An Argumentative Approach to Assessing Safety in Medical Device Software Using Defeasible Logic Programming

Article *in* IFMBE proceedings · January 2014

DOI: 10.1007/978-3-319-07653-9_34

CITATIONS
2

READS
100

3 authors, including:

Adrian Groza
Universitatea Tehnica Cluj-Napoca
159 PUBLICATIONS **352** CITATIONS
SEE PROFILE

Carlos Chesñevar
Universidad Nacional del Sur
125 PUBLICATIONS **2,445** CITATIONS
SEE PROFILE

An Argumentative Approach to Assessing Safety in Medical Device Software Using Defeasible Logic Programming

S.A. Gómez[1], A. Groza[2], and C.I. Chesñevar[1]

[1] Artificial Intelligence Research and Development Laboratory (LIDIA)
Department of Computer Science and Engineering, Universidad Nacional del Sur
Av. Alem 1253, (8000) Bahía Blanca, Argentina
{sag,cic}@cs.uns.edu.ar
[2] Intelligent Systems Group (ISG)
Department of Computer Science, Technical University of Cluj-Napoca
Baritiu 28, 400391, Cluj-Napoca, Romania
Adrian.Groza@cs.utcluj.ro

Abstract— **Modern health-care technology depends to a large extent on software deployed in medical devices, which brings several well-known benefits but also poses new hazards to patient safety. As a consequence, assessing safety and reliability in software in medical devices turns out to be a critical issue. In this paper we outline a method for safety assessment of medical devices based on Defeasible Logic Programming (DeLP), which provides an argumentative framework for reasoning with uncertain and incomplete knowledge. We contend that argumentation theory as defined in DeLP can be used to integrate and contrast different evidences for assessing the approval and commercialization of medical devices, aiming at increasing transparency to all the stakeholders involved in their certification. The outlined framework is validated by modeling the infamous Therac-25 accident.**

Keywords— **medical software, safety assurance, argumentation theory**

I. Introduction

Modern health-care technology depends nowadays to a large extent on software deployed in medical devices, which brings several well-known benefits but also poses new hazards to patient safety. Computer-related errors have played a significant role in serious injury or even death involving medical devices [1]. With more complex software employed in such devices [2], there is no possibility to perform exhaustive testing or formal methods. As a consequence, assessing safety and reliability in software in medical devices turns out to be a critical issue. The manufacturers of medical devices are expected not only to achieve an acceptable assurance level, but also to convince regulatory bodies that safeness has been achieved. In order to do this, identifying relevant evidence is not enough, as there might be conflicting or incomplete information involved. Consequently, structured, domain-based, safety *arguments* are needed to demonstrate that safety can be ensured.

In this paper we outline a method for safety assessment of medical devices based on Defeasible Logic Programming (DeLP), which provides an argumentative framework for reasoning with uncertain and incomplete knowledge. We contend that argumentation theory as defined in DeLP can be used to integrate and contrast different evidences for assessing the approval and commercialization of medical devices, aiming at increasing transparency to all the stakeholders involved in their certification. Our approach in this paper will aim at two particular issues: (i) how defeasible logic programming can be used to describe safety requirements of medical devices, and (ii) how argumentation theory can detect and solve inconsistencies or safety related doubts.

Our hypothesis is that argumentation theory (in this paper implemented via DeLP) can contribute to improve safety assurance in medical devices, providing a promising research topic (see e.g. [3, 4]). Safety arguments[1] should address at least two issues: (i) to show that all health-related hazards have been analyzed and how the assigned control measures contribute to risk mitigation, and (ii) to prove compliance with safety medical standards or guidelines.

It must be remarked that over the past few years there have been several formal complaints and court trials concerning the approval of medical devices [6], and several studies have shown that the regulatory system is opaque, fragmented and largely privatized [7]. We contend that argumentation theory as defined in DeLP can be used to integrate and contrast

[1] A safety case is defined in UK Defense Standard 00-56 as [5]: *"A structured argument, supported by a body of evidence that provides a compelling, comprehensible and valid case that a system is safe for a given application in a given operating environment." In our case, a safety case should present strong arguments that a medical device is safe to operate within a particular context [3].*

S. Vlad and R.V. Ciupa (eds.), *International Conference on Advancements of Medicine and Health Care through Technology*,
IFMBE Proceedings 44,
DOI: 10.1007/978-3-319-07653-9_34,

different evidences for assessing the approval and commercialization of medical devices, aiming at increasing transparency to all the stakeholders involved in their certification.

The rest of the paper is structured as follows: Section II presents the fundamentals of argumentation and DeLP, along with a simple motivational example. Section III explains and models the Therac-25 overdose radiation accident using DeLP. We illustrate how the arguments associated with this particular accident can be automatically obtained from the DeLP inference engine. Section IV reviews related work. Finally, in Section V we summarize the main contributions of our proposal.

II. Fundamentals of defeasible logic programming

Defeasible Logic Programming (DeLP) [8] provides a language for knowledge representation and reasoning that uses *defeasible argumentation* [9, 10] to decide between contradictory conclusions through a *dialectical analysis*. Codifying knowledge by means of a DeLP program provides a good trade-off between expressiveness and implementability for dealing with incomplete and potentially contradictory information. In a DeLP program $\mathscr{P} = (\Pi, \Delta)$, a set Π of strict rules $P \leftarrow Q_1, \ldots, Q_n$, and a set Δ of defeasible rules $P \prec Q_1, \ldots, Q_n$ can be distinguished. Defeasible rules account for tentative knowledge, and provide the basis for defining *arguments*.[2]

Definition 1 (Argument) *An* argument $\langle \mathscr{A}, H \rangle$ *is a minimal non-contradictory set of ground defeasible clauses* $\mathscr{A}$ *of* Δ *that allows to derive a ground literal* H *using rules from* $\Pi \cup \mathscr{A}$.

Since arguments may be in conflict (concept captured in terms of a logical contradiction), an attack relationship between arguments can be defined. A criterion is usually defined to decide between two conflicting arguments. If the attacking argument is strictly preferred over the attacked one, then it is called a *proper defeater*. If no comparison is possible, or both arguments are equi-preferred, the attacking argument is called a *blocking defeater*. Generalized specificity [11] is typically used as a syntax-based criterion among conflicting arguments. However, other alternative partial orders can also be valid such (see [8, Sect. 3.2.2]). In this paper we will abstract the comparison criterion and will suppose fthe existence of a relation $\succ$ for comparing two conflicting arguments.

In order to determine whether a given argument $\mathscr{A}$ is ultimately undefeated (or *warranted*), a dialectical process is recursively carried out, where defeaters for $\mathscr{A}$, defeaters for these defeaters, and so on, are taken into account forming a *dialectical tree*. The argument $\mathscr{A}$ is the root of its dialectical tree $\mathscr{T}$, its defeaters the first level of $\mathscr{T}$, the defeaters of these defeaters the second level, and so on. In a dialectical tree nodes are labeled as either defeated (D) or undefeated (U). Leaves are always labeled as undefeated; a node is labeled as undefeated iff all of its children are labeled as defeated, otherwise a node is labeled as defeated. Given a DeLP program $\mathscr{P}$ and a query H, the final answer to H w.r.t. $\mathscr{P}$ takes such dialectical analysis into account. The answer to a query can be: *yes* (when there exists a warranted argument $\langle \mathscr{A}, H \rangle$), *no* (when there exists a warranted argument $\langle \mathscr{A}, \sim H \rangle$), *undecided* (neither $\langle \mathscr{A}, H \rangle$ nor $\langle \mathscr{A}, \sim H \rangle$ are warranted), or *unknown* (H does not belong to $\mathscr{P}$).

In [12], Williams & Hunter define an Ontology Argumentation Framework which combines ontologies and Defeasible Logic Programming in order to reason on a medical scenario regarding the treatment of cancer patients. Next we present a simplified version of that scenario in order to illustrate how DeLP can be used to generate arguments useful for clinical practice.

Example 1 *Consider the DeLP program* $\mathscr{P}_1 = (\Pi_1, \Delta_1)$ *presented in Fig. 1. The set of strict rules* Π_1 *says that there are two treatments for cancer: based on Anastrazole and a program of five years administration of Tamoxifen, Ms. Jones has a disease named* d_1 *which is both a ER positive disease and can be identified as breast cancer. A person who is a man or a woman is among people, early breast cancer is a type of cancer and a treatment with Tamoxifen can be performed either by a three year or a five year program. The set* Δ_1 *of defeasible rules is a very simplified version of what was presented by [12]. The rules are ordered according to the priorities* $r_2 \succ r_1 \succ r_2$ *(meaning that an argument based on* r_2 *will be considered stronger that one based on* r_1 *and so forth). From* $\mathscr{P}_1$ *we can build the following arguments:* $\langle \{r_1\}, has\_treatment(ms\_jones, t) \rangle$, $\langle \{r_2\}, \sim has\_treatment(ms\_jones, t) \rangle$ *and* $\langle \{r_3\}, \sim has\_treatment(ms\_jones, a) \rangle$. *Comparing arguments by using a criterion based on rule priorities, we can see that* $\langle \{r_1\}, has\_treatment(ms\_jones, t) \rangle$ *is defeated by* $\langle \{r_2\}, \sim has\_treatment(ms\_jones, t) \rangle$, *which in turn is defeated by* $\langle \{r_3\}, \sim has\_treatment(ms\_jones, a) \rangle$. *The dialectical trees from program* $\mathscr{P}_1$ *can be seen in Fig. 2*

[2] For space reasons, we provide only some basic DeLP definitions in this paper to make it self-contained. For a detailed explanation of DeLP, the reader is referred to [8].

DeLP program $\mathscr{P}_1 = (\Pi_1, \Delta_1)$**:**

Facts and strict rules Π_1**:**

$anastrazole(a)$. $tamoxifen5Yr(t)$.
$woman(ms\_jones)$. $has\_disease(ms\_jones, d)$.
$er\_positive\_disease(d)$. $early\_breast\_cancer(d)$.
$people(X) \leftarrow man(X)$. $people(X) \leftarrow woman(X)$.
$breast\_cancer(X) \leftarrow early\_breast\_cancer(X)$.
$tamoxifen(X) \leftarrow tamoxifen3Yr(X)$.
$tamoxifen(X) \leftarrow tamoxifen5Yr(X)$.

Defeasible rules Δ_1 **with priorities** $r_2 \succ r_1 \succ r_3$**:**

$r_1 : has\_treatment(X,Z) \prec$
$woman(X), has\_disease(X,Y), early\_breast\_cancer(Y),$
$er\_positive\_disease(Y), tamoxifen5Yr(Z)$.
$r_2 : \sim has\_treatment(X,Z) \prec$
$woman(X), has\_disease(X,Y),$
$breast\_cancer(Y), tamoxifen(Z)$.
$r_3 : \sim has\_treatment(X,Z) \prec$
$woman(X), has\_disease(X,Y), breast\_cancer(Y), anastrazole(Z)$.

Fig. 1 DeLP program $\mathscr{P}_1$ on cancer treatment medical trials

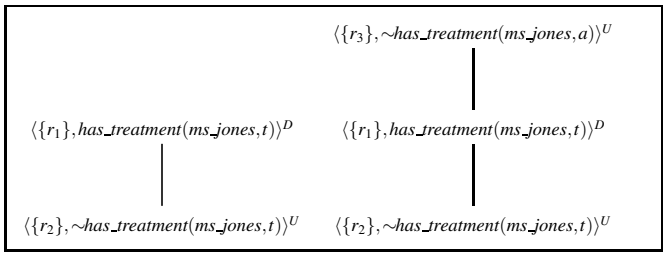

Fig. 2 Dialectical trees from $\mathscr{P}_1$

III. Arguing in the Therac-25 Accident

We now present a case study modeling the infamous Threrac-25 accelerator incident in which five people died of radiation overdose due to a malfunction during their cancer treatment sessions [13].

We discuss a DeLP program $\mathscr{P}$ for defining a safety verification system for such a system, we base our modeling in the extensive documentation regarding the Therac-25. Despite knowing that it is a somehow old case (it dates from 1987), we chose to show how DeLP can be used to perform a safety verification on a such a system as the Therac-25 accelerator problem can be considered a benchmark problem due to the extensive analyses it went through.

The main system safety verification is determining if using the accelerator is safe.

Definition 2 (Safety verification system) *A* safety verification system $\mathscr{V}$ *is a pair* $(\mathscr{P}, \mathscr{S})$ *where* $\mathscr{P}$ *is a DeLP program establishing logical criteria for assuring safety and* $\mathscr{S}$ *is a set of literals containing sensor information of the environment. The language* $\mathscr{L}_\mathscr{V}$ *of the safety verification system is the set of all literals in* $\mathscr{P} \cup \mathscr{S}$.

Definition 3 (Prospective safety decision) *Let* $\mathscr{V} = (\mathscr{P}, \mathscr{S})$ *be a safety verification system. A* prospective safety decision *is a literal in* $\mathscr{L}_\mathscr{V}$.

Definition 4 (Safety recommendation) *Let* $\mathscr{V} = (\mathscr{P}, \mathscr{S})$ *be a safety verification system and* $\mathscr{D}$ *be a prospective safety decision. A safety recommendation for* $\mathscr{D}$ *is either one of:*

- *Conform: If there is a warranting argument for* $\mathscr{D}$ *w.r.t. the DeLP program* $\mathscr{P} \cup \mathscr{S}$.
- *Do not Conform: If there is a warranting argument for* $\sim\mathscr{D}$ *w.r.t. the DeLP program* $\mathscr{P} \cup \mathscr{S}$.
- *Unable to Decide: There is neither a warranted argument for* $\mathscr{D}$ *nor* $\sim\mathscr{D}$ *w.r.t. the DeLP program* $\mathscr{P} \cup \mathscr{S}$.
- *Not Applicable:* $\mathscr{D}$ *does not belong to* $\mathscr{L}_\mathscr{V}$.

We will consider the modeling of a safety recommendation system $\mathscr{V} = (\mathscr{P}, safe(therac\_25))$, where the literal $safe(A)$ refers to whenever it is safe to use the accelerator A.

The rest of the section discusses the program $\mathscr{P}$ along with the dialectical analysis that can be derived from it w.r.t. the query $safe(therac\_25)$ in order to reach a safety recommendation for $\mathscr{V}$. We consider three accelerator models built by the AECL company named Therac-6, Therac-20 and Therac-25, represented by three facts: $accelerator(therac\_6)$, $accelerator(therac\_20)$, and $accelerator(therac\_25)$.

There were no reported cases of radiation overdose malfunction neither in Therac-6 nor in Therac-20 accelerator and an accelerator A is normally considered safe if it has a radiation overdose history with no reported cases. This is modeled as DeLP defeasible rules and facts as shown in Fig. 3.

$radiation\_overdose\_misfunction\_reported\_cases(therac\_6, 0)$.
$radiation\_overdose\_misfunction\_reported\_cases(therac\_20, 0)$.
$safe(A) \prec accelerator(A), \sim problems\_history(A)$.
$\sim problems\_history(A) \prec$
$radiation\_overdose\_misfunction\_reported\_cases(A, 0)$.

Fig. 3 Overdose history in Therac-25

At this point we can deduce an argument for the Therac-6 accelerator supporting the tentative conclusion that it is safe as no cases of radiation overdose have been reported $\langle \mathscr{A}_1, safe(therac\_6) \rangle$ where

$$\mathscr{A}_1 = \left\{ \begin{array}{l} (safe(therac\_6) \prec accelerator(therac\_6), \\ \quad \sim problems\_history(therac\_6)), \\ (\sim problems\_history(therac\_6) \prec \\ \quad radiation\_overdose\_misfunction\_reported\_cases(\\ \quad\quad therac\_6, 0)). \end{array} \right\}.$$

It is straightforward to see that the same is valid for the Therac-20 accelerator.

The initial assumption with the case of the Therac-25 accelerator was that an accelerator is usually considered safe whenever its software is based on a library written for a previous accelerator that is known to be problem-free, the software in each accelerator was tested in each respective device, and the Therac-6, Therac-20, Therac-25 were produced in chronological order, and it is also known that the software running on Therac-25 accelerator was based on Therac-6 and Therac-20 software packages (see Fig. 4). So, from all this information we are now able to build an argument $\langle \mathscr{A}_2, safe(therac\_25) \rangle$ where

$$\mathscr{A}_2 = \left\{ \begin{array}{l} (safe(therac\_25) \prec accelerator(therac\_25), \\ \quad software\_inside(therac\_25, therac\_25\_software), \\ \quad software\_based\_on(therac\_25\_software, \\ \qquad therac\_6\_package), \\ \quad software\_module(therac\_6\_package), \\ \quad tested(therac\_6\_package, therac\_6), \\ \quad accelerator(therac\_6), older\_model(therac\_6, therac\_25), \\ \quad \sim problems\_history(therac\_6)), \\ (older\_model(therac\_6, therac\_25) \prec \\ \quad next\_model(therac\_20, therac\_25), \\ \quad older\_model(therac\_6, therac\_20)), \\ (older\_model(therac\_6, therac\_20) \prec \\ \quad next\_model(therac\_6, therac\_20)), \\ (\sim problems\_history(therac\_6) \prec \\ \quad radiation\_overdose\_misfunction\_reported\_cases(\\ \qquad therac\_6, 0)) \end{array} \right\}$$

supporting the tentative conclusion that the Therac-25 accelerator is a safe device because its software was based on a Therac-6 module, which in turn was a problem free accelerator.

$safe(A) \prec accelerator(A), software\_inside(A,P),$
$\quad software\_based\_on(P,M),$
$\quad software\_module(M), tested(M,OA),$
$\quad accelerator(OA),$
$\quad older\_model(OA,A), \sim problems\_history(OA).$
$software\_module(therac\_6\_package).$
$software\_module(therac\_20\_package).$
$tested(therac\_6\_package, therac\_6).$
$tested(therac\_20\_package, therac\_20).$
$next\_model(therac\_6, therac\_20).$
$next\_model(therac\_20, therac\_25).$
$older\_model(X,Y) \prec next\_model(X,Y).$
$older\_model(X,Y) \prec next\_model(N,Y), older\_model(X,N).$
$software\_inside(therac\_25, therac\_25\_software).$
$software\_based\_on(therac\_25\_software, therac\_6\_package).$
$software\_based\_on(therac\_25\_software, therac\_20\_package).$

Fig. 4 Software modules in Therac-25

Another fact that came up during the Therac-25 incident was that the system had been programmed and tested by the same programmer and that it was not possible to determine if that programmer had had formal training in computer science. According to [13] the programmer could not be identified because at the time of the hearings in court he had left the AECL company who built the accelerators. So we will refer to him as John Doe (see Fig 5). In this regard, an argument $\langle \mathscr{A}_3, \sim safe(therac\_25) \rangle$ where

$$\mathscr{A}_3 = \left\{ \begin{array}{l} \sim safe(therac\_25) \prec accelerator(therac\_25), \\ \quad software\_inside(therac\_25, therac\_25\_software), \\ \quad developed\_by(therac\_25\_software, john\_doe), \\ \quad tested\_by(therac\_25\_software, john\_doe), \\ \quad programmer(john\_doe), \\ \quad no\_background\_information\_on\_programmer(john\_doe). \end{array} \right\}$$

(defeating argument $\mathscr{A}_2$) against the safety of the Therac-25 accelerator can be built.

$\sim safe(A) \prec accelerator(A), software\_inside(A,S),$
$\quad developed\_by(S,P), tested\_by(S,P),$
$\quad programmer(P),$
$\quad no\_background\_information\_on\_programmer(P).$
$developed\_by(therac\_25\_software, john\_doe).$
$tested\_by(therac\_25\_software, john\_doe).$
$programmer(john\_doe).$
$no\_background\_information\_on\_programmer(john\_doe).$

Fig. 5 Rules describing that there was no background on the programmer

Other criteria suggested that an accelerator is safely tested if its software module has gone through extensive unit, integration and simulator testing; otherwise, it cannot be considered safe. In the case of the Therac-25 software, it was used about 2700 hours without incidents and audits determined that the software in the Therac-25 went through an extensive integration testing but the unit testing was not that appropiate. No extensive simulator testing was performed either (see Fig. 6). In this case, it is easy to see that another argument in favor of the safety of the system based on the amount of hours of usage without incidents can be found but another one against the safety can be built on the grounds that it was proved that no extensive unit testing was performed on the Therac-25 software.

Another criteria for accelerator safety pointed out by [13] is that the Therac-20 accelerator included hardware safety interlocks as an additional safety mechanism to stop the machine in case of a malfunction, thus making Therac-20 a safe device. Those hardware safety interlocks were taken out in Therac-25 for cost reasons, thus making it unsafe (see Fig. 7).

A software module is unsafe if it runs on an unsafe operating system and an unsafe operating system is an operating system which is not safe. From [13], we know that the Therac-25 main bug was based on having a program running in a PDP-11 real time system that allowed concurrent access

$safe(A) \prec$
$\quad accelerator(A),$
$\quad software\_inside(A,S), safe\_software\_testing(S).$
$safe\_software\_testing(S) \prec$
$\quad used\_hours\_without\_problems(S,H),$
$\quad H > 2000.$
$safe\_software\_testing(S) \prec$
$\quad software\_module(S),$
$\quad extensive\_unit\_testing(S),$
$\quad extensive\_integration\_testing(S),$
$\quad extensive\_simulator\_testing(S).$
$\sim safe\_software\_testing(S) \prec$
$\quad \sim extensive\_unit\_testing(S).$
$\sim safe\_software\_testing(S) \prec$
$\quad \sim extensive\_integration\_testing(S).$
$\sim safe\_software\_testing(S) \prec$
$\quad \sim extensive\_simulator\_testing(S).$

$software\_module(therac\_25\_software).$
$\sim extensive\_unit\_testing(therac\_25\_software).$
$extensive\_integration\_testing(therac\_25\_software).$
$use\_hours\_without\_problems(therac\_25\_software, 2700).$

Fig. 6 Rules describing failed testing

$safe(A) \prec accelerator(A),$
$\quad includes\_hardware\_safety\_interlocks(A),$
$\quad radiation\_overdose\_misfunction\_reported\_cases(A,0).$
$includes\_hardware\_safety\_interlocks(therac\_20).$
$\sim safe(A) \prec accelerator(A),$
$\quad \sim includes\_hardware\_safety\_interlocks(A),$
$\quad based\_on\_previous\_accelerator(A,PA),$
$\quad\quad accelerator(PA),$
$\quad\quad includes\_hardware\_safety\_interlocks(PA).$
$based\_on\_previous\_accelerator(A,PA) \prec older\_model(PA,A).$
$\sim includes\_hardware\_safety\_interlocks(therac\_25).$

Fig. 7 Rules for expressing lack of safety interlocks

to shared variables but it did not allow to test and set for variables as indivisible operations (see Fig. 8). From this point of view, an argument for the non-safety of Therac-25 can be derived.

Finally, if we add the information establishing that at least one patient died, then we get only one tree formed by a single argument based on the strict rule that establishes that an accelerator is not safe if a patient died. So, an accelerator is not definitely safe if a single overdose problem is reported. In the case of Therac-25, five people died of radiation overdose (see Fig. 9). On the light of this final evidence, the safety decision would be that the accelerator is not safe as the safety recommendation of $\mathscr{P}$ w.r.t. to $safe(A)$ is *Do not Conform.*

IV. Related Work

A software controller for radiological devices has been simulated in [14], aiming at fine-tunning the radiological

$\sim safe(A) \prec software\_inside(A,M), unsafe\_module(M).$
$unsafe\_module(M) \prec software\_module(M),$
$\quad runs\_on(M,S),$
$\quad unsafe\_operating\_system(S).$
$unsafe\_operating\_system(S) \prec$
$\quad \sim safe\_operating\_system(S),$
$\quad operating\_system(S).$
$\sim safe\_operating\_system(S) \prec$
$\quad real\_time\_operating\_system(S),$
$\quad allows\_concurrent\_access\_to\_shared\_variables(S),$
$\quad \sim test\_and\_set\_for\_variables\_as\_indivisible\_operations(S).$
$real\_time\_system\_operating\_system(pdp\_11\_os).$
$allows\_concurrent\_access\_to\_shared\_variables(pdp\_11\_os).$
$\sim test\_and\_set\_for\_variables\_as\_indivisible\_operations(pdp\_11\_os).$

Fig. 8 Rules for expressing unsafe operating system

$\sim safe(A) \leftarrow accelerator(A),$
$\quad\quad problem\_detected(A, overdose, Place, Date).$
$problem\_detected(therac\_25, overdose, marietta\_georgia, jun\_1985).$
$problem\_detected(therac\_25, overdose, yakima\_washington, dec\_1985).$
$problem\_detected(therac\_25, overdose, hamilton\_ontario, jan\_1986).$
$problem\_detected(therac\_25, overdose, tyler\_texas, mar\_1986).$
$problem\_detected(therac\_25, overdose, tyler\_texas, apr\_1986).$

Fig. 9 Rules and facts expressing overdose reported cases

equipment. Complementary, the results as those obtained in [14] represent inputs of our argumentation machine. Thus, the pieces of evidence collected from [14] feed the DeLP system to reason about safeness of radiological devices.

A model driven approach improves the safety of Patient-Controlled Analgesic infusion pumpsin [15]. The UPPAAL formally verifies the safety properties of the controlling software. By enacting argumentation theory, we aimed to fill the gap between such low level safety specifications and high level safety cases required by conformance decisioneers.

Similar to our work, the goal in [16] has been to increase confidence in the medical-related software by enacting argumentation theory. A safety pattern is proposed for generic infusion pumps, modeled in the Goal Structuring Notation [17] graphical argumentation language. By employing defeasible logic, we additionally perform reasoning on the given model.

The rewrite logic has been used in [18] to verify safety of life-critical medical devices such as pacemakers, ventilator machines, or infusion pumps for pain medication. With the goal to establish safe medical design patterns, the work in [18] focuses on the programmers perspective. We argue that, the defeasible logic better supports decisions of different actors involved in the approval of medical device software.

One line to prove that medical-related software is safe emphases the need of traceability. The Med-trace method [19] identifies the requirements for traceability through each phase in the medical software life-cycle. Similar to our work, the need to increase transparency of health-care software was addressed in [19]. Different from our work, the focus in [19]

is on the development life-cycle of software from the project management perspective. We focus on organizing evidence from different perspectives, by enacting a formal method based on argumentation semantics of defeasible logic.

V. Conclusions

By using argumentation theory, we aimed at structuring the available evidence to demonstrate compliance with medical safety requirements and avoidance of hazards. Our method is able to automatically identify inconsistencies in the argumentation chain. Identifying inconsistencies or flaws in the justification chain signals a possible breach in the functionality of the medical device.

The benefits of using argumentation theory regard the improvement of comprehension of the safety argument among all the key stakeholders (software developers, safety engineers, medical certification bodies, physicians, operators of the device).

Conflict of Interest

The authors declare that they have no conflict of interest.

Acknowledgements

Part of this work was supported by the Romania-Argentina Bilateral Agreement entitled “ARGSAFE: Using argumentation for justifying safeness in complex technical systems” (MINCYT-MECTS Project RO/12/05) and Universidad Nacional del Sur, Argentina. Adrian Groza is supported by the intern research project at Technical University of Cluj-Napoca, Romania: “GREEN-VANETS: Improving transportation using Car-2-X communication and multi agent systems”. Carlos Chesñevar is funded by Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET).

References

1. Fu Kevin, Blum James. Controlling for cybersecurity risks of medical device software *Communications of the ACM.* 2013;56:35–37.
2. Olah P., Dobru D., Ciupa R.V., Marusteri M., Bacarea V., Muji M.. Database External Level Architecture for Use in Healthcare Information Systems in *Int. Conf. on Advancements of Medicine and Health Care through Technology* (Vlad Simona, Ciupa RaduV.. , eds.);36 of *IFMBE Proceedings*:36-39Springer Berlin Heidelberg 2011.
3. Graydon Patrick J., Habli Ibrahim, Hawkins Richard, Kelly Tim, Knight John C.. Arguing Conformance *IEEE Software.* 2012;29:50-57.
4. Chung Paul, Cheung Larry, Machin Colin. Compliance Flow - Managing the compliance of dynamic and complex processes *Know.-Based Syst.*. 2008;21:332–354.
5. Defence UK Ministry. 00-56 Safety Management Requirments for Defence Systems *UK Ministry of defence.* 2007;4.
6. Campillo-Artero Carlos. A full-fledged overhaul is needed for a risk and value-based regulation of medical devices in Europe *Health Policy.* 2013;113:38 - 44.
7. McCulloch Peter. The EUs system for regulating medical devices *BMJ: British Medical Journal.* 2012;345.
8. García A., Simari G.. Defeasible Logic Programming An Argumentative Approach *Theory and Prac. of Logic Program.*. 2004;4:95-138.
9. Chesñevar Carlos Iván, Maguitman Ana, Loui Ronald. Logical Models of Argument *ACM Computing Surveys.* 2000;32:337-383.
10. Bench-Capon Trevor J. M., Dunne Paul E.. Argumentation in Artificial Intelligence *Artificial Intelligence.* 2007;171:619–641.
11. Simari G., Loui R.. A Mathematical Treatment of Defeasible Reasoning and its Implementation *Artificial Intelligence.* 1992;53:125-157.
12. Williams M., Hunter A.. Harnessing ontologies for argument-based decision-making in breast cancer *Proc. of the Intl. Conf. on Tools with AI (ICTAI'07).* 2007:254–261.
13. Leveson Nancy, Turner Clark. An Investigation of the Therac-25 Accidents *IEEE Computer.* 1993;26:18–41.
14. Roman N.M., Colosi H., Pusca M.. Digital Simulation for Computer Control of Radiological Devices: A Preliminary Model Using Partial Differential Equations in *Int. Conf. on Advancements of Medicine and Health Care through Technology* (Vlad Simona, Ciupa RaduV., Nicu AncaI.. , eds.);26 of *IFMBE Proceedings*:37-42Springer Berlin Heidelberg 2009.
15. Kim BaekGyu, Ayoub Anaheed, Sokolsky Oleg, et al. Safety-assured development of the gpca infusion pump software in *Proceedings of the ninth ACM international conference on Embedded software*:155–164ACM 2011.
16. Ayoub Anaheed, Kim BaekGyu, Lee Insup, Sokolsky Oleg. A safety case pattern for model-based development approach in *NASA Formal Methods*:141–146Springer 2012.
17. Spriggs John. *GSN - The Goal Structuring Notation: A Structured Approach to Presenting Arguments*. Springer 2012.
18. Sun Mu, Meseguer José, Sha Lui. A formal pattern architecture for safe medical systems in *Rewriting Logic and Its Applications*:157–173Springer 2010.
19. Regan Gilbert, Caffery Fergal Mc, Daid Kevin Mc, Flood Derek. Medical device standards' requirements for traceability during the software development lifecycle and implementation of a traceability assessment model *Computer Standards and Interfaces.* 2013;36:3 - 9.