commutative and associative. This means applying the function multiple times, or swapping the two input automata, or changing the order of execution does not change the output automaton. The three properties are stated in theorems 2.4.3, 2.4.4, and 2.4.5, respectively.

Theorem 2.4.3. ***(Idempotent)*** *Given an automaton $k = (S, \to, s_i, S_g)$. It holds that the synchronous product automaton $k \times k$ is bisimilar with k:*

$$k \times k \mathrel{\underline{\leftrightarrow}} k$$

Proof. In order to prove $k \times k \mathrel{\underline{\leftrightarrow}} k$, we need to show the existence of a bisimulation relation R that relates $\langle s_i, s_i\rangle$ and s_i. The relation $R = \{\langle\langle s,s\rangle,\langle s\rangle\rangle | s \in S\}$. We need to check that R is a bisimulation relation. Therefore, we need to check the three properties of strong bisimulation from Definition 4.4.1.

1. Suppose the transition $\langle\langle s,s\rangle,\langle p,e\rangle,\langle s',s'\rangle\rangle \in \to_{k\times k}$. According to Definition 2.4.1 we can deduce that: $\langle s,\langle p,e\rangle,s'\rangle \in \to$. Clearly, $\langle s',s'\rangle R\langle s'\rangle$ holds.

2. Suppose the transition $\langle s,\langle p,e\rangle,s'\rangle \in \to$. Applying Definition 2.4.1 on $k \times k$, we can deduce that: $\langle\langle s,s\rangle,\langle p,e\rangle,\langle s',s'\rangle\rangle \in \to_{k\times k}$. Clearly, $\langle s',s'\rangle R\langle s'\rangle$ holds.

3. Suppose $\langle s,s\rangle \in S_{g_{k\times k}}$. According to Definition 2.4.1 we can deduce that: $s \in S_g$ Similarly, it can be shown that if $s \in S_g$, then $\langle s,s\rangle \in S_{g_{k\times k}}$.

□

Theorem 2.4.4. ***(Commutative)*** *Given two automata $k = (S_k, \rightarrow_k, s_{i_k}, S_{g_k})$ and $l = (S_l, \rightarrow_l, s_{i_l}, S_{g_l})$. It holds that the synchronous product automaton $k \times l$ is bisimilar with $l \times k$:*

$$k \times l \leftrightarrows l \times k$$

Proof. In order to prove $S_k \times S_l \leftrightarrows S_l \times S_k$, we need to show the existence of a bisimulation relation R that relates $s_{i_{k\times l}}$ and $s_{i_{l\times k}}$. The relation $R = \{\langle\langle s_k, s_l\rangle, \langle s_l, s_k\rangle\rangle | s_k \in S_k \wedge s_l \in S_l\}$. We need to check that R is a bisimulation relation. Therefore we have to check the three properties of strong bisimulation from definition 4.4.1.

1. Suppose the transition $\langle\langle s_k, s_l\rangle, \langle p, e\rangle, \langle s'_k, s'_l\rangle\rangle \in\rightarrow_{k\times l}$. According to definition 2.4.1 we can deduce that: $\langle s_k, \langle p, e\rangle, s'_k\rangle \in\rightarrow_k$ and $\langle s_l, \langle p, e\rangle, s'_l\rangle \in\rightarrow_l$. By applying definition 2.4.1, we get: $\langle\langle s_l, s_k\rangle, \langle p, e\rangle, \langle s'_l, s'_k\rangle\rangle \in\rightarrow_{l\times k}$. Clearly, $\langle s'_k, s'_l\rangle R \langle s'_l, s'_k\rangle$ holds.

2. This second case is symmetric to the first case and is therefore omitted.

3. Suppose $\langle s_k, s_l\rangle \in S_{g_{k\times l}}$, according to definition 2.4.1 we can deduce that: $s_k \in S_{g_k}$ and $s_l \in S_{g_l}$. By applying definition 2.4.1, it can be obtained that: $\langle s_l, s_k\rangle \in S_{g_{l\times k}}$. Similarly, it can be shown that if $\langle s_l, s_k\rangle \in S_{g_{l\times k}}$, then $\langle s_k, s_l\rangle \in S_{g_{k\times l}}$.

□

Theorem 2.4.5. ***(Associative)*** *Given three automata $k = (S_k, \rightarrow_k, s_{i_k}, S_{g_k})$, $l = (S_l, \rightarrow_l, s_{i_l}, S_{g_l})$, and $m = (S_m, \rightarrow_m, s_{i_m}, S_{g_m})$. It holds that the synchronous product automaton $(k \times l) \times m$ is bisimilar with $k \times (l \times m)$:*

$$(k \times l) \times m \leftrightarrows k \times (l \times m)$$

Proof. In order to prove $(k \times l) \times m \leftrightarrows k \times (l \times m)$, we use a similar structure as the proof of Theorem 2.4.4. Therefore, we need to show the existence of a bisimulation relation R that relates $\langle s_{i_{(k\times l)\times m}}, s_{i_{k\times(l\times m)}}\rangle$. The relation $R = \{\langle\langle s_k, s_l, s_m\rangle, \langle s_l, s_m, s_k\rangle\rangle | s_k \in S_k \wedge s_l \in S_l \wedge s_m \in S_m\}$. We need to check that R is a bisimulation relation. Therefore we have to check the three properties of strong bisimulation from definition 4.4.1.

1. Suppose the transition $\langle\langle s_k, s_l, s_m\rangle, \langle p, e\rangle, \langle s'_k, s'_l, s'_m\rangle\rangle \in\rightarrow_{k\times l\times m}$.
   First, according to definition 2.4.1 we can deduce that: $\langle\langle s_k, sl\rangle, \langle p, e\rangle, \langle s'_k, s'_l\rangle\rangle \in\rightarrow_{k\times l}$ and $\langle s_m, \langle p, e\rangle, s'_m\rangle \in\rightarrow_m$. Secondly, we can deduce from definition 2.4.1 that: $\langle s_k, \langle p, e\rangle, s'_k\rangle \in\rightarrow_k$ and $\langle s_l, \langle p, e\rangle, s'_l\rangle \in\rightarrow_l$. Now, we apply definition 2.4.1 to construct the automaton: $\langle\langle s_l, s_m\rangle, \langle p, e\rangle, \langle s'_l, s'_m\rangle\rangle \in\rightarrow_{l\times m}$. By applying the definition 2.4.1 again we can construct the automaton $\langle\langle s_l, s_m, s_k\rangle, \langle p, e\rangle, \langle s'_l, s'_m, s'_k\rangle\rangle \in \rightarrow_{l\times m\times k}$. Clearly, $\langle s'_k, s'_l, s'_m\rangle R \langle s'_l, s'_m, s'_k\rangle$ holds.

2. This second case is symmetric to the first case and is therefore omitted.

3. Suppose $\langle s_k, s_l, s_m\rangle \in S_{g_{k\times l\times m}}$, according to definition 2.4.1 we can deduce that: $\langle s_k, s_l\rangle \in S_{g_{k\times l}}$ and $s_m \in S_{g_m}$. Furthermore, we can deduce that: $s_k \in S_{g_k}$ and $s_l \in S_{g_l}$. By applying definition 2.4.1 twice, we get: $\langle s_l, s_m, s_k\rangle \in S_{g_{l\times m\times k}}$. Similarly, it can be shown that if $\langle s_l, s_m, s_k\rangle \in S_{g_{l\times m\times k}}$, then $\langle s_k, s_l, s_m\rangle \in S_{g_{k\times l\times m}}$.

□

The synchronous product is applied to construct an automaton that accepts only the sequences of workflow steps that are feasible in both automata. Therefore, Theorem 2.4.6 must hold.

Theorem 2.4.6. ***(Language intersection)*** *Given two automata $k = (S_k, \rightarrow_k, s_{i_k}, S_{g_k})$, and $l = (S_l, \rightarrow_l, s_{i_l}, S_{g_l})$. It holds that the language of the synchronous product automaton $k \times l$ is equivalent with the intersection of the language of k and l:*

$$\mathfrak{L}(k \times l) = \mathfrak{L}(k) \bigcap \mathfrak{L}(l)$$

Proof. First, we prove the case from left to right. $\mathfrak{L}(k \times l) \subseteq \mathfrak{L}(k) \bigcap \mathfrak{L}(l)$. Suppose the two goal states $s_{g_l} \in S_{g_l}$ and $s_{g_k} \in S_{g_k}$. Furthermore, suppose an accepting path of the automaton $k \times l = s_{i_{k \times l}} \xrightarrow{\langle p,e \rangle}_{k \times l} s'_{k \times l} \xrightarrow{\langle p',e' \rangle}_{k \times l} s''_{k \times l}, \ldots, s'''_{k \times l} \xrightarrow{\langle p'',e'' \rangle}_{k \times l} s_{g_{k \times l}}$. From this path we can deduce the word $\sigma = \xrightarrow{\langle p,e \rangle}_{k \times l}, \xrightarrow{\langle p',e' \rangle}_{k \times l}, \ldots, \xrightarrow{\langle p'',e'' \rangle}_{k \times l} \in \mathfrak{L}(k \times l)$. From definition 2.4.1 can be deduced that $\xrightarrow{\langle p,e \rangle}_k, \xrightarrow{\langle p',e' \rangle}_k, \ldots, \xrightarrow{\langle p'',e'' \rangle}_k \in \mathfrak{L}(k)$ and $\xrightarrow{\langle p,e \rangle}_l, \xrightarrow{\langle p',e' \rangle}_l, \ldots, \xrightarrow{\langle p'',e'' \rangle}_l \in \mathfrak{L}(l)$ and therefore $\sigma \in \mathfrak{L}(k) \bigcap \mathfrak{L}(l)$.

Secondly, we prove the case from right to left $\mathfrak{L}(k) \bigcap \mathfrak{L}(l) \subseteq \mathfrak{L}(k \times l)$. Suppose the two goal states $s_{g_l} \in S_{g_l}$ and $s_{g_k} \in S_{g_k}$. Furthermore, suppose an accepting path for automaton $k = s_{i_k} \xrightarrow{\langle p,e \rangle}_k s'_k \xrightarrow{\langle p',e' \rangle}_k s''_k, \ldots, s'''_k \xrightarrow{\langle p'',e'' \rangle}_k s_{g_k}$ and the accepting path for automaton $l = s_{i_l} \xrightarrow{\langle p,e \rangle}_l s'_l \xrightarrow{\langle p',e' \rangle}_l s''_l, \ldots, s'''_l \xrightarrow{\langle p'',e'' \rangle}_l s_{g_l}$. By applying definition 2.4.1 we get $\xrightarrow{\langle p,e \rangle}_{k \times l}, \xrightarrow{\langle p',e' \rangle}_{k \times l}, \ldots, \xrightarrow{\langle p'',e'' \rangle}_{k \times l} \in \mathfrak{L}(k \times l)$.

□

The synchronous product automaton of Definition 2.4.1 is deterministic if both input automata are deterministic. In order to prove this property we need to show that Theorem 2.4.7 holds.

Theorem 2.4.7. ***(Deterministic synchronous product)*** *Given two deterministic automata $k = (S_k, \rightarrow_k, s_{i_k}, S_{g_k})$, and $l = (S_l, \rightarrow_l, s_{i_l}, S_{g_l})$. Then the synchronous product $k \times l$ is also deterministic.*

Proof. Suppose the two transitions $\langle \langle s_k, s_l \rangle, \langle p, e \rangle, \langle s'_k, s'_l \rangle \rangle \in \rightarrow_{k \times l}$ and $\langle \langle s_k, s_l \rangle, \langle p, e \rangle, \langle s''_k, s''_l \rangle \rangle \in$ $\rightarrow_{k \times l}$. From definition 2.4.1 we can deduce that $\langle s_k, \langle p, e \rangle, s'_k \rangle \in \rightarrow_{s_k}$, $\langle s_l, \langle p, e \rangle, s'_l \rangle \in \rightarrow_{s_l}$, $\langle s_k, \langle p, e \rangle, s''_k \rangle \in \rightarrow_{s_k}$ and $\langle s_l, \langle p, e \rangle, s''_l \rangle \in \rightarrow_{s_l}$. Because, k and l are deterministic automata, it follows from definition 2.3.1 that $s'_k = s''_k$ and $s'_l = s''_l$. Therefore, $\langle s'_k, s'_l \rangle = \langle s''_k, s''_l \rangle$ and it can be concluded that the synchronous product automaton is also deterministic.

□

2.5 Function to remove infeasible states

An automaton can contain infeasible states. These states do not contribute to the accepting paths of an automaton. In other words, these states are not reachable from the initial state or they are not connected to a final state. In some situations it is not desirable that these states are included into the automaton. In Definition 2.5.1 and Definition 2.5.2 two functions are defined that are able to remove the infeasible states from the automaton.

Definition 2.5.1. ***(Remove the transitions that do not lead to a final goal)*** *Given an automaton $x = (S_x, \rightarrow_x, s_{i_x}, S_{g_x})$. We define the function $\mathscr{G}(x) = (S, \rightarrow, s_i, S_g)$, where:*

- $S = \{s_j \,|\, s_j \xrightarrow{\langle p,e\rangle}_x s'_x \xrightarrow{\langle p',e'\rangle}_x s''_x, \ldots, s'''_x \xrightarrow{\langle p'',e''\rangle}_x s_{g_x} \wedge s_{g_x} \in S_{g_x}\} \bigcup \{s_{i_x}\}$.
- $\rightarrow = \{\langle s_j, \langle p, e\rangle, s'_j\rangle \,|\, s_j, s'_j \in S \wedge \langle s_j, \langle p, e\rangle, s'_j\rangle \in \rightarrow_x\}$.
- $s_i = s_{i_x}$.
- $S_g = S_{g_x}$.

If the initial state s_i can not reach a final goal from the set S_g, then the automaton is invalid.

Definition 2.5.2. ***(Remove unreachable states of the automaton)*** *Given an automaton $x = (S_x, \rightarrow_x, s_{i_x}, S_{g_x})$. We define the function $\mathscr{I}(x) = (S, \rightarrow, s_i, S_g)$, where:*

- $S = \{s_j \,|\, s_{i_x} \xrightarrow{\langle p,e\rangle}_x s'_x \xrightarrow{\langle p',e'\rangle}_x s''_x, \ldots, s'''_x \xrightarrow{\langle p'',e''\rangle}_x s_j\}$.
- $\rightarrow = \{\langle s_j, \langle p, e\rangle, s'_j\rangle \,|\, s_j, s'_j \in S \wedge \langle s_j, \langle p, e\rangle, s'_j\rangle \in \rightarrow_x\}$.
- $s_i = s_{i_x}$.
- $S_g = S_{g_x} \bigcap S$.

Both functions do not affect the deterministic behaviour of a deterministic automaton. This statement is trivial, because both functions remove transitions. Therefore, it can be concluded that an automaton remains deterministic after applying the two functions.

Example 2.5.3. *Recall the result of Example 2.4.2. Figure 2.5.1 shows the result after applying the function $\mathscr{G}$ on the automaton of Figure 2.4.3.*

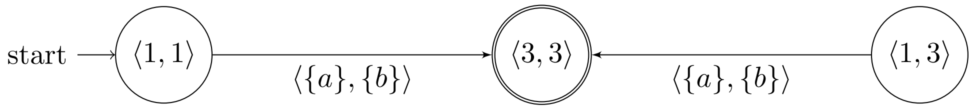

Figure 2.5.1: Direct graph of the automaton $\mathscr{G}(k \times l)$

Example 2.5.4. *The automaton of Example 2.5.3 can be further reduced by using the function* $\cancel{I}$. *The resulting automaton is depicted in Figure 2.5.2.*

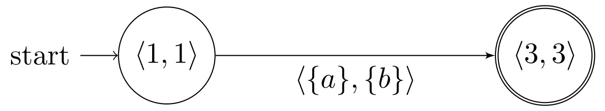
$$\begin{tikzcd}\text{start} \arrow[r] & \langle 1,1\rangle \arrow[r, "\langle\{a\},\{b\}\rangle"'] & \langle 3,3\rangle\end{tikzcd}$$

Figure 2.5.2: Direct graph of the automaton $\cancel{I}(\cancel{G}(k \times l))$

3 Patient space

During the execution of a clinical workflow, many decisions must be taken in order to choose the appropriate workflow steps to reach the clinical intent of a treatment. The availability of the workflow steps depend on the history of the treatment, the status of the patient, and the protocol that is being used.

The patient space describes the complete space of workflow steps that can be carried out during a treatment, *i.e.,* all workflow sequences that can be performed from the start to the clinical goal of the treatment.

The patient space will be expressed as an automaton. The definition of this automaton will be defined in the following subsections.

3.1 Conditions

The status of the patient is expressed as a set of conditions. A condition is not always directly related to the physical condition of the patient. Hence, a condition can be an image set, a result of a measurement or an answer to a question.

The conditions in the report are mostly of a high abstraction level *e.g.,* a, b, c, etcetera. We write C for the set of all conditions.

In this report it is assumed that the condition *stop* is always in the set of conditions. This condition will be applied to terminate a case that is no longer useful. This concept is explained in detail in section 4.6.

3.2 Workflow step

The clinical workflow for the treatment of a patient is a sequence of separated workflow steps. A workflow step can be executed when its input conditions are satisfied. After the execution of a workflow step new results are available. The new results unified with the old results, describe the condition of the patient at that moment of time. The input requirements for the succeeding workflow steps are examined on these conditions. Therefore, the input and output conditions of the workflow steps are considered as pre and post conditions of the workflow steps. The pre and post conditions describe the value dependency between the tasks, as explained in section 1.4.

Definition 3.2.1. ***(Workflow step)*** *Given a set of conditions C. We define a workflow step as a pair $w \in P \times E$, where:*

- $P \subseteq 2^C$ *is a set of preconditions;* i.e., *input conditions.*
- $E \subseteq 2^C$ *is a set of effects;* i.e., *post conditions.*

A special workflow step is called w_{stop}, this workflow step will be used in section 4.6.

Definition 3.2.2. ***(Workflow step end case)*** *We define the workflow step* $w_{stop} = P_{stop} \times E_{stop}$ *as an end case workflow step, where*

- $P_{stop} = \emptyset$
- $E_{stop} = \{stop\}$

Example 3.2.3. *In the domain of brachytherapy a common workflow step is contouring an organ. During this workflow step, the medical specialist emphasizes a specific organ in an image set by drawing a contour around this organ. The medical specialist can only start this workflow step if the image set is available. The result of the workflow step is the contour of the organ. Therefore, The precondition of this workflow element is defined by the image set and the postcondition is a contour.*

3.3 Workflow element

A workflow element is a set of workflow steps. It combines workflow steps that are related to each other. For instance, the workflow element contour can consist of the workflow steps contour target and contour organs at risk.

Definition 3.3.1. ***(Workflow element)*** *We call the set in the form* $wfe \in 2^{w}$ *a workflow element.*

3.4 Protocol

A protocol describes the workflow steps that are allowed to be executed during the treatment of a patient, and it describes the external dependency rules between the workflow steps. Therefore, the protocol describes the best practice for a specific clinical intent.

The execution ordering of workflow steps is the result of the inter-task dependency. Hence, it depends on the pre and post conditions of the workflow steps, the protocol rules, and the quality conditions, where the treatment must reside.

The protocol rules describe the ordering between two arbitraire workflow steps w_1 and w_2. There are four types of rules. The meaning of these rules are described below:

- $w_1 \succ w_2$: Eventually after the execution of w_1, w_2 needs to be executed.
- $w_1 \gg w_2$: Directly after the execution of w_1, w_2 needs to be executed.
- $w_1 \prec w_2$: Somewhere before the execution of w_2, w_1 must have been executed.
- $w_1 \ll w_2$: Directly before the execution of w_2, w_1 must have been executed.

The protocol can restrict the feasible workflow steps of the workflow space, based on its rules. The protocol can be very loose, such that many different workflows lead to the clinical goal, or the protocol can be very strict, such that the patient space only consist of one single path. Hence, when a protocol is very strict then there is also less space for parallelism.

It is possible that the protocol restricts the patient space in a manner that the protocol is in conflict with the final state and therefore the clinical goal can not be reached. This type of protocol is called an invalid protocol.

Definition 3.4.1. ***(Protocol)*** *Given a finite set of conditions C, preconditions $P \subseteq C$ and effects $E \subseteq C$. We define a protocol as a sixtuple $Pr = (W, \succ, \gg, \prec, \ll, Q)$, where:*

- *$W \subseteq P{\times}E$ is a finite set of workflow steps, which are allowed to be used.*
- *$\succ \subseteq W{\times}W$ is a set of pairs representing the eventually after rules between the workflow steps.*
- *$\gg \subseteq W{\times}W$ is a set of pairs representing the directly after rules between the workflow steps.*
- *$\prec \subseteq W{\times}W$ is a set of pairs representing the eventually before rules between the workflow steps.*
- *$\ll \subseteq W{\times}W$ is a set of pairs representing the directly before rules between the workflow steps.*
- *Q is a triple of quality conditions, $Q = (QC, Q_i, Q_g)$, where:*
  - *$QC \subseteq C$ is the set of quality conditions in which the patient must reside.*
  - *$Q_i \subseteq QC$ are the quality conditions of the initial state.*
  - *$Q_g \subseteq QC$ are the quality conditions of the finial state.* i.e., *conditions of the clinical goal.*

Throughout the report several examples are provided based on the protocol of the following example.

Example 3.4.2. This protocol consists of two workflow steps. The pre and postcondition of the first workflow step are equal as, both have the condition a. The second workflow step has the precondition a and the postcondition b.

The protocol contains one single rule, namely: directly after workflow $\langle a, a\rangle$ the workflow step $\langle a, b\rangle$ must be executed.

The initial state consists of the condition a and the end state consists of the conditions a and b.

Given the conditions $C = \{a, b\}$, initial conditions $I_C = \{a\}$. We construct the protocol Pr.

$$Pr = \{\{\langle a, a\rangle, \langle a, b\rangle\}, \{\}, \{\langle\langle a, a\rangle, \langle a, b\rangle\rangle\}, \{\}, \{\}, \{\{a, b\}, \{a\}, \{a, b\}\}\}.$$

3.5 Workflow space

The workflow space is an automaton, which forms the base of the patient space. The workflow space represents all the workflows that are possible with use of the provided workflow steps of the protocol. As result the workflow space describes all possible execution sequences and is not influenced by the rules of the protocol. The initial state of the workflow space contains the initial conditions defined by the protocol.

Definition 3.5.1. ***(Workflow Space)*** *Given a set of conditions C, a set of initial patient conditions $I_C \subseteq C$ and a protocol $Pr = (W, \succ, \gg, \prec, \ll, (QC, Q_i, Q_g))$, where $Q_i \subseteq I_C$. We can express all the possible transitions and states, which are reachable by the workflow steps of a protocol, as an automaton $W_s = (S,\ \rightarrow,\ s_i,\ s_g)$, where:*

- *$S = 2^C$ is a powerset of conditions, which represents the states of the patient model.*
- *$\rightarrow = \{\langle s, \langle p, e\rangle, s \cup e\rangle \,|\, p \subseteq s \wedge \langle p, e\rangle \in W\}$ is a set of relations between two states of the patient model.*
- *$s_i = I_C$ is the initial state.*
- *$S_g = \{s_g \,|\, s_g \supseteq I_C \cup Q_g\}$.*

The workflow automaton is always deterministic. In order to prove this statement, we need to prove Theorem 3.5.2.

Theorem 3.5.2. ***(Deterministic workflow space)*** *Every workflow space W_s is deterministic.*

Proof. Suppose the two transitions $\langle s, \langle p, e\rangle, s'\rangle \in \rightarrow_{W_s}$ and $\langle s, \langle p, e\rangle, s''\rangle \in \rightarrow_{W_s}$. From definition 3.5.1 we can deduce that $s' = e \cup s$ and $s'' = e \cup s$, therefore $s' = s''$ and W_s is deterministic.

□

Example 3.5.3. The workflow space described in this example is based on the protocol of Example 3.4.2. The initial state consists of the condition a and the end state must consist of the conditions a and b.

Given the conditions $C = \{a, b\}$, initial conditions $I_C = \{a\}$ and a protocol Pr

$$Pr = \{\{\langle a, a\rangle, \langle a, b\rangle\}, \{\}, \{\langle\langle a, a\rangle, \langle a, b\rangle\rangle\}, \{\}, \{\}, \{\{a, b\}, \{a\}, \{a, b\}\}\}$$

We can construct the following Workflow space automaton, which is also depicted in Figure 3.5.1:

$$W_s = \{\{\{\}, \{a\}, \{b\}, \{a, b\}\}, \{\langle\{a\}, \langle a, a\rangle, \{a\}\rangle, \langle\{a\}, \langle a, b\rangle, \{a, b\}\rangle\} \cup$$

$$\{\langle\{a, b\}, \langle a, a\rangle, \{a, b\}\rangle, \langle\{a, b\}, \langle a, b\rangle, \{a, b\}\rangle\}, \{a\}, \{a, b\}\}$$

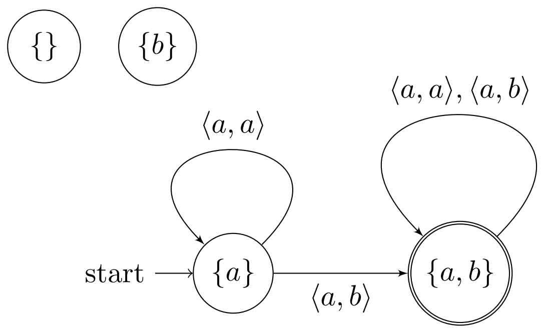

Figure 3.5.1: Graph representation Workflow Space W_s

3.6 Rule automata

The patient space is expressed as an automaton. This automaton must be restricted by the protocol rules. Therefore, we need to transform the protocol rules into automata. The rule automaton constrains the workflow space such that the patient space is restricted with respect to the inter-task dependencies.

Definition 3.6.1. ***(Automaton eventually after,*** $A^{\succ}$***)*** *Given a protocol* $Pr = (W, \succ, \gg, \prec, \ll, Q)$. *We define for each pair* $\langle\langle p_0, e_0\rangle, \langle p_1, e_1\rangle\rangle \in \succ$ *an automaton* $A^{\succ}$*, where*

$$\begin{aligned} A^{\succ} =& (\{0,1\}, \{\langle 0, \langle p_0, e_0\rangle, 1\rangle, \langle 1, \langle p_1, e_1\rangle, 0\rangle\} \cup \\ & \{\langle 0, \langle p_i, e_i\rangle, 0\rangle | \langle p_i, e_i\rangle \in W \wedge \langle p_i, e_i\rangle \neq \langle p_0, e_0\rangle\} \cup \\ & \{\langle 1, \langle p_j, e_j\rangle, 1\rangle | \langle p_j, e_j\rangle \in W \wedge \langle p_j, e_j\rangle \neq \langle p_1, e_1\rangle\}, 0, \{0\}) \end{aligned}$$

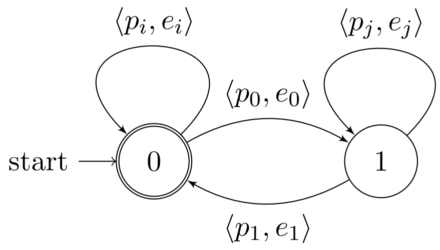

Figure 3.6.1: Automaton eventually after, $\langle p_0, e_0\rangle \succ \langle p_1, e_1\rangle$

Definition 3.6.2. ***(Automaton directly after,*** $A^{\gg}$***)*** *Given a protocol* $Pr = (W, \succ, \gg, \prec, \ll, Q)$. *We define for each pair* $\langle\langle p_0, e_0\rangle, \langle p_1, e_1\rangle\rangle \in \gg$ *an automaton* $A^{\gg}$*, where*

$$\begin{aligned} A^{\gg} =& (\{0,1\}, \{\langle 0, \langle p_0, e_0\rangle, 1\rangle, \langle 1, \langle p_1, e_1\rangle, 0\rangle\} \cup \\ & \{\langle 0, \langle p_i, e_i\rangle, 0\rangle | \langle p_i, e_i\rangle \in W \wedge \langle p_i, e_i\rangle \neq \langle p_0, e_0\rangle\}, 0, \{0\}) \end{aligned}$$

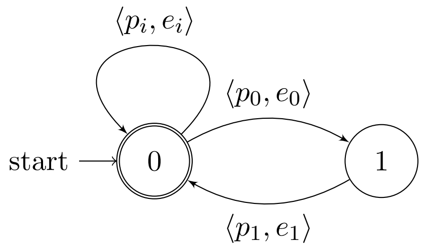

Figure 3.6.2: Automaton directly after, $\langle p_0, e_0\rangle \gg \langle p_1, e_1\rangle$

Definition 3.6.3. ***(Automaton eventually before,*** $A^{\prec}$***)*** *Given a protocol* $Pr = (W, \succ, \gg, \prec, \ll, Q)$*. We define for each pair* $\langle\langle p_0, e_0\rangle, \langle p_1, e_1\rangle\rangle \in \prec$ *an automaton* $A^{\prec}$*, where*

$$A^{\prec} = (\{0,1\}, \{\langle 0, \langle p_0, e_0\rangle, 1\rangle, \langle 1, \langle p_1, e_1\rangle, 0\rangle, \langle 1, \langle p_j, e_j\rangle, 1\rangle | \langle p_j, e_j\rangle \in W \wedge \langle p_j, e_j\rangle \neq \langle p_1, e_1\rangle\} \cup \{\langle 0, \langle p_i, e_i\rangle, 0\rangle | \langle p_i, e_i\rangle \in W \wedge \langle p_i, e_i\rangle \neq \langle p_0, e_0\rangle \wedge \langle p_i, e_i\rangle \neq \langle p_1, e_1\rangle\}, 0, \{0,1\})$$

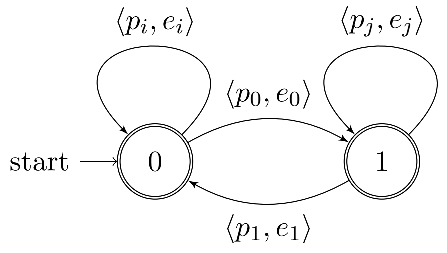
$\langle p_i, e_i\rangle$
$\langle p_j, e_j\rangle$
$\langle p_0, e_0\rangle$
start
0
1
$\langle p_1, e_1\rangle$

Figure 3.6.3: Automaton eventually before, $\langle p_0, e_0\rangle \prec \langle p_1, e_1\rangle$

Definition 3.6.4. ***(Automaton directly before,*** $A^{\ll}$***)*** *Given a protocol* $Pr = (W, \succ, \gg, \prec, \ll, Q)$*. We define for each pair* $\langle\langle p_0, e_0\rangle, \langle p_1, e_1\rangle\rangle \in \ll$ *an automaton* $A^{\ll}$*, where*

$$A^{\ll} = (\{0,1\}, \{\langle 0, \langle p_0, e_0\rangle, 1\rangle, \langle 1, \langle p_i, e_i\rangle, 0\rangle | \langle p_i, e_i\rangle \in W \wedge \langle p_i, e_i\rangle \neq \langle p_0, e_0\rangle\} \cup \{\langle 1, \langle p_0, e_0\rangle, 1\rangle, \langle 0, \langle p_j, e_j\rangle, 0\rangle | \langle p_j, e_j\rangle \in W \wedge \langle p_j, e_j\rangle \neq \langle p_0, e_0\rangle \wedge \langle p_j, e_j\rangle \neq \langle p_1, e_1\rangle\}, 0, \{0,1\})$$

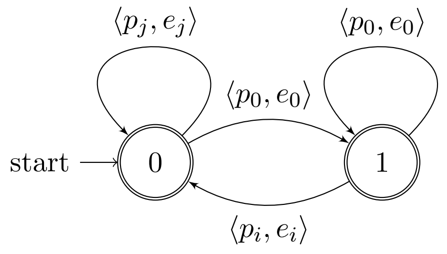
$\langle p_j, e_j\rangle$
$\langle p_0, e_0\rangle$
$\langle p_0, e_0\rangle$
start
0
1
$\langle p_i, e_i\rangle$

Figure 3.6.4: Automaton directly before, $\langle p_0, e_0\rangle \ll \langle p_1, e_1\rangle$

3.7 Patient space automaton

The patient space is the synchronous product of the workflow space and all the rule automata, which is finally reduced with use of the two cancelation functions.

The patient space is only valid when there is a path from the initial state to the goal state. Furthermore, the set of quality states needs to contain all the effects *i.e.,* post conditions of the workflow steps, which are allowed to executed within the protocol. Otherwise it is possible to have a workflow step that leads to a state outside the set of quality conditions.

Definition 3.7.1. ***(Patient space)*** *Given a protocol* $Pr = (W, \succ, \gg, \prec, \ll, Q)$*. Let* $a_0, \cdots, a_k \in \succ$*,* $b_0, \cdots, b_l \in \gg$*,* $c_0, \cdots, c_m \in \prec$*, and* $d_0, \cdots, d_n \in \ll$ *be the pairs, which describes the rules between the workflow steps, and let* $A_{a_0}, \cdots, A_{a_k}$*,* $A_{b_0}, \cdots, A_{b_l}$*,* $A_{c_0}, \cdots, A_{c_m}$ *and* $A_{d_0}, \cdots, A_{d_n}$ *be all the related automata of the rules. We define the patient space as the synchronous product automaton of the Workflow Space automaton and all the automata of the eventually after, direct after, eventually before and direct before rules, and the cancelation of the states which are unreachable from the initial state, and the states which does not reach the goal state.*

$$Ps = \cancel{I}(\cancel{G}(W_s \times A_{a_0} \times \cdots \times A_{a_k} \times A_{b_0} \times \cdots \times A_{b_l} \times A_{c_0} \times \cdots \times A_{c_m} \times A_{d_0} \times \cdots \times A_{d_n}))$$

The patient space is a deterministic automaton, because the workflow space and protocol rule automata are deterministic, the synchronous product of a two deterministic automata is deterministic, and the results of the cancelation functions are deterministic. See Theorem 2.4.7, Theorem 3.5.2, and section 2.5.

Example 3.7.2. ***(Patient space)*** Recall the workflow space W_s of Example 3.5.3 based on the protocol Pr of Example 3.4.2. In the following example is the patient space constructed based on these two examples.

Given the workflow space W_s *and the protocol* Pr*, where*

$$\begin{aligned} Pr =& \{\{\langle a,a\rangle, \langle a,b\rangle\}, \{\}, \{\langle\langle a,a\rangle, \langle a,b\rangle\rangle\}, \{\}, \{\}, \{\{a,b\}, \{a\}, \{a,b\}\}\}, \textit{ and} \\ W_s =& \{\{\{\}, \{a\}, \{b\}, \{a,b\}\}, \{\langle\{a\}, \langle a,a\rangle, \{a\}\rangle, \langle\{a\}, \langle a,b\rangle, \{a,b\}\rangle\} \cup \\ & \{\langle\{a,b\}, \langle a,a\rangle, \{a,b\}\rangle, \langle\{a,b\}, \langle a,b\rangle, \{a,b\}\rangle\}, \{a\}, \{a,b\}\} \end{aligned}$$

We construct the protocol rule automaton $A^{\gg}$*, where*

$$A^{\gg} = (\{0,1\}, \{\langle 0, \langle a,a\rangle, 1\rangle, \langle 1, \langle a,b\rangle, 0\rangle, \langle 0, \langle a,b\rangle, 0\rangle\}, 0, \{0\})$$

We can calculate the synchronous product of $W_s \times A^{\gg}$*, where*

$$\begin{aligned} W_s \times A^{\gg} =& (\{\langle\{\}, 0\rangle, \langle\{a\}, 0\rangle, \langle\{b\}, 0\rangle, \langle\{a,b\}, 0\rangle, \langle\{\}, 1\rangle, \langle\{a\}, 1\rangle, \langle\{b\}, 1\rangle, \langle\{a,b\}, 1\rangle\}, \\ & \{\langle\langle\{a\}, 0\rangle, \langle a,a\rangle, \langle\{a\}, 1\rangle\rangle, \langle\langle\{a\}, 1\rangle, \langle a,a\rangle, \langle\{a\}, 1\rangle\rangle\} \cup \\ & \{\langle\langle\{a,b\}, 0\rangle, \langle a,b\rangle, \langle\{a,b\}, 0\rangle\rangle, \langle\langle\{a,b\}, 1\rangle, \langle a,b\rangle, \langle\{a,b\}, 0\rangle\rangle\} \cup \\ & \{\langle\langle\{a,b\}, 0\rangle, \langle a,b\rangle, \langle\{a,b\}, 0\rangle\rangle, \langle\langle\{a,b\}, 1\rangle, \langle a,b\rangle, \langle\{a,b\}, 0\rangle\rangle\}, \\ & \langle a, 0\rangle, \{\langle\{a,b\}, 0\rangle\}) \end{aligned}$$

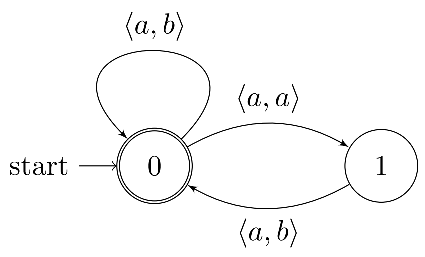

Figure 3.7.1: Automaton directly after, $\langle a,a\rangle \gg \langle a,b\rangle$

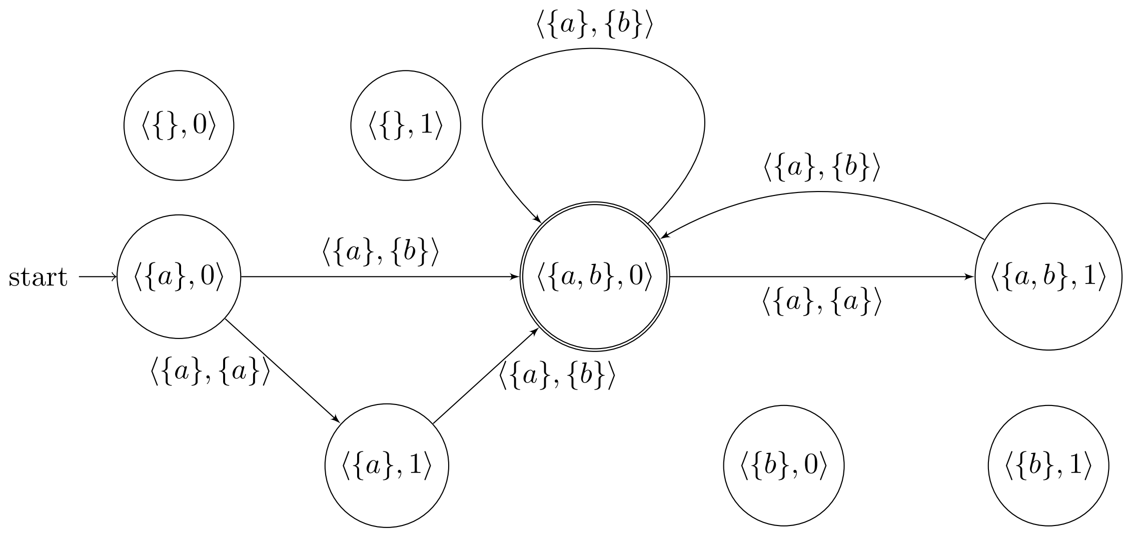

Figure 3.7.2: Direct graph of the automaton $W_s \times A^{\gg}$

The construction of the patient space PS is almost finished. The last step is to apply the functions $\cancel{I}$ and $\cancel{G}$ on the synchronous product $W_s \times A^{\gg}$. These two functions remove the infeasible states of the automaton.

We can construct $PS = \cancel{I}(\cancel{G}(W_s \times A^{\gg}))$, *where*

$$\begin{aligned}PS =(&\{\langle\{a\},0\rangle,\langle\{a,b\},0\rangle,\langle\{a\},1\rangle,\langle\{a,b\},1\rangle\},\\
&\{\langle\langle\{a\},0\rangle,\langle a,a\rangle,\langle\{a\},1\rangle\rangle,\langle\langle\{a\},1\rangle,\langle a,a\rangle,\langle\{a\},1\rangle\rangle\}\cup\\
&\{\langle\langle\{a,b\},0\rangle,\langle a,b\rangle,\langle\{a,b\},0\rangle\rangle,\langle\langle\{a,b\},1\rangle,\langle a,b\rangle,\langle\{a,b\},0\rangle\rangle\}\cup\\
&\{\langle\langle\{a,b\},0\rangle,\langle a,b\rangle,\langle\{a,b\},0\rangle\rangle,\langle\langle\{a,b\},1\rangle,\langle a,b\rangle,\langle\{a,b\},0\rangle\rangle\},\\
&\langle a,0\rangle,\{\langle\{a,b\},0\rangle\})\end{aligned}$$

$$\begin{tikzcd}
\text{start} \arrow[r] & \langle \{a\}, 0 \rangle \arrow[rr, "\langle \{a\}, \{b\} \rangle"] \arrow[dr, swap, "\langle \{a\}, \{a\} \rangle"] & & \langle \{a, b\}, 0 \rangle \arrow[loop above, "\langle \{a\}, \{b\} \rangle"] \arrow[r, swap, "\langle \{a\}, \{a\} \rangle"] & \langle \{a, b\}, 1 \rangle \arrow[l, bend right, swap, "\langle \{a\}, \{b\} \rangle"] \\
& & \langle \{a\}, 1 \rangle \arrow[ur, swap, "\langle \{a\}, \{b\} \rangle"] & &
\end{tikzcd}$$

Figure 3.7.3: Direct graph of the patient space PS

4 Workflow controller

The aim of a workflow controller is to determine, activate, control and finally terminate the feasible workflow steps. The feasible workflow steps depend on the current status of the treatment.

The workflow controller needs to interpret the patient space and the current state in this patient space in order to determine the feasible workflow steps. The workflow controller determines its current state with support of a case space.

The case space is an automaton, which represents the status of the treatment based on all the cases of a patient. A case represents a part of the treatment. It is composed of three separate parts: the history including the conditions of the treatment; the current protocol; and the active workflow steps. With these three parts it is possible to determine the feasible workflow steps.

The treatment can consist of multiple cases. These cases can be manipulated by predefined functions. All possible manipulations of the cases are recorded by the case space.

This section starts with the explanation of time stamps. Subsequently, the definition of a case and the feasible workflow steps of such a case are given in sections 4.2 and 4.3. The manipulations on the cases are tracked by the case space. The definition of the case space is given in section 4.4. The manipulations are the results of the functions described in the sections 4.5 until 4.8.

4.1 Time stamps

In order to evaluate and log the treatment, we need some notion of time. Therefore, we introduce a time stamp, which represents the year, date, and time. We call the set T the set of time stamps. An element of the set T defines the start and finish time of a workflow step.

4.2 Case

A single case describes a piece of the treatment. It describes the history, the active workflow steps and the current protocol of the case. The history is stored in a list and it contains all states and workflow steps that are carried out during the treatment. The history list can only be extended. Thus, when states and workflow steps are added to the history list, they become immutable.

The current active workflow steps are maintained in a set with their related start times.

When a workflow step is accomplished it will be added to the history list and deleted from the set of active workflow steps.

Definition 4.2.1. ***(Case)*** *Given a protocol $Pr = (W, \succ, \gg, \prec, \ll, Q)$, the patient space $PS = (S, \rightarrow, s_i, S_g)$ based on the protocol Pr, a set of states $S_h \subseteq S$, a set of workflow steps $W_h \subseteq W$, a set of time stamps T related to the start and finish time of the workflow steps. We call a triple in the form $CS = (HL, AW, Pr)$ a case, where*

- *$HL = (s_i, t_1, s_1, t_2, s_2, \ldots, s_{n-1}, t_n, s_n)$ is a history list, where*
  - *$s_i, \ldots, s_n \in S_h$, represents the states that are reached during the treatment of the patient, where s_i is the initial state of the history list and s_n is the current state of the patient.*
  - *$t_1, \ldots, t_n \in T \times W_h \times T$, represents the transitions between the states. Those transitions are composed of workflow steps, which are carried out during the treatment and the related start and finish time of the workflow step.*
- *$AW \subseteq T \times W$ is a set of active workflow steps, which represents the set of currently running workflow steps with their related start times.*

We define the function $head(CS) = s_n$.

Definition 4.2.2. ***(Valid history list)*** *Given a protocol $Pr = (W, \succ, \gg, \prec, \ll, Q)$, the patient space $PS = (S, \rightarrow, s_i, S_g)$ based on the protocol Pr, a set of states $S_h \subseteq S$, a set of workflow steps $W_h \subseteq W$, a set of time stamps T related to the start and finish time of the workflow steps, and a history list $HL = (s_i, t_1, s_1, t_2, s_2, \ldots, s_{n-1}, t_n, s_n)$, where*

$$t_1 = \langle st_1, w_1, ft_1 \rangle, t_2 = \langle st_2, w_2, ft_2 \rangle, \ldots, t_n = \langle st_n, w_n, ft_n \rangle.$$

We say that the history list HL is a valid history list, if and only if it holds that:

- $s_i \xrightarrow{w_1} s_1 \xrightarrow{w2} s_2, \cdots, s_{n-1} \xrightarrow{w_n} s_n$,
- *For all k in the range from 1 till n, $st_k < ft_k$, and*
- $ft_1 < ft_2 < \cdots < ft_n$.

4.3 Feasible workflow steps of a case

The set of feasible workflow steps represents the workflow steps that are executable for a specific case. This set of feasible workflow steps is called AT. A workflow step can only be activated or started, when it is present in the set of feasible workflow steps.

The set of feasible workflow steps depends on the workflow steps that are active at the current moment in time. If the set of active workflow steps AW is empty, *i.e.,* no workflow steps are active, then the set AT consist of all the workflow steps that are feasible from the head state of the current case with respect to the patient space. This situation is depicted in Figure 4.3.1.

If the set of active workflow steps AW is not empty *i.e.,* at least one workflow step is active, then another workflow step is only feasible if it is feasible from the head node of the

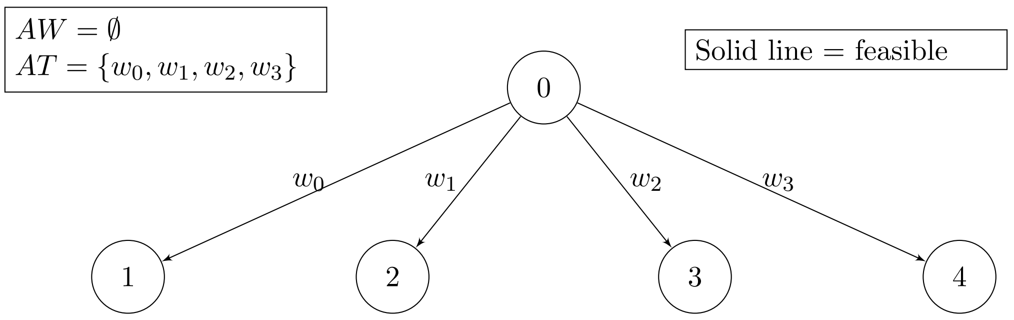

Figure 4.3.1: Feasible workflow steps if no workflow step is active

case and both workflow steps reach the same state after the termination of both workflow steps.

This restriction allows concurrent workflow steps, as long as the workflow steps that are executed concurrently are aiming for the same treatment goal. The restriction must hold for all the workflow steps that are currently active in the specific case.

Figure 4.3.2 depicts a situation where one workflow step is active. It can be seen that only workflow step w_2 is feasible.

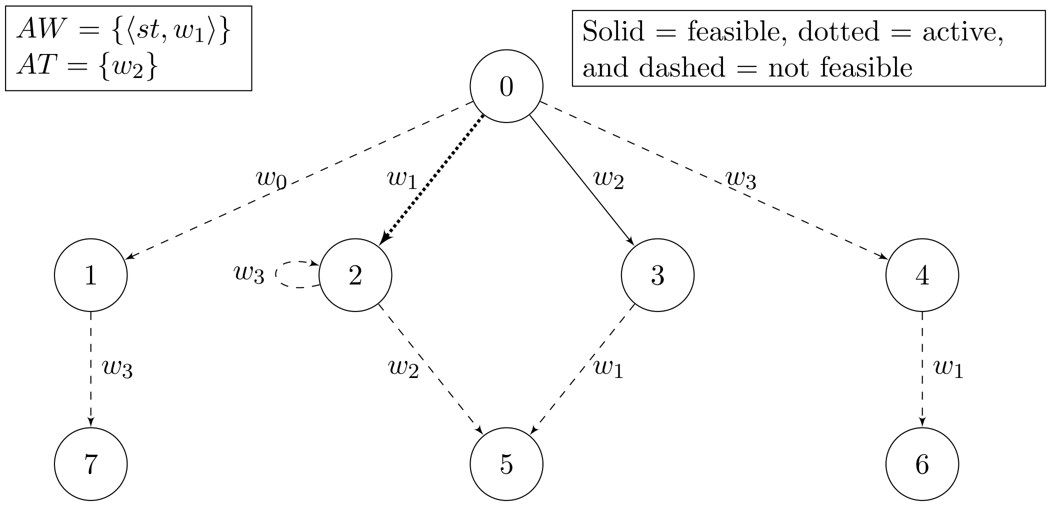
$AW = \{\langle st, w_1\rangle\}$
$AT = \{w_2\}$
Solid = feasible, dotted = active, and dashed = not feasible
0
w_0
w_1
w_2
w_3
1
2
3
4
w_3
w_3
w_2
w_1
w_1
7
5
6

Figure 4.3.2: Feasible workflow steps if a workflow step is active

Definition 4.3.1. ***(Predicate to determine the feasible workflow steps)*** *Given a patient space $PS = (S, \rightarrow, s_i, S_g)$, a finite set of workflow steps W, and the state $s \in S$. We define the predicate $Check(W, s)$ as follows:*

$$Check(W,s) = \begin{cases} \forall_{x \in W}, \exists_{s' \in S} : \langle s, x, s' \rangle \in \rightarrow & \text{if } Size(W) \leqslant 1, \\ \forall_{x \in W}, \forall_{y \in W \backslash \{x\}}, \exists_{s', s'', s''' \in S} : \langle s, x, s' \rangle, \langle s, y, s'' \rangle, & \text{otherwise.} \\ \quad \langle s'', x, s''' \rangle, \langle s', y, s''' \rangle \in \rightarrow \wedge Check(W \backslash \{x\}, s'). \end{cases}$$

If the set W validates the predicate $Check(W, s)$, then the set has the property that every workflow step can be permutated with every other workflow step from the set W and still reach the same end state. Therefore, the order of execution does not influence the end state that will be reached, or in other words: all possible paths from state s composed of all the workflow steps of set W that are executed once, will end in the same state. This property is satisfied in the proof of Theorem 4.3.2.

Theorem 4.3.2. ***(Permutation)*** *Given a patient space $PS = (S, \rightarrow, s_i, S_g)$, a finite set of workflow steps $W = \{w_1, \ldots, w_n\}$ and a state $s \in S$, which satisfy the function $Check(W, s)$. Call σ the path of workflow steps in the form $\sigma = permutation(w_1, \ldots, w_n)$. The following statement holds:*

$$\exists_{s' \in S} : s \xrightarrow{\sigma} s'$$

Proof. Lemma 4.3.3 states that, if we swap two neighbor workflow steps in the path σ we still end in the same state. Lemma 4.3.4 states that every possible sequence can be constructed by swapping the workflow steps of the path σ pairwise. Hence, we can determine that every possible sequence of the path σ starting from the state s ends in the same state, say s'.

□

Theorem 4.3.2 makes use of Lemma 4.3.3, which states that every workflow element in a path from state s can be permutated with another workflow step and still end in the same state. Figure 4.3.3 depicts the situation of Lemma 4.3.3. The path σ leads to a state t. From this state there are two paths that reach the same state. The order of the workflow steps in the paths are permutated. Note that the workflow steps w_i and w_j are not in the path σ.

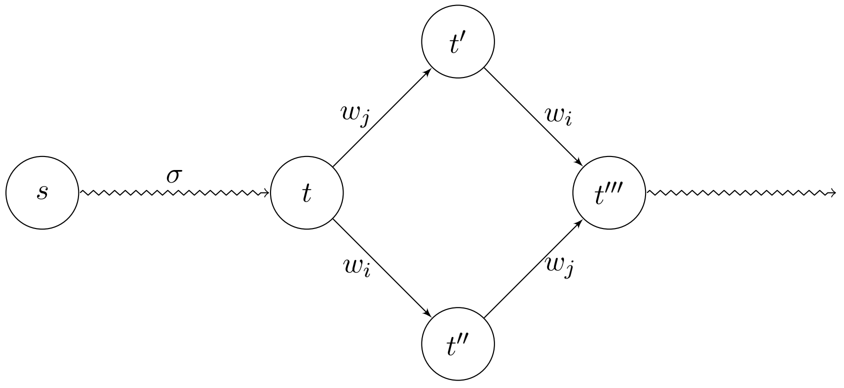

Figure 4.3.3: Permutation of workflow step w_i and w_j

Lemma 4.3.3. *Given a patient space $PS = (S, \rightarrow, s_i, S_g)$, a finite set of workflow steps $W = \{w_1, \ldots, w_n\}$ and a state $s \in S$, which satisfies the predicate $Check(W, s)$. We define the path σ as a path that starts from state s and consists of feasible workflow steps of set W excepts the workflow steps w_i, w_j. Hence, $\sigma = \langle s, w_k, s' \rangle, \ldots, \langle s'', w_l, t \rangle$, where $w_k, \ldots, w_l \in W \backslash \{w_i, w_j\}$ and $w_k \neq \ldots \neq w_l$. The following lemma holds if the predicate $Check(W, s)$ is valid:*

$$\forall_{w_i, w_j \in W \wedge w_i \neq w_j}, \exists_{u,v,w \in S} : \langle t, w_i, u \rangle, \langle u, w_j, w \rangle, \langle t, w_j, v \rangle, \langle v, w_i, w \rangle \in \rightarrow$$

Proof. The workflow steps w_i and w_j must be two different workflow steps. Therefore, we reason about a set larger than 1. We will use induction on the length of the path σ.

Base If σ is empty, then $s = t$. Hence, we validate the predicate $Check(W, t)$. Definition 4.3.1 contains two cases, In our case is the set W greater than 1. Thus, it follows that $\langle t, w_i, u \rangle, \langle u, w_j, w \rangle, \langle t, w_j, v \rangle, \langle v, w_i, w \rangle \in \rightarrow$ exists.

Induction Step If σ is not empty, then the predicate $Check(W, s)$ is valid. We can determine by Definition 4.3.1, that the predicate $Check(W', t)$ holds, where $W' = W \backslash \{w_k, \ldots, w_l\}$. The set W' is larger than 1, because $W' \supseteq \{w_i, w_j\}$. We can determine from the predicate $Check(W', t)$ that $\langle t, w_i, u \rangle, \langle u, w_j, w \rangle, \langle t, w_j, v \rangle, \langle v, w_i, w \rangle \in \rightarrow$ exists.

□

Lemma 4.3.4. *Given a path of workflow steps $\sigma = (w_1, w_2, \ldots, w_n)$. We state that we can construct every sequence with the workflow steps of the path by swapping the workflow steps pairwise. In other words, each workflow step can only swap with its neighbors workflow step.*

Proof. We will prove Lemma 4.3.4 by induction on the length of the path σ.

Base We take as base a length of two elements. Hence, $\sigma = (w_1, w_2)$. It is trivial that we can create every possible sequence, because we swap w_1 and w_2 and we have all possible sequences.

Induction step We assume that we can construct every sequence for $\sigma = (w_1, w_2, \ldots, w_n)$. Now, we have to prove that it holds for $\sigma = (w_1, w_2, \ldots, w_n, w_{n+1})$.

We can swap every workflow step from w_1 till w_{n+1} by the induction hypothesis. Hence, we can swap every workflow step to the place of w_n and we can swap w_{n+1} with the element on place w_n. Therefore we can determine that we can create every possible sequence.

□

Definition 4.3.5. ***(Feasible workflow steps)*** *Given a case $CS = (HL, AW, Pr)$, the head state $s = head(CS)$, where $stop \notin s$, and the patient space $PS = (S, \rightarrow, s_i, S_g)$ based on the protocol $Pr = (W, \succ, \gg, \prec, \ll, Q)$ and a set of workflow steps $AW_w = \{w \mid \langle st, w \rangle \in AW\}$. We define the set $AT \subseteq W$ of workflow steps that can be executed from the head node of the case.*

$$AT = \{x \mid x \in W \backslash AW_w \wedge Check(AW_w \cup \{x\}, s)\}$$

If we execute all workflow elements from the set AW and an additional workflow step from the set AT, then we will still end in a state which is an element of the set S.

Corollary 4.3.6. *Given a case $CS = (HL, AW, Pr)$ and the head state $s = head(CS)$, a patient space PS based on the protocol $Pr = (W, \succ, \gg, \prec, \ll, Q)$. We can determine the set AT based on the set AW and the patient space PS. Call σ the path of workflow steps in the form $\sigma = permutation(w_1, \ldots, w_n, w_x)$, where $\{w_1, \ldots, w_n\} = \{w \mid \langle st, w\rangle \in AW\}$ and $w_x \in AT$. It holds that:*

$$\exists s' \in S : s \xrightarrow{\sigma} s'$$

4.4 Case space

The case space is used as an event logging database. Hence, the case space saves all the manipulations that are generated during the treatment of a patient. Therefore, the entire treatment history can be obtained from the case space.

The case space is represented by a finite automaton, where the states represent the cases and the transitions represent the manipulations on these cases. The head states of the case space represents the cases, which are currently active.

The case space changes when workflow steps are activated or terminating. The case space can also be extended by new cases and a protocol of a case can be switched. All this changes are described by so called manipulations. These manipulations are the result of the functions described in the sections from 4.5 until 4.8.

The following functions are defined:

- *StartWFS*.
- *TerminateWFS*.
- *SwitchProtocol*.
- *TerminateUnexpWFS*.
- *StartBranch*.

Definition 4.4.1. ***(Manipulations)*** *We define the set of manipulations as*

$$\begin{aligned} M = &\{StartWFS,\ TerminateWFS,\ SwitchProtocol,\ TerminateUnexpWFS\} \cup \\ &\{StartBranch,\ TerminateCase\}. \end{aligned}$$

Definition 4.4.2. ***(Case Space Automaton)*** *Given a set of manipulations M and a set of cases CSP. We call a quadruple of the form $CSA = (S, \rightarrow, s_b, S_h)$ a Case Space Automaton, where:*

- *$S = CSP$ is a finite set of states.*
- *$\rightarrow \subseteq S \times M \times S$ is a set of transition relations.*
- *$s_b \in S$ is the initial state of the treatment.*
- *$S_h \subseteq S$ is the set of head states.*

The treatment of a patient starts always with an empty case space. This is an automaton without any transitions and the initial state contains only the conditions that are initially known. The definition of an empty Case Space Automaton is given in Definition 4.4.3.

Definition 4.4.3. ***(Empty Case Space Automaton)*** *Given a protocol Pr, a set of initial conditions I_C, and the patient space $PS = (S, \rightarrow, s_i, S_g)$ based on the protocol Pr and the set of initial conditions I_C. We call the case $CS = \langle s_i, \emptyset, Pr\rangle$ the new case. We define the quadruple of the form $CSA_e = (S, \rightarrow, s_i, s_g)$ as an empty case space automaton, where:*

- *$S = \{CS\}$.*
- *$\rightarrow = \emptyset$.*
- *$s_b = CS$.*
- *$S_h = \{CS\}$.*

4.5 Execution of a workflow step

The execution of a workflow step is carried out in three steps. The first step is selecting a workflow step from the set of feasible workflow steps AT, which is explained in section 4.3. The second step describes the starting procedure of the selected workflow step and the last step describes the termination of the workflow step.

Definition 4.5.1 defines the start of a workflow step. When the selected workflow step is activated, it will directly be added to the list of running workflow steps AW, together with the current time stamp. This action will be carried out by the function *StartWFS*.

Definition 4.5.1. ***(Start workflow step)*** *Given the case space automaton $CSA_x = (S_x, \rightarrow_x, s_{b_x}, S_{h_x})$, a state $CS_x \in S_{h_x}$, where $CS_x = \langle HL_x, AW_x, Pr_x \rangle$, the set of manipulations M, the set of feasible workflow steps AT based on the case CS_x, a selected workflow step $w_m \in AT$, and the related start time $st_m \in T$. We call the new state $s_h = \langle HL_x, AW_x \cup \{\langle st_m, w_m \rangle\}, Pr_x \rangle$. We define the function $StartWFS = (S, \rightarrow, s_b, S_h)$, where*

- $S = S_x \cup \{s_h\}$.
- $\rightarrow = \rightarrow_x \cup \{\langle CS_x, StartWFS, s_h \rangle\}$.
- $s_b = s_{b_x}$.
- $S_h = (S_{h_x} \backslash \{CS_x\}) \cup \{s_h\}$.

If the workflow step is successfully executed and the expected post conditions are reached, then the history list of the case is not longer up to date. Therefore, the workflow step will be deleted from the running set and it will be added to the history list together with the newly reached state. This handling mechanism is performed by the function *TerminateWFS*.

Definition 4.5.2. ***(Terminating workflow step)*** *Given the case space automaton $CSA_x = (S_x, \rightarrow_x, s_{b_x}, S_{h_x})$, a state $CS_x \in S_{h_x}$, where $CS_x = \langle HL_x, AW_x, Pr_x \rangle$, the set of manipulations M, the state $s = head(CS_x)$, the patient space $PS = (S, \rightarrow, s_i, S_g)$ based on protocol Pr_x, a terminating workflow step w_m and the related start time st_m, where $\langle st_m, w_m \rangle \in AW_x$, the finishing time $ft_m \in T$ of workflow step w_m, and the transition $\langle s, w_m, s' \rangle \in \rightarrow$. We call $s_h = \langle ((HL_x + \langle st_m, w_m, ft_m \rangle) + s'), AW_x \backslash \{\langle st_m, w_m \rangle\}, Pr_x \rangle$ the new state of the case space.*

We define the function $TerminateWFS = (S, \rightarrow, s_b, S_h)$, where

- $S = S_x \cup \{s_h\}$.
- $\rightarrow = \rightarrow_x \cup \{\langle CS_x, TerminateWFS, s_h \rangle\}$.
- $s_b = s_{b_x}$.
- $S_h = (S_{h_x} \backslash \{CS_x\}) \cup \{s_h\}$.

4.6 Switching protocol

During the treatment it is possible to change the protocol of the treatment. This can happen, when for instance, the user decides to change the goal of the treatment.

When a protocol is switched, a new case is started and the current case will terminate with a so called *STOP*-state. This action can be performed with the function *SwitchProtocol*.

Definition 4.6.1. ***(Switch protocol)*** *Given the case space automaton $CSA_x = (S_x, \rightarrow_x, s_{b_x}, S_{h_x})$, a state $CS_x \in S_{h_x}$, where $CS_x = \langle HL_x, AW_x, Pr_x \rangle$ and $Pr_x = (W_x, \succ_x, \gg_x, \prec_x, \ll_x, Q_x)$, the set of manipulations M, the state $s = head(CS_x)$, the set of conditions C used to construct the protocol Pr_x , the set of conditions of the head state $HC = s \bigcap C$, the start and finish time $st, ft \in T$ of workflow step w_{stop}, the new protocol $Pr = (W, \succ, \gg, \prec, \ll, (QC, Q_i, Q_g))$, where $Q_i \subseteq HC$ and $HC \subseteq QC$, and the patient space $PS = (S, \rightarrow, s_i, S_g)$, where $HC \subseteq s_i$. We call the terminating state $s_y = \langle ((HL_x + \langle st, w_{stop}, ft \rangle)+ (s \bigcup \{stop\})), \emptyset, Pr_x \rangle$ and the state with the new protocol $s_h = \langle s_i, \emptyset, Pr \rangle$. We define the function $SwitchProtocol = (S, \rightarrow, s_b, S_h)$, where*

- $S = S_x \bigcup \{s_h, s_y\}$.
- $\rightarrow = \rightarrow_x \bigcup \{\langle CS_x, SwitchProtocol, s_y \rangle, \langle CS_x, TerminateCase, s_h \rangle\}$
- $s_b = s_{b_x}$
- $S_h = (S_{h_x} \backslash \{CS_x\}) \bigcup \{s_h\}$

4.7 Exception handling

In section 4.5 the assumption is made that the post conditions of a terminating workflow step are equal to the post conditions of the started workflow step. However, it is possible that an unexpected workflow step terminates.

An unexpected workflow step occurs when an active workflow step executed erroneously. This means that for instance a workflow step is aborted. An unexpected workflow step is not a member of the set of active workflow steps AW. Although, the unexpected workflow step is always linked to a workflow step that is in the set of active workflow steps.

There are two possible scenarios when the unexpected workflow step occurs, namely:

1. The unexpected workflow step was in the set of feasible workflow steps before termination, or
2. The unexpected workflow step did not reside within the protocol, and therefore it is not an element of the set of feasible workflow steps.

The termination of the first scenario is almost equal to the termination of an expected workflow element, because the workflow controller has to delete the related workflow step from the set of active workflow steps and it has to add the unexpected workflow step and post condition to the history list. This termination mechanism is performed by the function *TerminateUnexpWFS*.

Definition 4.7.1. ***(Terminate unexpected workflow step)*** *Given the case space automaton $CSA_x = (S_x, \rightarrow_x, s_{b_x}, S_{h_x})$, a state $CS_x \in S_{h_x}$, where $CS_x = \langle HL_x, AW_x, Pr_x \rangle$, the set of manipulations M, the state $s = head(CS_x)$, the patient space $PS = (S, \rightarrow, s_i, S_g)$ based on protocol Pr_x, the set of feasible workflow steps AT based on the case CS_x, an unexpected terminating workflow step $w_u \in AT$, the related workflow step w_m, the related start time st_m, where $\langle st_m, w_m \rangle \in AW_x$, the finishing time $ft_m \in T$ of workflow step w_m, and the transition $\langle s, w_u, s' \rangle \in \rightarrow$. We call the new state $s_h = \langle ((HL_x + \langle st_m, w_u, ft_m \rangle) + s'),$ $AW_x \backslash \{\langle st_m, w_m \rangle\}, Pr_x \rangle$. We define the function $TerminateUnexpWFS = (S, \rightarrow, s_b, S_h)$, where*

- $S = S_x \bigcup \{s_h\}$.
- $\rightarrow = \rightarrow_x \bigcup \{\langle CS_x, TerminateUnexpWFS, s_h \rangle\}$.
- $s_b = s_{b_x}$.
- $S_h = (S_{h_x} \backslash \{CS_x\}) \bigcup \{s_h\}$.

The second scenario is more complicated, because the unexpected workflow step was not allowed to run, otherwise it was an element of the set AT. Therefore, it is not possible to continue the treatment, because there is a state reached that is not in the patient space.

This scenario will be handled in the following way. The current case will be closed and the user needs to start a new case with a protocol that includes the side effects of the unexpected workflow. This action can be performed with the function *SwitchProtocol*, which is explained in section 4.6.

If the current case contains workflow steps in the set of active workflow steps, then the user has to decide how he will terminate these workflow steps. This is out of the scope of this report.

4.8 Branching

The workflow controller supports a branching mechanism, which is comparable to the principle of the branching mechanism in a software version control repository management system. In such a system branching is supported to modify software objects in parallel along different branches to avoid interference. If in one of the branches a software object is modified, then the final object will be merged to the main branch.

Parallelism can also be beneficial in the medical field. For example, multiple algorithms are required to calculate the pareto point of a dose rate distribution. Those calculations can be performed in parallel to reduce the calculation time. A second benefit can be found in the medical research, because from each state in the history it is possible to analyse what should happen if different choices were made during the treatment.

The branching mechanism that is supported by the workflow controller is slightly different in comparison with the branching methodology of the software management system in the sense that it does not support the concept of merging to a main branch.

With branching it is feasible to conceive an extension from each state, which is in the case space. This implies that a new case can be added to the case space with the conditions and history of the state where the branch is started. This new case can continue with the current protocol or it can switch to a new protocol.

It is feasible to start a single branch or multiple branches with the use of a workflow step. The workflow step will start and supervise the subbranches. The workflow step will wait till all branches terminate and it is then also able to terminate.
The case, which starts the branch is called the primary branch. Therefore the protocol of this case is the "global" protocol. The global protocol is responsible for all the subcases. Therefore, it is able to overview all the meta data in those subcases.
The subcases contains "local" protocols. The local protocol is responsible for the accomplishment of the goal in a single branch. It is conceivable that a local protocol starts a new branch with the use of a workflow step, then the local protocol overviews those subbranches.

Definition 4.8.1. ***(Start new branch)*** *Given the case space automaton* $CSA_x = (S_x, \rightarrow_x, s_{b_x}, S_{h_x})$*, a state* $CS_x \in S$*, where* $CS_x = \langle HL_x, AW_x, Pr_x \rangle$*, and* $Pr_x = (W, \succ, \gg, \prec, \ll, Q)$*, the set of manipulations* M*, the head state* $Bs = head(CS_x)$*, the set of conditions of the current protocol* $PC = \{P \cup E \mid \langle P, E \rangle \in W\}$*, the set of conditions of the head state* $BC = Bs \cap PC$*, the new protocol* $Pr_y = (W_y, \succ_y, \gg_y, \prec_y, \ll_y, (QC_y, Q_{i_y}, Q_{g_y}))$*, where* $Q_{i_y} \subseteq BC$ *and* $BC \subseteq QC_y$*, and the patient space* $PS = (S, \rightarrow, s_i, S_g)$ *based on protocol* Pr_y*, where* $BC \subseteq s_i$*. We call the new branch state* $s_b = \langle s_i, \emptyset, Pr_y \rangle$*. We define the function* $StartBranch = (S, \rightarrow, s_b, S_h)$*, where*

- $S = S_x \cup \{s_b\}$.
- $\rightarrow = \rightarrow_x \cup \{\langle CS_x, Branch, s_b \rangle\}$.
- $s_b = s_{b_x}$.
- $S_h = S_{h_x} \cup \{s_b\}$.

The function *StartBranch* can be executed through a workflow step. The effect of this workflow step is equal to the union of the goals of the protocols that are started in the branches.

Definition 4.8.2. ***(Workflow branch)*** *Given the protocols* $Pr_1 = (W_1, \succ_1, \gg_1, \prec_1, \ll_1, (QC_1, Q_{i_1}, Q_{g_1})), \ldots, Pr_n = (W_n, \succ_n, \gg_n, \prec_n, \ll_n, (QC_n, Q_{i_n}, Q_{g_n}))$ *and the function StartBranch.*

We define the workflow step $w_{branch} = P \times E$*, where*

- $P = Q_{i_1} \cup Q_{i_2} \cup \ldots \cup Q_{i_n}$.
- $E = Q_{g_1} \cup Q_{g_2} \cup \ldots \cup Q_{g_n}$.

Example 4.8.3. ***(Branching)*** Figure 4.8.1 shows an example of branching. It represents a treatment with multiple treatment plans. The global protocol contains the workflow element that creates four different plans in separated branches. After activating the four branches, the global protocol will wait until the meta data of the four plans is available. When each branch reaches its goal and terminates, then the workflow controller can provide a new workflow step based on the rules of the global protocol.

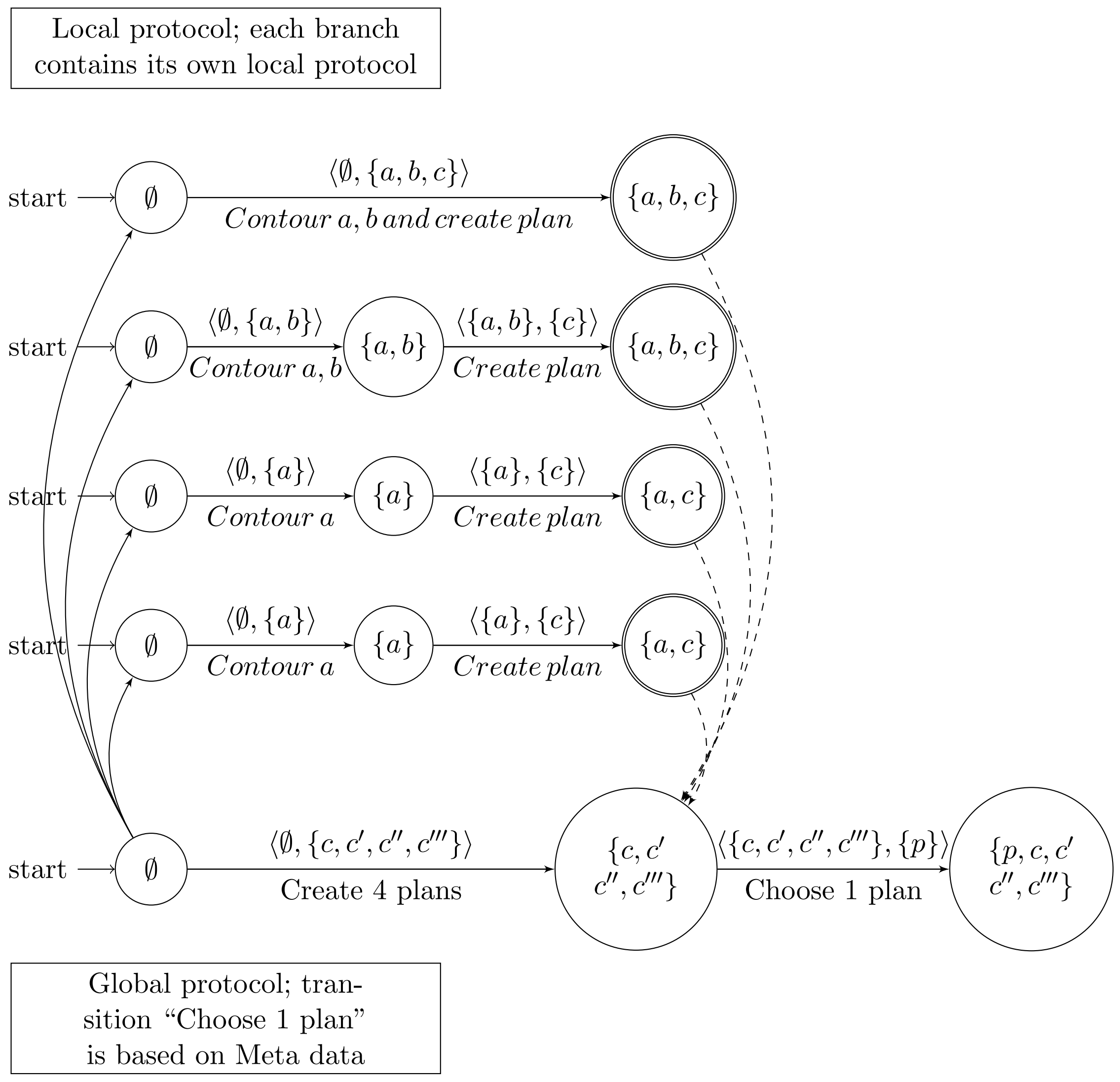

Figure 4.8.1: Example of workflow "create four treatment plans"

4.9 Concurrency

The workflow rule $w_0 \gg w_1$ states that directly after termination of workflow step w_0, workflow step w_1 needs to be executed. Therefore, no other workflow element may execute in parallel with w_0. In order to prove this statement we need to show that Theorem 4.9.1 holds.

Theorem 4.9.1. ***(Concurrency)*** *Given the case space automaton $CSA = (S, \rightarrow, s_b, S_h)$, a state $CS \in S_h$, where $CS = \langle HL, AW, Pr \rangle$, which is only manipulated with the functions of the report. Given the head state $s = head(CS)$, the patient space PS based on the protocol $Pr = (W, \succ, \gg, \prec, \ll, Q)$, which includes the protocol rule $w_0 \gg w_1$.*
It holds that:

$$\text{If } \langle st_0, w_0 \rangle \in AW \text{ then } AW = \{\langle st_0, w_0 \rangle\}.$$

Proof. We prove that for all functions on a Case Space Automaton, Theorem 4.9.1 holds. The functions that are defined in the report are:

1. *StartWFS*, see Definition 4.5.1.
2. *TerminateWFS*, see Definition 4.5.2.
3. *TerminateUnexpWFS*, see Definition 4.7.1.
4. *SwitchProtocol*, see Definition 4.6.1.
5. *StartBranch*, see Definition 4.8.1.

Theorem 4.9.1 holds trivially for functions 2 and 3, because these functions remove elements from the set AW, so if the theorem holds before executing the functions it also holds after the execution. We can also determine that it holds after applying functions 4 and 5, because the case in the new states are empty cases after applying these functions.

Now, we only need to prove the theorem for function 1. The set AW is defined in Definition 4.5.1 as: $AW = AW_x \cup \{\langle st_m, w_m \rangle\}$, where $w_m \in AT$. Therefore, we have to prove two cases:

Case 1: If $\langle st_0, w_0 \rangle \in AW_x$ then $\{\langle st_0, w_0 \rangle\} = AW_x$. This rule is not applicable, because $w_m \notin AT$. Hence, $AT = \emptyset$ following Lemma 4.9.2.

Case 2: If $\langle st_0, w_0 \rangle \in \{\langle st_m, w_m \rangle\}$ then $\langle st_0, w_0 \rangle = \langle st_m, w_m \rangle$. Therefore $m = 0$. Now, we have to prove that $AW_x \subseteq \{\langle st_0, w_0 \rangle\}$. $w_0 \in AT$, so $AW_x = \emptyset$ following Lemma 4.9.3.

□

Lemma 4.9.2. *Given a case $CS = (HL, AW, Pr)$, the head state $s = head(CS)$, a patient space PS, which is based on the protocol $Pr = (W, \succ, \gg, \prec, \ll, Q)$, which includes the protocol rule $w_0 \gg w_1$, and the set of feasible workflow steps AT. It holds that:*

$$\text{If } \{\langle st_0, w_0 \rangle\} = AW \text{ then } AT = \emptyset$$

Proof. We can determine from Definition 4.3.5 that $w_0 \notin AT$ as $w_0 \in AW_w$. The lemma follows from the following observation, which we must prove:

$$\forall x \in W\backslash\{w_0\} : Check(\{w_0, x\}, s) = False.$$

We choose an arbitrary workflow step $w_k \in W\backslash\{w_0\}$ for x. The size of the set $\{w_0, w_k\}$ is two. Therefore, we have to prove that:

$$(\exists_{s',s'',s'''\in S} : \langle s, w_k, s'\rangle, \langle s, w_0, s''\rangle, \langle s'', w_k, s'''\rangle, \langle s', w_0, s'''\rangle \in\rightarrow \wedge Check(\{w_0\}, s')) \wedge$$

$$(\exists_{s',s'',s'''\in S} : \langle s, w_0, s'\rangle, \langle s, w_k, s''\rangle, \langle s'', w_0, s'''\rangle, \langle s', w_k, s'''\rangle \in\rightarrow \wedge Check(\{w_k\}, s')) = False.$$

We weaken the predicate by substituting the second part by $True$ and in the first part we substitute the predicate $Check(\{w_0\}, s')$ by $True$, hence:

$$(\exists_{s',s'',s'''\in S} : \langle s, w_k, s'\rangle, \langle s, w_0, s''\rangle, \langle s'', w_k, s'''\rangle, \langle s', w_0, s'''\rangle \in\rightarrow) = False.$$

We make a distinction between two cases, either $w_k \neq w_1$ or $w_k = w_1$. The first case satisfies the predicate, because after the execution of w_0, there does not exists a transition to a new state with a workflow step different from w_1, so $s \xrightarrow{w_0} s'' \xrightarrow{w_k} s'''$ does not exists.

Now, we need to prove that:

$$(\exists_{s',s'',s'''\in S} : \langle s, w_1, s'\rangle, \langle s, w_0, s''\rangle, \langle s'', w_1, s'''\rangle, \langle s', w_0, s'''\rangle \in\rightarrow) = False.$$

Definition 3.6.2 of the protocol rule automaton, states that after the execution of w_0 a different state is reached than after the execution of w_1. Therefore, there does not exist a state s''' with two incoming transitions with workflow step w_0 and w_1.

□

Lemma 4.9.3. *Given a case $CS = (HL, AW, Pr)$, the head state $s = head(CS)$, a patient space PS, which is based on the protocol $Pr = (W, \succ, \gg, \prec, \ll, Q)$, which includes the protocol rule $w_0 \gg w_1$. , and the set of feasible workflow steps AT. We define the set $AW_w = \{w \mid \langle st, w\rangle \in AW\}$. It holds that:*

$$\textit{if } w_0 \in AT \textit{ then } AW = \emptyset.$$

Proof. Suppose that $w_0 \in AT$ then there is a transition $\langle s, w_0, s'\rangle \in\rightarrow$ and there is a set AW_w of workflow steps, which satisfies the predicate $Check(AW_w \cup \{w_0\}, s)$. If AW_w is empty it satisfies the predicate $Check(\{w_0\}, s)$, see Definition 4.3.1.

We can determine that if $w_0 \in AW_w$ then $w_0 \notin AT$ by Definition 4.3.5. Now, we can reason about the subset $W\backslash\{w_0\}$. We define the set AW_w as a set with a single element, say $AW_w = \{w_k\}$, where $w_k \in W\backslash\{w_0\}$. The workflow step w_0 can only be an element of AT if it satisfy the predicate $Check(\{w_0, w_k\}, s)$. Therefore, we prove that $Check(\{w_0, w_k\}, s) = False$. Now, we can use partly the prove of Lemma 4.9.2, namely:

$$(\exists_{s',s'',s'''\in S} : \langle s, w_k, s'\rangle, \langle s, w_0, s''\rangle, \langle s'', w_k, s'''\rangle, \langle s', w_0, s'''\rangle \in\rightarrow \wedge Check(\{w_0\}, s')) \wedge$$

$$(\exists_{s',s'',s'''\in S} : \langle s, w_0, s'\rangle, \langle s, w_k, s''\rangle, \langle s'', w_0, s'''\rangle, \langle s', w_k, s'''\rangle \in\rightarrow \wedge Check(\{w_k\}, s')) = False$$

We weaken the predicate by substituting the second part by $True$ and in the first part we substitute the predicate $Check(\{w_0\}, s')$ by $True$, hence:

$$(\exists_{s',s'',s'''\in S} : \langle s, w_k, s'\rangle, \langle s, w_0, s''\rangle, \langle s'', w_k, s'''\rangle, \langle s', w_0, s'''\rangle \in\rightarrow) = False.$$

We make a distinction between two cases, either $w_k \neq w_1$ or $w_k = w_1$. The first case satisfies the predicate, because after the execution of w_0, there does not exists a transition to a new state with an outgoing transition with a workflow step different then w_1, so $s \xrightarrow{w_0} s'' \xrightarrow{w_k} s'''$ does not exists.

Now, we need to prove that:

$$(\exists_{s',s'',s''' \in S} : \langle s, w_1, s' \rangle, \langle s, w_0, s'' \rangle, \langle s'', w_1, s''' \rangle, \langle s', w_0, s''' \rangle \in \rightarrow) = False.$$

Definition 3.6.2 of the protocol rule automaton, stated that after the execution of w_0 a different state is reached then after the execution of w_1. Therefore, there is no state s''' with two incoming transitions with workflow step w_0 and w_1.

□

5 Practical medical treatment

This section describes the validation of the protocol semantics described in the previous sections. Therefore, an intended workflow, based on a practical example of a gynaecology protocol used in the field of brachytherapy, is compared with the workflow resulting from an implementation of the example in our formal protocol semantics.

The protocol consists of a main protocol, which makes use of two sub protocols, namely: Plan creation and Delivery. The workflows of the main protocol and the two sub protocols are depicted in three flow diagrams, which are shown in Figure 5.1.1, Figure 5.1.2 and Figure 5.1.3.

5.1 Implementation

The flow diagrams are transformed into the protocol semantics described in section 3.4. The protocol semantics is defined as follows:

$$Pr = (W, \succ, \gg, \prec, \ll, (QC, Q_i, Q_g)).$$

The protocols are constructed on the generic bases of the inter-task dependencies. The inter-task dependencies are categorized in two groups, the value and the external dependencies. The transitions depicted in the flow diagrams describe the external dependencies. The pre- and postconditions of the workflow steps describe the value dependencies.

The transitions in the flow diagrams are interpreted as follows: When no concurrency is possible, then the transition is interpreted as a direct rule. Otherwise the transition is interpreted as an eventually rule.

The pre- and postconditions and the abbreviations of the workflow steps are described in Appendix A.

Main protocol: Gynaecology

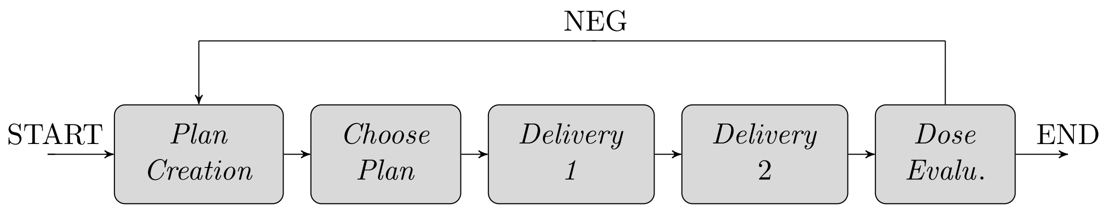

Figure 5.1.1: Main flow diagram of the protocol: GYN

The implementation of the main protocol is straightforward, because there is no concurrency possible in the flow diagram of Figure 5.1.1. Therefore, each transition is interpreted as a directly after rule. Except for the transitions between the workflow elements *Delivery 2* and *Dose Evaluation*. These transitions are not described, because the workflow element *Dose Evaluation* contains two workflow steps and the semantics of a direct rule do not support two workflow steps after a single workflow step.

In other words, the workflow element *Dose Evaluation* can have either a positive or negative result. Therefore, it consist of two workflow steps. After the execution of the workflow step $Branch_{Delivery_2}$, one of the two workflow steps of the *Dose Evaluation* needs to be executed. Thus, the execution of another workflow step beside these two workflow steps is not allowed. Therefore, a protocol rule is needed that states: "Directly after the execution of workflow step 'A', workflow step 'B' or workflow step 'C' needs to be executed."

The main protocol GYN is formalized as follows:

$$\begin{aligned} Pr_{GYN} = &(\{Branch_{Plancreation}, CP, Branch_{Delivery_1}, Branch_{Delivery_2}, DEP, DEN\}, \{\},\\ &\{\langle Branch_{Plancreation}, CP\rangle, \langle CP, Branch_{Delivery_1}\rangle\} \cup\\ &\{\langle Branch_{Delivery_1}, Branch_{Delivery_2}\rangle, \langle DEN, Branch_{Plancreation}\rangle\},\\ &\{\}, \{\}, \langle\{DoseReached\}, \{\}, \{DoseReached\}\rangle). \end{aligned}$$

Sub protocol: Plan Creation

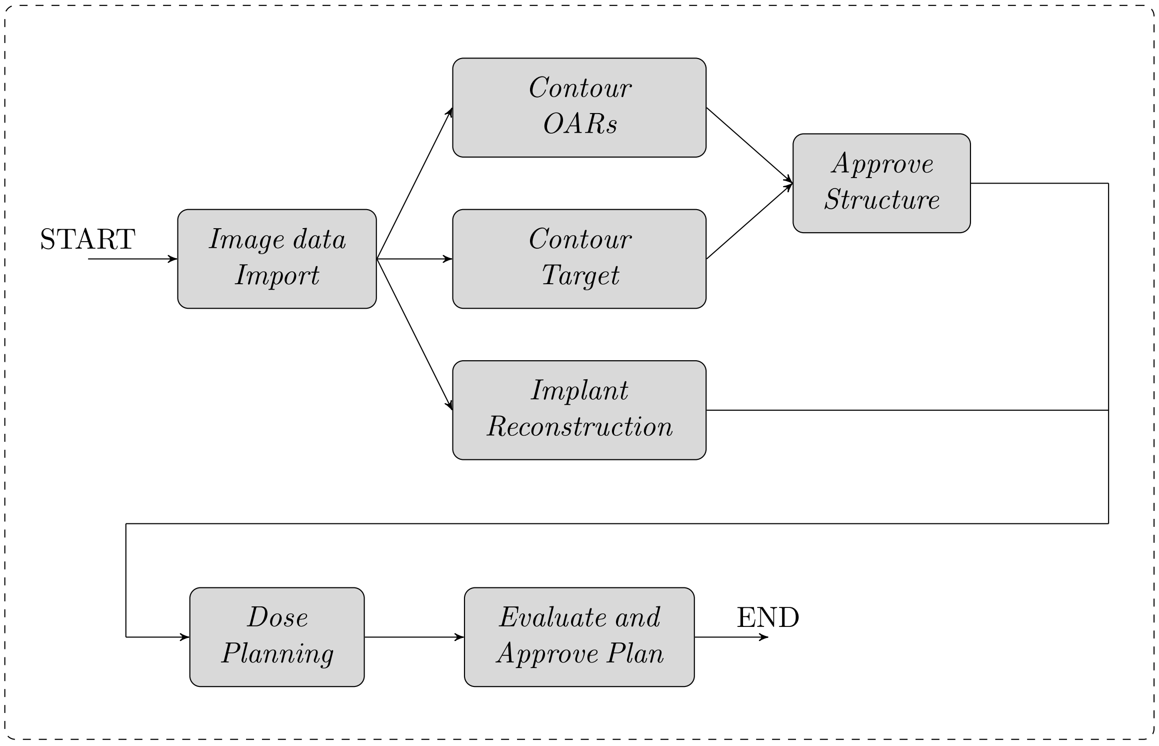

Figure 5.1.2: Flow diagram of the sub protocol: Plan Creation

The flow diagram of Plan Creation, shown in Figure 5.1.2, allows concurrency until the workflow step *Dose Planning*. Therefore, the transitions until this workflow element are interpreted as eventually after rules.

The transition between *Dose Planning* and *Evaluate and Approve Plan* must be interpreted as a directly after rule. However, this transition is not described, because the workflow element *Evaluate and Approve Plan* contains two workflow steps and that is not supported by the semantics, just as explained in the implementation of the main protocol.

The sub protocol Plan Creation is formalized as follows:

$$\begin{aligned} Pr_{PlanCreation} =& (\{IM, CT, CO, ASS, UAS, IR, DP, PUA, PA\}, \{\langle IM, CT\rangle, \langle IM, CO\rangle\} \cup \\ & \{\langle CT, ASS\rangle, \langle CO, ASS\rangle, \langle IR, DP\rangle, \langle UAS, ASS\rangle\}, \{\}, \\ & \{DP, ASS\}, \{\}, \langle\{ApprovedPlan\}, \{\}, \{ApprovedPlan\}\rangle). \end{aligned}$$

Sub protocol: Delivery

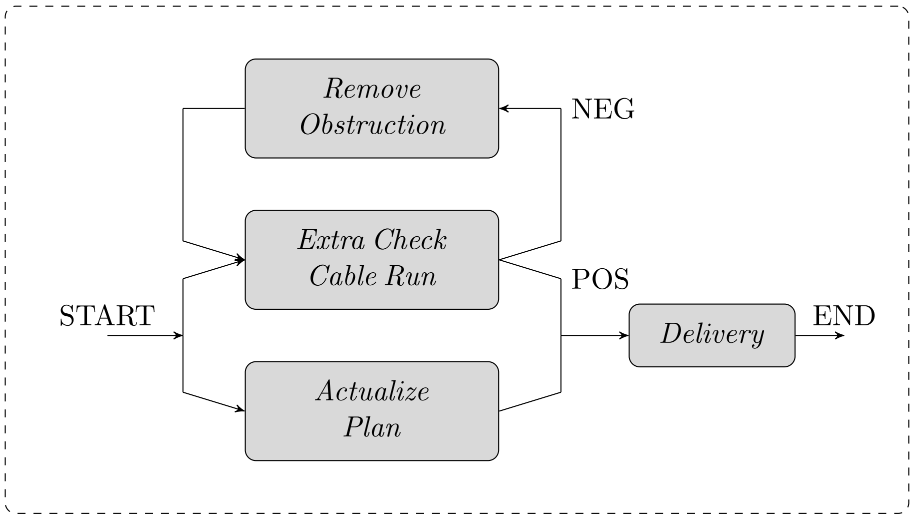

Figure 5.1.3: Flow diagram of the sub protocol: Delivery

The flow diagram of the protocol Delivery, shown in Figure 5.1.3, allows that workflow steps are executed concurrent, except for the workflow step *Delivery*.
However, not all the transitions of this flow diagram are interpreted as an eventually after rule, because the intention of this protocol differs in comparison with the two other protocols. This protocol needs the usage of the eventually before rules, because the intended workflow states that before the execution of a workflow step, another workflow step must have been executed before.

For example, before an obstruction in a catheter can be removed, it must be known which catheter is obstructed. Therefore, the workflow step *Check Cable Neg* must have been taken before the workflow element *Remove Obstruction* can be executed.

The sub protocol Delivery is formalized as follows:

$$\begin{aligned} Pr_{Delivery} =& (\{CCP, CCN, AP, RO, D\}, \{\langle RO, CCP\rangle, \langle CCN, RO\rangle\}, \{\}, \\ & \{\langle CCP, D\rangle, \langle CCN, RO\rangle, \langle AP, D\rangle\}, \{\}, \\ & \langle\{ApprovedPlan, Dose\}, \{ApprovedPlan\}, \{Dose\}\rangle). \end{aligned}$$

5.2 Evaluation

For each protocol appointed in section 5.1 a patient space is constructed. These patient spaces can be found in Appendix B, except for the sub protocol Plan Creation. The patient space is used to determine the feasible workflow steps during the treatment of a patient.

This section describes the intended behavior of the workflow of a gynaecology protocol, described by the flow diagrams. Secondly, the obtained behavior of the workflow of our protocols are described. The intended and obtained behaviors of the workflows are compared to determine the discrepancy. Each protocol will be evaluated separately.

Main protocol: Gynaecology

This protocol describes the overall workflow of the treatment of the patient. During the treatment the patient receives the desired dose, which is calculated during the plan creation.

(Intended behavior) Figure 5.1.1 depicts a flow diagram where concurrency in the workflow is excluded. The treatment starts with the creation of a plan and should terminate after a positive dose evaluation. When the result of the desired dose evaluation is negative, a new plan needs to be created and the same workflow will be executed. There should always be an evaluation step after two deliveries.

(Obtained behavior) The patient space of Figure B.0.1 in Appendix B depicts the obtained workflow, where concurrency is almost excluded. The treatment starts with creation of a plan. However, the protocol does not terminate after reaching the desired dose. Therefore, it is always possible to execute a following workflow step. This has as consequence that after reaching the desired dose, a new dose can be delivered to the patient. This is in practice undesirable. Moreover, it is possible to irradiate the patient more than twice without evaluating the dose at the patient, which is also undesirable.

(Discrepancies) The first discrepancy is the termination of the treatment. With our semantics it is not possible to define a rule to terminate a treatment. Therefore, the semantics must be extended with a rule that states that after the execution of a certain workflow step, some workflow steps are disabled.

The second discrepancy is the number of irradiations between the dose evaluations of the patient. This discrepancy occurs, because the transitions between the workflow elements *Delivery 2* and *Dose Evaluation* can not be described in the implemented protocol.

Sub protocol: Plan Creation

This protocol creates a plan based on the location of the applicator inside the patient.

(Intended behavior) The flow diagram, shown in Figure 5.1.2, describes a workflow where it is possible to execute the workflow steps concurrently until the workflow element *Dose Planning*. After this workflow step the plan can be approved and the workflow terminates. This sub protocol does not use workflow elements that can change the physical condition of the patient.

(Obtained behavior) The patient space of this sub protocol is not included into the Appendix, because the number of states and transitions were to large to create a clear picture.

It is not always possible to work concurrent in the workflow. However, the workflow elements *Dose Planning* and *Evaluation and Approve Plan* are able to work concurrent with other workflow steps.

Figure 5.2.1 shows a trace of the patient space in order to give some insight into the behavior of the obtained workflow. This trace shows the following invalid trace: An user approved a certain structure set, subsequently he creates a new contour and disapproves this new structure set, where after he continues the process and creates a dose plan on the unapproved structure set.

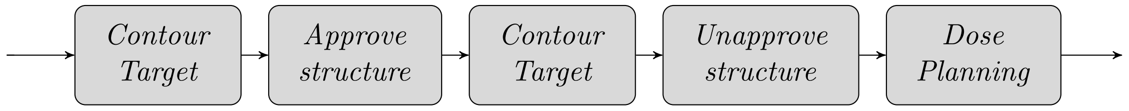

Figure 5.2.1: Single trace of sub protocol Plan Creation

(Discrepancies) The first discrepancy is the possible concurrency between the two behaviors. The intended workflow shows that no concurrency is possible from the start of the workflow element *Dose planning*. However, in the obtained workflow is concurrency still possible. During the implementation, we already observed that the transition between the workflow elements *Dose planning* and *Evaluate and Approve Plan* can not be transformed into the protocol semantics. If this was possible then it was not possible to work concurrent with the workflow element *Evaluate and Approve Plan*. However, it was still possible to work concurrent with the workflow element *Dose planning*. It is possible to extend the protocol semantics with a rule that disallows concurrency.

The invalid trace is also not clearly described in the intended workflow. However, it is not possible to avoid this trace with the protocol semantics described in this report. Although it is also for this problem possible to extend the protocol rules with a rule that disallows a workflow step between the occurrences of two other workflow steps.

Sub protocol: Delivery

This protocol describes the steps to deliver the dose to the patient based on the plan created in the sub protocol Plan Creation.

(Intended behavior) The flow diagram in Figure 5.1.3 depicts a workflow that should start with two workflow elements, which can be executed concurrently. The workflow should terminate after the execution of the workflow element *Delivery*.

Workflow element *Delivery* can only be activated if the result of the workflow element *Extra Cable Check Run* is positive and the workflow step *Actualize Plan* is executed. Otherwise, the obstruction must be removed and the workflow element *Extra Cable Check Run* needs to be executed again.

The workflow element *Delivery* should not be executable in parallel with any other workflow element.

(Obtained behavior) The patient space of Figure B.0.2 in Appendix B describes a workflow that starts with the three workflow steps *Actualizeplan*, *Check Cable Pos* and *Check Cable Neg*, which can be executed concurrently.

The workflow does not terminate. Hence, it is always possible to execute a new workflow step. Therefore, it is possible to executed the workflow step *Delivery* multiple times. The workflow step *Delivery* can only be activated if the workflow steps *Check Cable Pos* and *Actualize Plan* are executed. The workflow step *Delivery* can not work concurrently with other workflow steps.

(Discrepancies) The first discrepancy in the sub protocol Delivery is the possibility to radiate the patient multiple times without evaluating the total dose on the patient.

The evaluation of the total dose is done in the main protocol. Therefore, it must be impossible to execute the workflow step *Delivery* more than once in this sub protocol. Therefore, a solution is needed such as appointed in the explanation of the main protocol discrepancy.

The solution should extend the protocol semantics with a rule that disables some workflow steps after executing a workflow step. Hence, if the workflow step *Delivery* is disabled after executing once, then the workflow elements *Extra Check Cable Run*, *Actualize plan* and *Remove Obstruction* are obsolete. Therefore, all workflow steps can be disabled after executing the workflow step *Delivery*.

The last discrepancy occurs when the workflow step *Cable Check Neg* is directly followed by the workflow step *Delivery*, this means that there is an obstruction in one of the catheters during delivery.

This problem is also solved if the workflow directly terminates after the workflow step *Delivery*, because there must exists a path from every state to an accepting state. If the workflow step *Cable Check Neg* is executed it will reach a non accepting state. Therefore, if the treatment terminates after the workflow step *Delivery*, it is not longer possible to reach an accepting state and this path would not exist.

6 Conclusion

This section emphasis the contribution to the research of workflow management systems and it provides a summary of the semantics for the workflow management system described in this report. Moreover, it discusses the gap between theory and practice. This section ends with comments on possible future research.

6.1 Recapitulation

This report describes the formal semantics of a protocol based workflow management system. The semantics are based on automata theory and are supported by formal proofs and some examples. The workflow management system is used to control, execute and to monitor the workflow of a treatment.

A workflow is composed of small steps called workflow steps. These workflow steps make use of devices and algorithms to perform the treatment of the patient. A workflow step is described by pre and post conditions. The pre conditions represent the start requirements and the post conditions the results of a workflow step.

A workflow in the medical field is described by a protocol, which describes the constraints or inter-task dependencies between the workflow steps and the workflow steps that are needed to accomplish the medical intervention. Hence, the workflow management system is designed as a constraint based workflow management system, which means that every workflow step is allowed to perform, except for those workflow steps that violates the inter-task dependencies.

The inter-task dependencies are separated in two categories, namely the value and external dependencies. The value dependencies are based on the pre conditions of a workflow step and the post conditions generated by certain workflow steps. The external dependencies describe the preferred user ordering of the workflow steps. They are described by the protocol rules.

All conceivable workflows of a single protocol are described in an automaton, called the patient space. The base of this automaton is the workflow space, which is constructed by the value dependencies of the workflow steps described in the protocol. This workflow space will be restricted by the external dependencies with use of the protocol rule automata.
A synchronous product automaton can be constructed with use of the workflow space and all the protocol rule automata. After removing the infeasible workflow steps from this automaton the patient space emerged.

In the workflow management system a workflow controller interprets the patient space. The workflow controller determines the feasible workflow steps based on the achieved results of the treatment, the protocol, and the currently active workflow steps. These parts are described in a case. The treatment of a patient can contain multiple cases, which are combined in a case space.
The workflow controller supports a branching mechanism in order to allow parallel work and research purposes.

6.2 Gap between theory and practice

This report describes a medical example of a gynaecology protocol to validate the formal protocol semantics.

From the discrepancies between the intended and obtained workflows, it can be concluded that the protocol semantics are not yet expressive enough in order to achieve the desired workflows. However, the protocol semantics can be extended to cover all the demands of the practical example.

Although the protocol semantics can be extended, there is still a gap between theory and practice. This report provides the notion of workflow elements as a set of workflow steps that are related to each other. However, there is a problem in the definition of what a workflow element is in relation to workflow steps.

For example, the workflow element *Dose Evaluation* contains two workflow steps, namely: *Dose Evaluation Positive* and *Dose Evaluation Negative*. The report assumed that the user can activate one of these two workflow steps. Although in practice the user activates the workflow element *Dose Evaluation* and not one of the two workflow steps. If the result of the workflow element is achieved, then one of the two workflow steps will be taken. Therefore, the assumption that one of the two workflow steps can be activated is incorrect.

Another example of a shortcoming in the definition of a workflow element is the opportunity to include side effects into workflow elements. For some workflow elements it is possible to predict the side effects. For instance the side effects of a dose delivery when it is aborted partially. If these side effects can be modeled into the workflow element as a workflow step, then it is possible to extend the protocol with a workflow that resolves the result of such side effects.

However, in the current semantics is this not possible, because if the side effect is modeled as a workflow step, then it will appear in the set of feasible workflow steps. Hence, the user is able to activate a workflow step which is not desirable. Therefore, we would be able to activate the workflow element including the workflow steps, which models the side effects and if the workflow element terminates, then the related workflow step can be taken.

6.3 Future research

First of all, the gap mentioned above between theory and practice must be resolved.

Secondly, the authorization of executive medical staff must be included to the protocol. The treatment of a patient is carried out through qualified medical staff with different authorizations, and tasks in the workflow. The semantics of the workflow management system does currently not support the notion of authorization.

Finally, the semantics of the workflow controller supports a minor feature to handle unexpected workflow steps. This feature can be improved, because our workflow management system is a constraint based workflow, which means that every workflow step that does not violate constraints is allowed to perform. Therefore, it is possible to determine the reason behind the unexpected workflow step. The unexpected workflow step violates either the protocol rules, or it was not included into the protocol. If the reason of the occurrence of the unexpected workflow step is known, then the protocol can be modified in a manner that the workflow step is allowed once. The benefit of this handling mechanisme is that the workflow can continue.

A Workflow Elements

The tables in this appendix describe the workflow elements and workflow steps used in the practical example.

Actualize Plan

Name workflow step	Precondition	Postcondition
Actualize plan (AP)	Approved Plan	Actualized Plan

Approve Structure

Name workflow step	Precondition	Postcondition
Approve structure set (ASS)	Structure set Target Structure Set OARs	Approved set
Unapprove structure set (UAS)	Structure set Target Structure Set OARs	Unapproved set

Choose Plan

Name workflow step	Precondition	Postcondition
Choose Plan (CP)	Plan	Plan

Contour Target

Name workflow step	Precondition	Postcondition
Contour Target (CT)	3D Image set	Structure set Target

Contour OARs

Name workflow step	Precondition	Postcondition
Contour OARs (CO)	3D Image set	Structure set OARs

Delivery

Name workflow step	Precondition	Postcondition
Delivery (D)	Actualized Plan	Dose

Delivery 1

Name workflow step	Precondition	Postcondition
Branch Delivery 1 (BD1)	Approved Plan	Dose1

Delivery 2

Name workflow step	Precondition	Postcondition
Branch Delivery 2 (BD2)	Approved Plan	Dose2

Dose Evaluation

Name workflow step	Precondition	Postcondition
Dose Evaluation Pos (DEP)	Dose	DoseReached
Dose Evaluation Neg (DEN)	Dose	DoseNotReached

Dose Planning

Name workflow step	Precondition	Postcondition
Dose Planning (DP)	3D Image Set Structure Set OARs Structure Set Target Reconstructed Applicator	Plan

Evaluate and Approve Plan

Name workflow step	Precondition	Postcondition
Plan Approval (PA)	Plan	Approved Plan
Plan Un-Approval (PUA)	Plan	Un-Approved Plan

Extra Check Cable Run

Name workflow step	Precondition	Postcondition
Check Cable Pos (CCP)	Approved Plan	Pos Validation Cable
Check Cable Neg (CCN)	Approved Plan	Neg Validation Cable

Image Data Import

Name workflow step	Precondition	Postcondition
Image Import (IM)	None	3D image

Implant Reconstruction

Name workflow step	Precondition	Postcondition
Implant Reconstruction (IR)	3D Image Set	Reconstructed Applicator

Plan Creation

Name workflow step	Precondition	Postcondition
Branch Plan Creation (BPC)	None	Approved Plan

Remove Obstruction

Name workflow step	Precondition	Postcondition
Remove Obstruction (RO)	None	None

B Medical protocol implementation

The figures in this appendix describe the patient spaces, which are determined on the bases of the practical example.

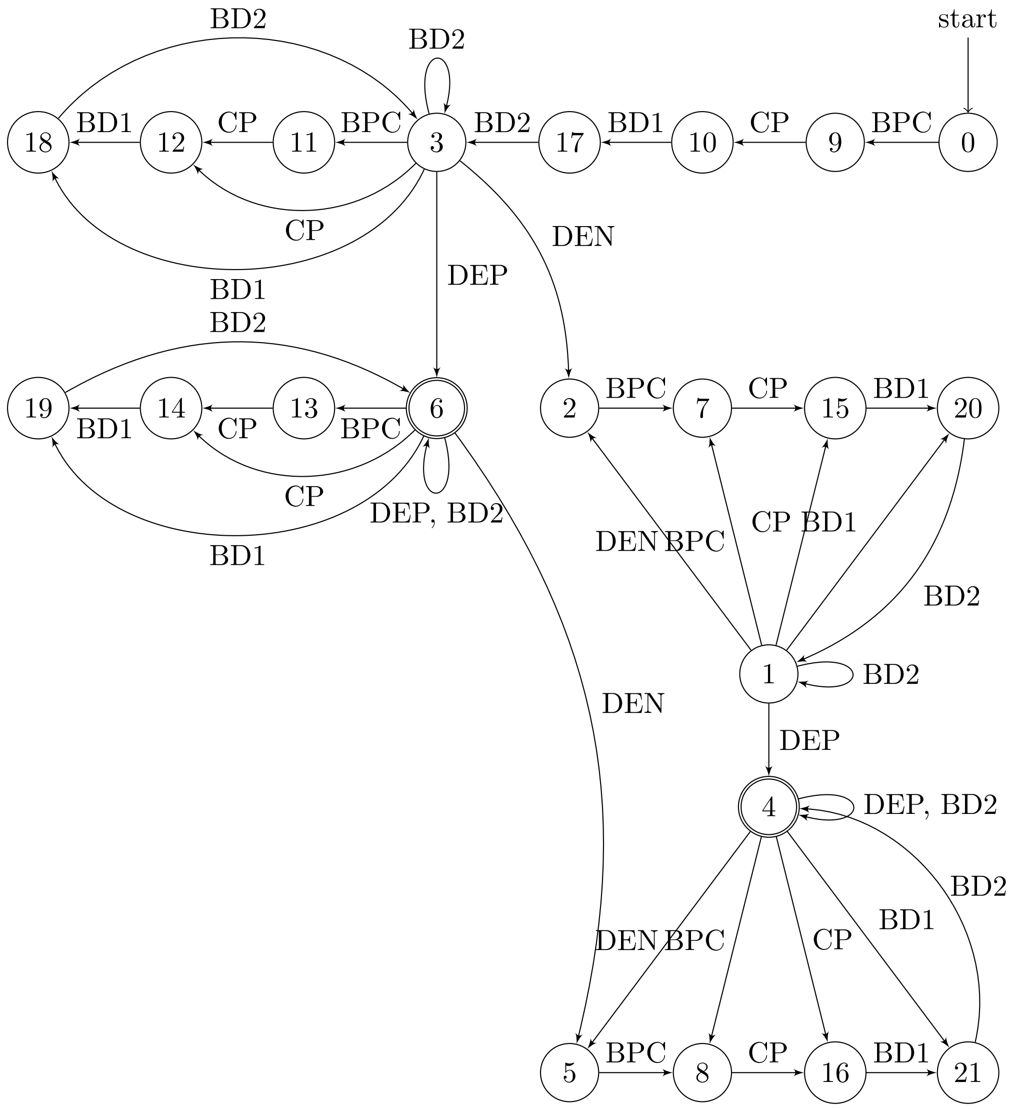

Figure B.0.1: Patient space of the main protocol: GYN

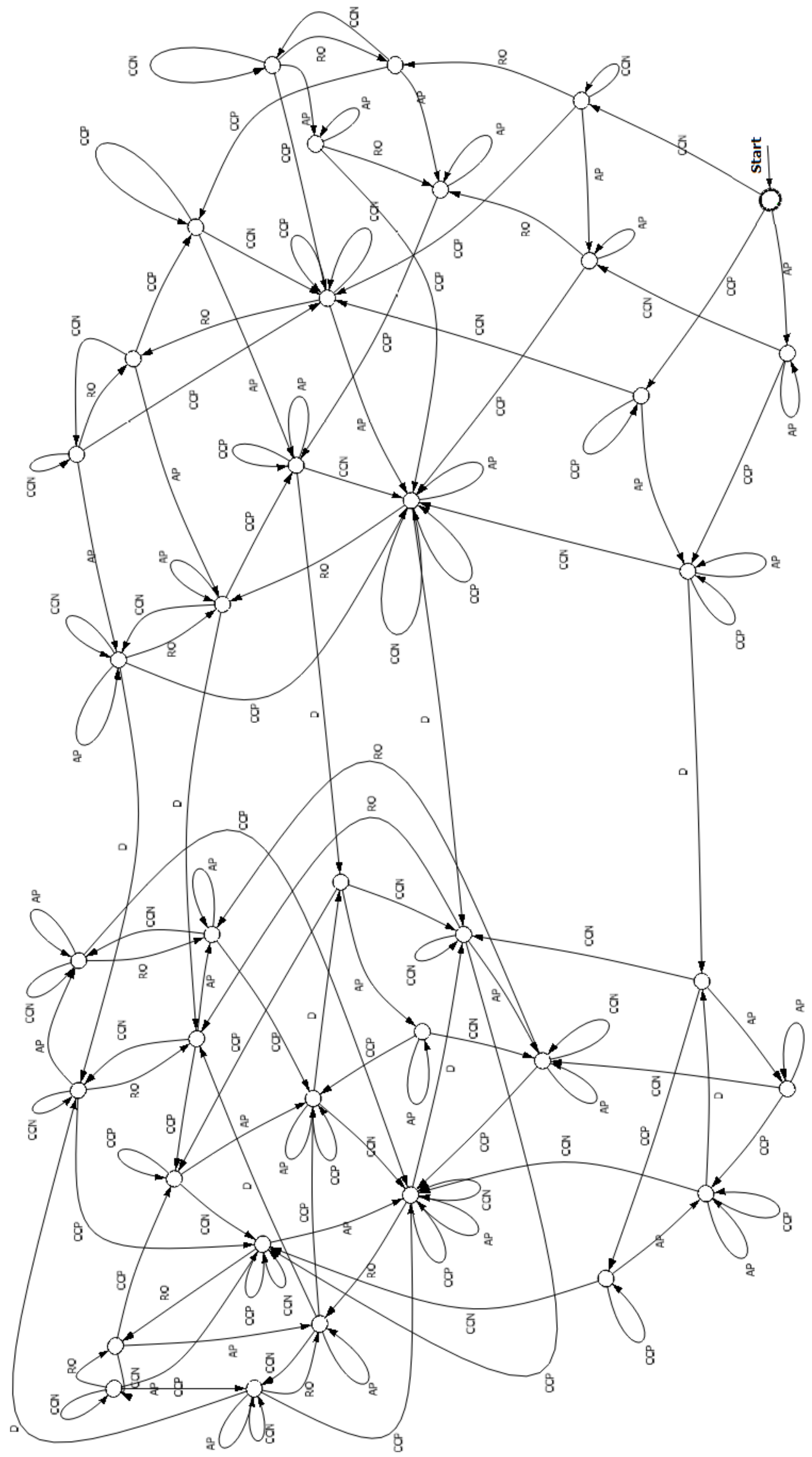

Figure B.0.2: Patient space of the sub protocol: Delivery

References

[1] J.F. Groote and M.R. Mousavi, *Modelling and Analysis of Communicating Systems,* MIT Press, 2014.

[2] R. Milner. *A Calculus of Communicating Systems.* LNCS 92, Springer-Verlag, 1980.

[3] W.M.P. van der Aalst, *The Application of Petri Nets to Workflow Management,* Journal of circuits, systems, and computers, World Scientific, 1998.

[4] M. Adams, *Yawl - User Manual,* YAWL Foundation, Yawlfoundation.org, version 2.3, 2004-2012

[5] W.M.P. van der Aalst, A.H.M. ter Hofstede, and M. Weske, *Business Process Management: A Survey,* International Conference, BPM 2003 Eindhoven, The Netherlands, June 26-27, Proceedings 2003.

[6] W.M.P. van der Aalst, *Modeling and Analyzing Interorganizational Workflows,* International Conference, Application of Concurrency to System Design, Proceedings 1998.

[7] R. Hamadi and B. Benatallah, *A Petri net-based model for web service composition,* Proceedings of the 14th Australasian database conference - Volume 17, pages 191-200, 2003.

[8] W.M.P van der Aalst, *Three Good Reasons for Using a Petri-net-based Workflow Management System,* The Kluwer International Series in Engineering and Computer Science: Information and Process Integration in Enterprises: Rethinking Documents, Chapter 10, 1998.

[9] W. Reisig, *Petri Nets: An Introduction* EATCS Monographs on Theoretical Computer Science, vol. 4Springer, Berlin, Germany, 1985.

[10] Whitepaper AAMI, *Medical device interoperability,* Association for the Advancement of Medical Instrumentation, 2012.

[11] A.R. Khemuka, *Workflow modeling using finite automata,* Graduate Thesis and Dissertations, University of South Florida, 2003.

[12] C. Ouyang, E. Verbeek, W.M.P. van der Aalst, S. Breutel, M. Dumas, and A.H.M ter Hofstede, *Formal semantics and analysis of control flow in WS-BPEL* Science of Computer Programming 67, p.162-198, 2007.

[13] M. Pesic, M.H. Schonenberg, N. Sidorova, and W.M.P. van der Aalst, *Constraint-Based Workflow Models: Change Made Easy* On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS. Lecture Notes in Computer Science Volume 4803, pp 77-94, 2007.

[14] Patient Guide, *Brachytherapy: The precise answer for tackling cancer*, Source: http://www.aboutbrachytherapy.com/en-us/patients/resources/Documents/ Brachytherapy -patient-guide.pdf, version 888.00166 MKT [00].