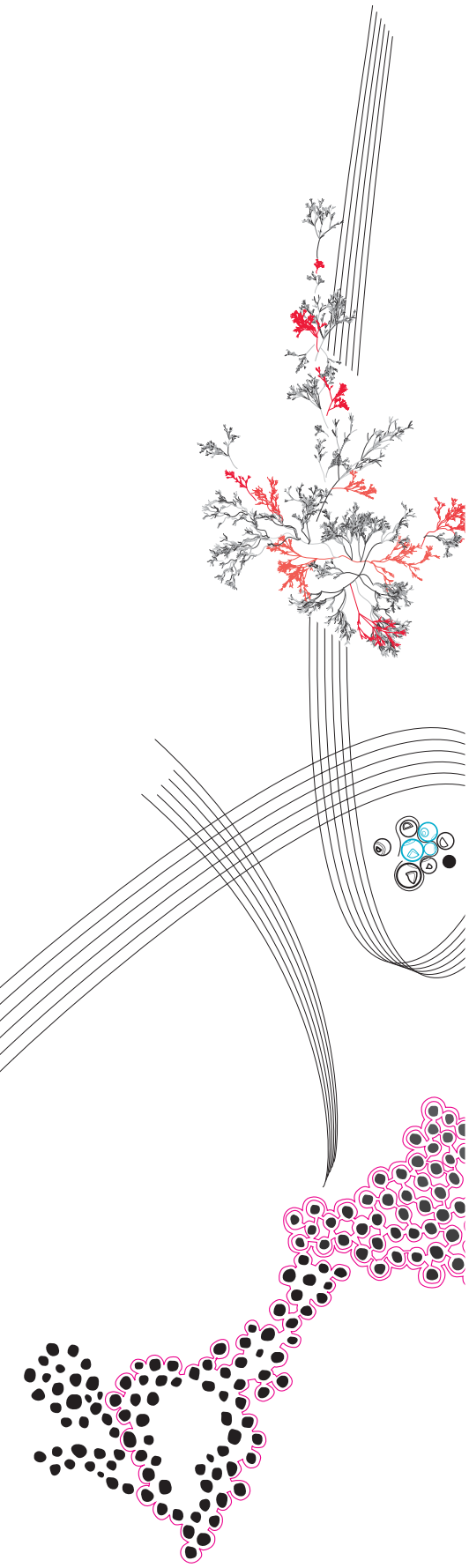

Automatic failure analysis for big data-driven industry

Bart Verkuil

October, 2021

Graduation committee:
Dr. Doina Bucur
Dr. ir. Vadim Zaytsev
Dr. ir. Maurice van Keulen

MSc Computer Science, Data Science & Technology
University of Twente

Preface

This work is the result of several months of Internship at Intergas where I was given access to large datasets containing information about boiler usage. After talking with Intergas about new uses for the enormous amounts of data they had collected over the past years we decided on exploring ways to automatically construct fault trees from the data. This was directly in line with my interests, and it would help Intergas better understand how errors could propagate through their products. I enjoyed my time testing different algorithms and was glad to see that the end product was able to automatically generate useful trees. Even though progress during this project was not always as quick as I initially had hoped for, I am pleased with the end result.

I would like to thank my primary supervisor, Doina, for her valuable feedback and guidance during this project. I would also like to thank Maurice and Vadim who joined my graduation committee in a later stage and also gave tips on how to improve this research. I would like to thank my primary contact at Intergas, Peter Cool, for putting me in contact with the right people at Intergas and helping me out with any questions I had. I also would like to thank Steven Verkuil for setting up an environment where I could access all the data I needed for my research.

Bart Verkuil
October 2021

Abstract

Many industries have been increasing their interest in data collection and storage over the past decades. With recent technologies allowing huge amounts of data to be processed with relative ease, companies are constantly looking at how their data may help them find and prevent failures. In this work we propose a combination of existing methods for failure analysis which we adjusted to work on continuous big data to automatically learn fault trees from observational data. Our solution scales well and is tested on a real-world dataset of domestic boiler usage. Our approach is based on two previous methods, the C4.5 algorithm for converting continuous to Boolean data, and the LIFT algorithm for learning fault trees from the Boolean data. We use the C4.5 algorithm to automatically split variables on a optimal threshold, replacing the continuous values with a 0 or 1 depending on whether the value is below or above the threshold. This Boolean split allows us to distinguish between faulty and normal operation of the variable and makes it suitable as input for the LIFT algorithm. Besides evaluation of our results we also provide critical feedback on potential problems with previous approaches.

Contents

1 Introduction

1.1 Problem statement and approach

Failure analysis has been an important tool for many industries over the past decades. Sufficient failure analysis can lead to early detection of hazardous situations whereas insufficient failure analysis can lead to unreliable products. The roots of formal failure analysis begin roughly 60 years ago with traditional methods like failure mode and effect analysis (FMEA), followed by other techniques such as fault tree analysis (FTA) [17]. At the basis of FTA lies the production of a fault tree that encodes a system's components as a set of connected events and gates. One major drawback of FTA in practice is that it often requires manual labour from domain experts. Especially in larger systems this is error-prone and sometimes the effort it requires to do FTA simply does not outweigh the potential benefits. However, following recent trends in industry to make products smart, new possibilities to automate the process of constructing fault trees are available.

In this research we provide a new method to aid companies that have access to large datasets of their products' health and status to perform automated FTA. Our work provides the ability to automatically generate fault trees for known component and system failures to help domain experts understand and evaluate the cause of a problem. The main research question and the sub-questions that lead to this paper are:

- How can we automate the production of fault trees directly from observational big data?
  - What are the shortcomings of existing approaches?
  - How accurate is our approach on a real-world dataset?

Methods to directly generate fault trees from observational data have recently been attempted, for example with LIFT [11], but not on the scale of this research yet. This work will reuse parts of the LIFT algorithm, but also make necessary adjustments to scale to big data. There are also semi-automatic tools available for domain experts to translate different models to fault trees, but they often require prior system knowledge or models to work properly. Our approach does not require additional system information which makes it very approachable.

To test our implementation and answer the second subquestion we collaborated with one of the leading producers of high-efficiency boilers, Intergas[1]. Next to having cutting-edge efficiency of boilers, Intergas also prioritizes data collection of their products. Almost all installed boilers communicate with Intergas about their status and health regularly, which is stored in a database. The boilers are installed in domestic environments throughout the Netherlands. We used this historical real-world data of

[1] https://www.intergas-verwarming.nl/en/consumer/

the heaters as an input for our algorithms and produce fault trees for multiple known system failures as output. The considered dataset contains more than 30.000 unique heaters and millions of datapoints. Each of these datapoints contains information about the minimum value, maximum value and the average of 27 variables measured each day per heater. This case study shows us that our approach is scalable and capable of correctly constructing fault trees. We believe that our case study provided enough evidence that our method is applicable to other industries as well.

1.2 Background

Fault trees are used to model how component failures can propagate through a system, eventually resulting in system-wide failures. We provide an example fault tree in Figure 1. The top of the tree contains the event we want to investigate, called the *top event*. This is often an error or failure preventing normal operation of a product. The other events in a fault tree are either basic or intermediate events, where intermediate events are the events found above a gate. Whilst basic events can occur randomly and are often annotated with a probability, intermediate events are a result of at least two other events combined in a gate. In Figure 1, $bc\_tapflow \leq 0.0$ is a basic event and activates when the variable $bc\_tapflow$ gets below 0.0. The $ch\_pressure \leq 0.49$ event is an intermediate event and is connected through an OR gate with the bottom two basic events. This example fault tree shows us when the error *Warning low ch_pressure* is most likely to occur and gives domain experts knowledge in which components might be responsible. In this work we consider two different gates: AND gates and OR gates. AND gates give an output if both input components fail and OR gates activate when at least one of the inputs fails.

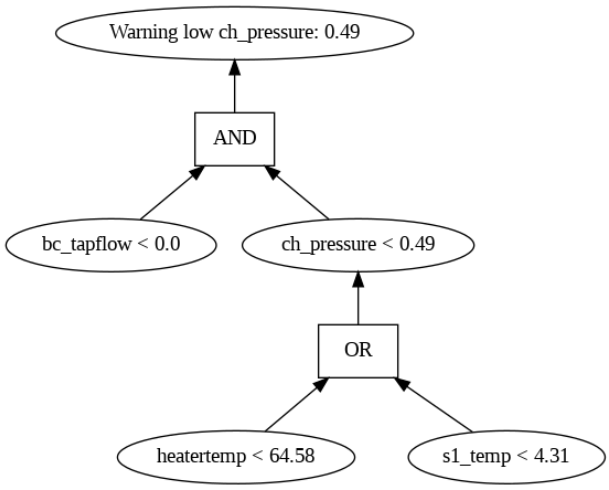

Figure 1: Example fault tree of the Intergas boiler dataset regarding the error *Warning low ch_pressure*.

1.3 Related work

Even though our specific problem and solution has not been seen in other research yet, attempts of automating fault trees have been made before. Most of the existing methods rely on Model-Based Dependability Analysis (MBDA), meaning that other system descriptions are needed to reason about causality or association of an observational dataset. In our work we only consider the raw data and do not require any further information about the system. When more information about the system is known however we refer to Kabir [4], who provides an excellent overview of how to translate existing formal models to fault trees.

Automatic fault tree generation from data One of the earlier attempts of modelling causal effects from data is the IFT algorithm. It can produce fault trees including AND/OR gates from observational data without additional system knowledge. The core of the IFT algorithm is based on the ID3 algorithm for the induction of decision trees [12]. This has as drawback that the resulting fault tree is equivalent to a decision tree classifier that the ID3 algorithm would produce. This approach disregards the causality between events occurring in a system.

Another approach is the LIFT [11] algorithm. Just like our research, the LIFT algorithm only requires observational data and produces a fault tree encoding relations between variables. LIFT produces a fault tree with Boolean events, AND/OR gates and has the possibility to annotate failure probabilities. A key difference between our research and the LIFT algorithm is the input of LIFT is only compatible with Boolean data, whereas our research extends this to also deal with continuous data. Even though we use core parts of the LIFT algorithm in this paper, we also found some issues which we documented in section 3. Another difference is that we consider association in the data instead of causation, but we argue that the authors of LIFT also did not implement a correct method for measuring causality.

A second data-driven approach is seen in the work of Linard et al. [8]. Here an evolutionary algorithm is proposed to find an optimal fault tree for a given Boolean observational dataset. Their approach shows promising results regarding accuracy of fault trees on synthetic data when compared to other implementations, but once again does not accept the continuous data which is often present in real-world datasets.

Other approaches A recent paper [9] investigated the possibilities of translating Bayesian networks to fault trees. A special type of Bayesian networks called causal Bayesian networks (also known as causal graphs) can successfully be constructed from observational data. Using causal graphs as an intermediate step, these methods can lead to a successful method of constructing fault trees from observational data. Although we do not extensively explore this possibility in this research, a successful implementation of causal graph inference is found in the work of Gao et al. [1]. In their research a real-world dataset of Alzheimer's disease is used to automatically construct causal graphs, which are in turn used to form an analysis framework. Re-

sults showed that they were able to produce consistent and accurate frameworks that could enhance knowledge about gene function and disease etiology. A different research [6] automatically generated causal graphs to study the causal effects of miRNA on mRNA. They show that they are able to identify relationships between miRNAs and mRNAs using causal graphs, and follow-up experiments verify the validity of their findings. Summarizing papers describing the use of and approach to automatic causal graph generation are: The work of Guo et al. [2], which provides an extensive overview of the algorithms to use, and the work of Raghu et al. [13], which evaluates a selection of these algorithms on simulated data and reports on the usability in different scenarios.

Compared to the related work our research improves on multiple aspects. One of the key points is allowing the input data to contain continuous data, whilst not requiring additional system information. We also tailored our algorithms to be compatible with big data, which is often the nature of real-world datasets. By constructing fault trees from raw observational data without needing any intermediate steps (such as constructing a causal graph) we improve on the efficiency of other proposed solutions.

2 Description of Dataset

In this section we further explain the dataset provided by Intergas. The dataset contains continuous observational data of domestic boilers installed throughout the Netherlands. We show how the data is obtained, structured and processed to be compatible with our algorithms.

2.1 Data collection

The collection and storage of our dataset is a combined effort between the installed heaters and the storage system running at Intergas. This process starts at a heater. A heater contains multiple sensors measuring various internal or external parameters. Although the usage and number of sensors can differ between different types of heaters, sensor 1 and sensor 2 are used to do safety-critical measurements. They are placed in the heat exchanger and measure whether the heat from the gas is correctly transferred to the water. We provide a schematic overview of a boiler in Figure 2, where we show the placement of components measuring the variables used in our case study. For a full description of the other components we reference the Intergas installation guide [3]. The values of their components are read by a control loop running in the heater. If a sensor reports critical values the device will be turned off automatically.

The storage of the data happens through multiple Python scripts, MySQL databases and a Hadoop distributed file system (HDFS) environment. During each period of 24 hours the data of the heaters is collected and stored temporarily. Once every 24

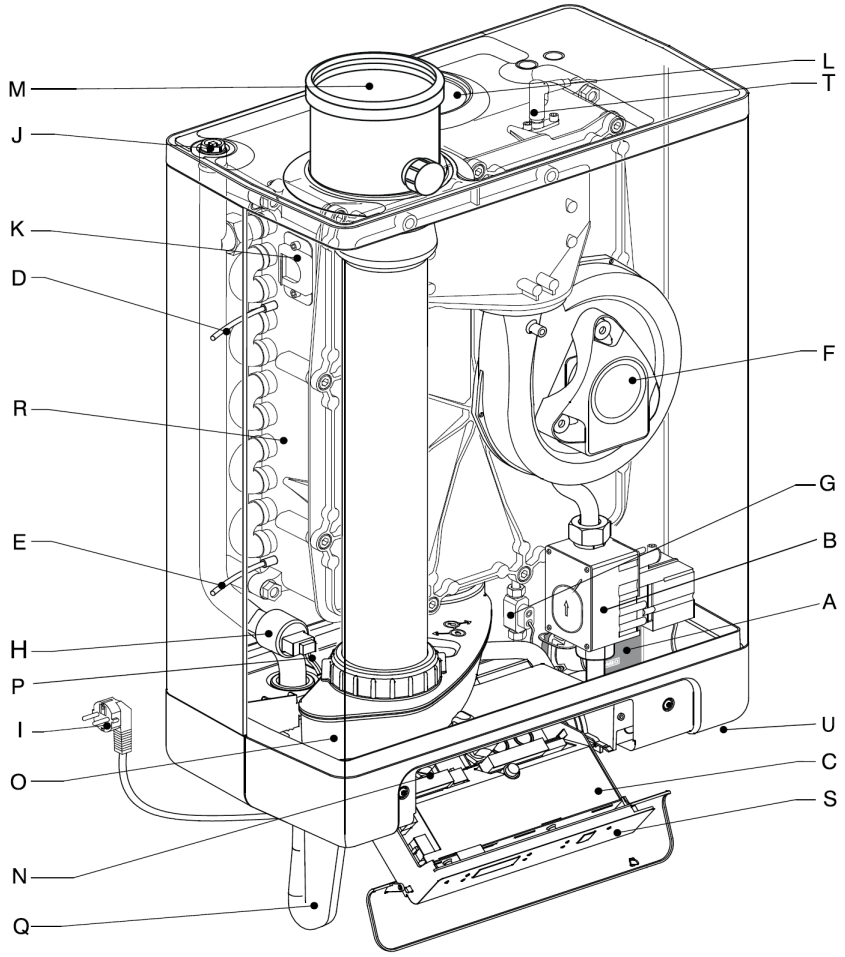

Figure 2: Schematic overview of a boiler, provided by Intergas. D: Sensor 1 (*s1_temp*), E: Sensor 2 (*s2_temp*), G: Flow sensor (*waterflow_ch*), H: Pressure sensor (*ch_pressure*), P: Sensor 3 (*s3_temp*). For a full overview of components see the Intergas installation guide [3].

hours a script will run to aggregate, merge and restructure the data and store it in the database, which can manually be exported to the HDFS.

2.2 Data description

In our research we use the data available in the HDFS as a starting point. In total we have access to multiple terabytes of raw data, containing more than 31 million unique heater-day pairs. Each of the files contains the values of 27 variables for a single day. We provide Table 1 for a description of the variables. For each of these variables 3 numbers are recorded; the minimum value the variable reached, the average value and the maximum value, all measured over a 24 hour period. The files also include extra information about exact timestamps when room temperatures are changed and

water is used, but we choose to only use the $[min, average, max]$ as input for our algorithms. As we will show in section 4 the values of the variables can be directly linked to errors. An example of such an event is the variable *ch_pressure* reaching a critical value as shown in Figure 3. We plotted the data of the pressure variable of a single heater over the course of 100 days. We can clearly see a drop at the 60 day mark where the minimum pressure value drops below a threshold of 0.49, caused by a system failure which was also recorded in the Intergas database. At day 83 the pressure in the heater is restored, probably by filling it manually with additional water.

Variable	**Description**
bc_tapflow	Water usage (l/min)
boilertemp	Temperature of boiler (°C)
burnerstarts_24h	Number of starts per 24 hour
ch_pressure	Water pressure (bar)
flue_sided_resistance	Resistance of the flue duct
gasmeter_ch_24h	Gas used per 24 hour (m^3) for central heating
gasmeter_dhw_24h	Gas used per 24 hour (m^3) for domestic hot water
heaterload_ch_24h	Delivered power percentage of full power over the last 24 hour, per hour in central heating mode
heaterload_ch_total	Delivered power percentage of full power over the last 24 hour, in central heating mode
heaterload_dhw_24h	Delivered power percentage of full power over the last 24 hour, per hour in domestic hot water mode
heaterload_dhw_total	Delivered power percentage of full power over the last 24 hour, in domestic hot water mode
heatertemp	Boiler temperature (°C)
io_curr_high	Ionization current on high output power (µA)
io_curr_low	Ionization current on low output power (µA)
outside_temp	Outside temperature (°C)
override_outside_temp	Alternative temperature measure(°C)
pump_pwm	Pulse width modulation control signal for modulating pump (percentage)
room_override_zone1	Override temperature for zone 1 (°C)
room_override_zone2	Override temperature for zone 2 (°C)
room_set_zone1	Set temperature for zone 1 (°C)
room_set_zone2	Set temperature for zone 2 (°C)
room_temp_zone1	Measured temperature for zone 1 (°C)
room_temp_zone2	Measured temperature for zone 2 (°C)
s1_temp	Supply water temperature (°C)
s2_temp	Return water temperature (°C)
s3_temp	Warm water temperature (°C)
waterflow_ch	Calculated waterflow through the central heating ducts.

Table 1: Description of variables in Intergas dataset

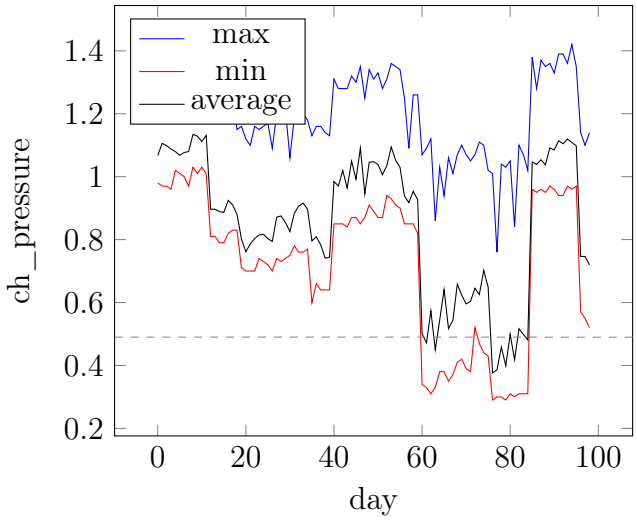

Figure 3: Example of the *ch_pressure* variable for a heater over time, which a sudden drop below the warning threshold of 0.49 at day 60

For our case study we considered 11 errors, which are given in table 2. For each of these errors notifications are sent by a heater when they occur, which allows us to find the values of the variables of the heater on the day the error was reported.

Error	Description
Lockout: code 0	Sensor fault after self check
Lockout: code 4	No flame signal
Lockout: code 5	Poor flame signal
Lockout: code 8	Incorrect fan speed
Lockout: code 11	Sensor fault related to S1
Lockout: code 13	Sensor fault related to S1
Warning Flame lost from 0 to 1	Flame lost
Warning Ignition failed from 0 to 4	Ignition failed 4 times
Warning low ch_pressure: 0.49	Channel pressure dropped below 0.49
Warning low t1: 0.0	Value of variable s1_temp reached 0.0
Warning low t2: 0.0	Value of variable s2_temp reached 0.0

Table 2: Errors present in the Intergas dataset

2.3 Data quality

Because we are dealing with real-life data we to take into consideration that some data may be incorrect. Incorrect data can be of the form of missing data, duplicated data, or corrupt data. Internal research at Intergas on the same dataset we use found that for an extended period each entry got duplicated multiple times. Similarly, there

have been known cases of missing data in the same dataset. The cause of missing data often lies with either the heater or a connecting gateway having connection issues. Also, because communication between Intergas Data and gateways uses UDP, there is no guarantee that messages are delivered and will not duplicated.

To give an indication of the data quality we provide Figure 4, which displays the number of unique heaters recorded per day. The first records are from 2015, with our newest data coming from 2020. We can see that there are clear drops in the line, indicating that there are missing days and periods of data collection. When we looked at the individual heaters we see that roughly 90% of all the heaters in our data miss at least one data entry between the first and last record. We also found some heaters reporting unrealistic values for some variables or simply having an enormous amount of failures. We decided to manually omit these heaters from our experiments. The process of cleaning the input data is not part of our proposed solution.

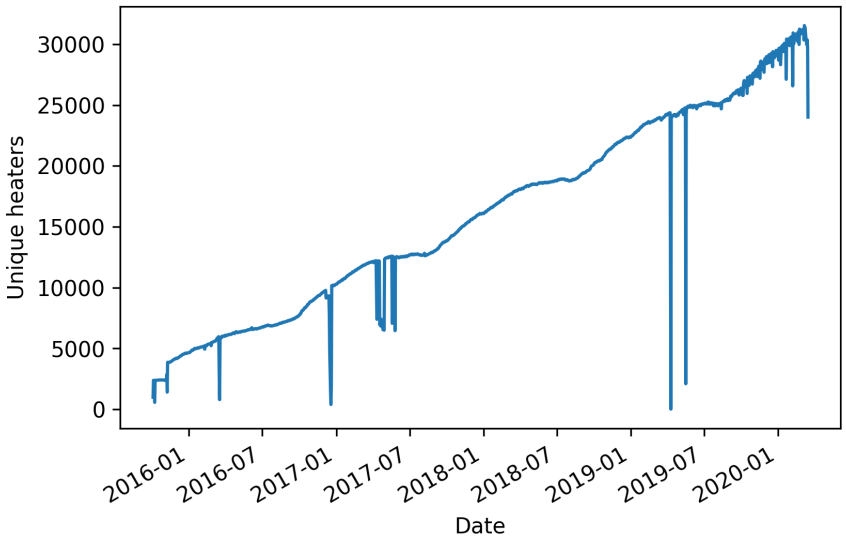

Figure 4: Number of unique heaters recorded per day in the Intergas dataset. Sudden drops indicate failure in the data collection system.

3 Automatic Fault Tree Generation

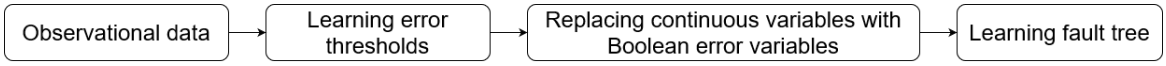

Figure 5: Steps of converting observational data to a fault tree

We provide Figure 5 for a quick overview of the steps we take to convert observational data to fault trees. In this section we give a detailed description of how each of these steps work. Even though we tailored our algorithms to suit the dataset Intergas provided, we believe that other datasets are easily converted to be usable by our algorithm as well.

The input dataset is expected to have a tabular structure. We require a column for each of the variables available in the dataset and an additional column indicating whether the row is part of data collected when a system-wide failure was observed. This system-wide failure will be the top event of the eventual fault tree. If the row is part of the faulty data a 1 is expected in the *Error* column, otherwise a 0. Each complete run of our algorithm considers the construction of a fault tree for a single error, meaning that for our Intergas dataset we run the algorithm 11 times with different values in the *Error* column each run. The values in the other columns are each a triplet of 3 continuous values. The list starts with the minimum value of the variable, followed by the average and maximum value. Depending on the time interval between datapoints, the interpretation of the minimum, average and maximum can differ. In our case study we have data per day, meaning that each row in the dataset represents the minimum, average and maximum for each of the variables for a single day for an unique heater. When running our algorithm we can also calculate the range of each variable by subtracting the minimum from the maximum. To visually illustrate how an suitable dataset would look like we provide Table 3, where we have n variables and m datapoints and also the required *Error* column. For our case study we labelled all datapoints recorded on a day that an error occurred with a 1 in the *Error* column, and the data collected a day before an error with a 0.

Variable 1	Variable 2	...	Variable n	Error
$[min, average, max]_1$	$[min, average, max]_1$	...	$[min, average, max]_1$	0/1
$[min, average, max]_2$	$[min, average, max]_2$	...	$[min, average, max]_2$	0/1
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$[min, average, max]_m$	$[min, average, max]_m$	...	$[min, average, max]_m$	0/1

Table 3: Example of how a correct input should look like with n variables and m datapoints.

3.1 Translating continuous to Boolean

To allow the use of continuous variables as input data we need to translate them to Boolean variables before feeding them to the algorithm. This makes sense because fault trees only encode Boolean relations. For this translation we use the C4.5 algorithm [16], most used in decision trees. C4.5 proposes to apply a binary split based on a threshold value. We can determine this optimal threshold value for each attribute by maximizing the gain guided by the *Error* column. What we mean by this is that we replace all continuous values observed for a variable with a 0 or 1 based on if it is below or above a threshold. A 0 means that the component related to the variable is operating normally and is not likely to contribute to the system-wide failure (top event) occurring. A 1 means that the variable is behaving more like it would in the event of the system-wide failure happening, suggesting something might be wrong. The C4.5 algorithm extensively tests all unique values as a splitting threshold and picks the threshold that maximizes the information gain [5]. To calculate the information gain we need to calculate the entropy using equation 1.

$$E(S) = \sum_{i=1}^{c} -p_i \cdot log_2 p_i \tag{1}$$

Here the S stands for the error or variable (with potential conditions) we want calculate the Entropy for. The c stands for the number of classes (always two in our case), and p_i is simply the probability of the class occurring in our data. One class is the data that is part of the normal behaviour dataset (a 0 in the *Error* column), and the other class is data from when an error occurred. We can then calculate the gain of a proposed split using equation 2.

$$Gain(S) = E(before) - p_{left} \cdot E(left) - p_{right} \cdot E(right) \tag{2}$$

In this formula the entropy of *before* is the entropy of the start situation. The entropy of *left* and *right* are the entropy of the left and right part of a proposed split. By weighing the entropy of both sides of the split by their probability and subtracting it from the starting entropy we get the gain. The gain tells us the amount of entropy removed by the split. In a perfect split all datapoints below a threshold would be part of one class (error or normal data), and all datapoints above the threshold would be part of the other class.

Using this approach we replace all continuous values with a Boolean value that best represents whether the value of the variable is related to the occurrence of an error. To give an idea of how this approach works in practice we provide the following example using a known error from the Intergas dataset.

Example An example of the process of calculating the information gain for each value is found in Figure 6. Here we show the normal and error data overlapped with the gain for each value of *s2_temp* if we were to split on that value. The gain is multiplied by a factor of 500 to fit the scale of the graph. We see that the gain is maximized for 21, indicated by the red vertical line. To demonstrate how we calculate the gain for a given value we will work through an example with 16 as threshold and see which gain value that would have given us. To start we first need to consider the entropy, which was given by equation 1.

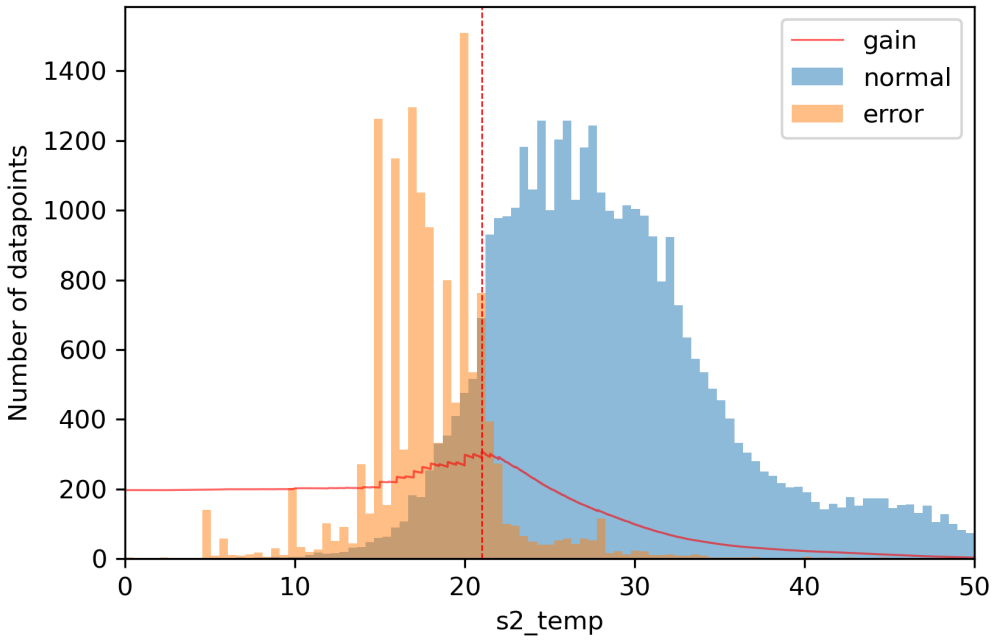

Figure 6: Continuous *s2_temp* variable split with a threshold of 21

Since we would like to optimize a split regarding a certain variable in relation to our *Error* variable we first calculate $E(Error)$. To help illustrate our procedure we take a split of the continuous variable $s2\_temp$ in regards to the error *Warning low t1* from our Intergas dataset as an example. In this example we use the minimum value of the $s2\_temp$ variable. Calculating the entropy we get:

$$\begin{aligned} p_1 = \frac{37866}{74421} \qquad p_2 = 1 - p_1 = \frac{36555}{74421} \\ E(Error) = -p_1 \cdot log_2(p_1) - p_2 \cdot log_2(p_2) \approx 0.99 \end{aligned} \tag{3}$$

The next step it to calculate $E(Error|s2\_temp \leq x)$ and $E(error|s2\_temp > x)$ for each of the continuous values x that are part of the $s2\_temp$ column. We use

$x = 16.0$ in our example. We get the following calculations:

$$\begin{aligned} p_1 = \frac{27272}{28116} \qquad p_2 = 1 - p_1 = \frac{844}{74421} \\ E(Error|s2\_temp \leq 16.0) = -p_1 \cdot log_2(p_1) - p_2 \cdot log_2(p_2) \approx 0.19 \end{aligned} \tag{4}$$

$$\begin{aligned} p_1 = \frac{10594}{46305} \qquad p_2 = 1 - p_1 = \frac{35711}{46305} \\ E(Error|s2\_temp > 16.0) = -p_1 \cdot log_2(p_1) - p_2 \cdot log_2(p_2) \approx 0.78 \end{aligned} \tag{5}$$

We can now calculate the information gain, which was given by equation 2. The entropy of *left* and *right* are given in (4) and (5) respectively, and are the entropy of the left and right part of the proposed split. We need to weigh the entropy of both sides of the split by their probability and subtract it from the starting entropy to get the gain.

$$\begin{aligned} Gain(Error, s2\_temp\ split\ 16.0) = E(Error) &- p(Error|s2\_temp \leq 16.0) \\ &\cdot E(Error|s2\_temp \leq 16.0) \\ &- p(Error|s2\_temp > 16.0) \\ &\cdot E(Error|s2\_temp > 16.0) \\ &= 0.99 - \frac{28116}{74421} \cdot 0.19 - \frac{46305}{74421} \cdot 0.78 \approx 0.44 \end{aligned} \tag{6}$$

In a full run of the algorithm we would calculate the gain for each of the potential splits and choose to split our continuous variable on the value that yielded the maximum gain. In this example the split with the highest gain would be on the value of 21, as shown in Figure 6 with a red vertical line. This means that if we would like to construct a fault tree for the error *Warning low t1* we should split the variable $s2\_temp$ on 21, and replace all $s2\_temp$ values below 21 with a 1 and all values above 21 with a 0. The reason that the values below 21 are labelled faulty with a 1 in this specific case is because there are more faulty datapoints below the threshold of 21 than above. In this example we only considered one variable, whereas in reality this process will take place for all variables provided in the input before passing it to the LIFT algorithm.

3.2 Measuring causal effect

In this section we explain the current approach of the LIFT algorithm for constructing gates and measuring causality. In subsection 3.2.1 we provide an overview of the

problems that come with the current implementation and give an alternative approach in subsection 3.2.2. To optimize the LIFT algorithm for our case study we provide an overview of further alterations to LIFT in section 3.3

The LIFT algorithm applies an exhaustive method to find suitable gates for a fault tree describing an error E. To determine the significance of a proposed gate G we need a way of measuring the causal effect of G on an error E. A potential candidate is the Mantel-Haenszel test [10]. The Mantel-Haenszel test operates on the concept of stratifying the data and applying an algorithm to score the causality. With stratifying the data we mean the process of constructing multiple tables (strata) where we count the state of a gate in relation to the error under investigation. Such a strata looks like:

s_k	$e=1$	$e=0$	*Total*
$g=1$	A	B	N_1
$g=0$	C	D	N_2
Total	M_1	M_2	T

In this example for stratum s_k, g represents a gate (AND/OR) of 2 or more variables. The row for $g = 1$ indicates that when the gate is active, there are A datapoints where there has been an error ($e = 1$), and B datapoints of when the gate would have been active but no error was observed. Similarly, when the gate is not activated, B and D represent the amount of times we see a datapoint with and without an error respectively. To say something about the amount of causation for a stratum we use the PAMH score. For this score we also require the totals of the rows and columns, which are visible in the example stratum s_k. The PAMH score is calculated as follows:

Algorithm 3.1.

$$PAMH(g,e) = \frac{\left(\left|\sum_{i=1}^{K} A_i - \frac{N_{1i}M_{1i}}{T_i}\right| - \frac{1}{2}\right)^2}{\sum_{i=1}^{K} \frac{N_{1i}N_{2i}M_{1i}M_{2i}}{T_i^2(T_i-1)}}$$

This algorithm sums the values over K strata, one for each potential choice of variables in the input of g. Depending on which values for variables are chosen, the values in the stratum will differ, which is shown in the PAMH formula with a i for the i-th stratum.

The reason why this approach says something about causality rather than association is because strata are constructed for each possible combination of a gate and the remaining variables, which are then weighed appropriately by the PAMH algorithm. The PAMH score can be seen as the test statistic which follows a χ^2 distribution with 1 degree of freedom. Just like other statistical tests we set a null hypothesis and compare the outcome to determine statistical significance. In our case the null hypothesis is

that gate g and error e are completely independent. We provide appendix B for an example of how the PAMH algorithm works for choosing the best gate for constructing a fault tree from synthetic data.

3.2.1 Problems with PAMH

Even though PAMH looks to be a valid option, problems can occur when we change the dimensionality or size of the data. We would like to address both problems and how these impact the PAMH score.

High dimensional data When dealing with high dimensional data the conventional approach of the Mantel-Haenszel test will produce too many strata. This has three problems. The most obvious is perhaps the exponential computation time. With n free variables (variables not yet in the tree) the PAMH algorithm needs to consider in the worst case scenario 2^n strata. Since strata need to be constructed for each potential gate, poor execution times for large datasets are expected. The second problem is more fundamental. When we apply perfect stratification to high dimensional datasets we obtain many strata with small sizes. Some of these small strata might yield a false observation due to limited availability of samples in the dataset. Combined, these strata can negatively impact the statistical power of discovering dependencies. For the latter problem a solution has been proposed [15]. Instead of constructing all strata over the remaining variables we can use the propensity score [14] to group individual data samples to strata based on this score. Like Li et al. [7] show, this approach can potentially result in a better average recall, but it comes with extra computational cost. Their method with perfect stratification has a time complexity of $O(mn \log n)$ whereas adding a propensity score increases this to $O(mn^\alpha)$ where $2 < \alpha < 3$. Due to this increase in time complexity we do not further investigate this approach of this research.

The third and final problem with the PAMH approach is that large datasets potentially yield unrealistically large PAMH scores. To illustrate we once again consider dataset D_1 and multiply all counts by a factor of 10, and call this dataset $D_1{}'$. Where D_1 included 50 data samples, D_1' includes 500. Even though we increased the size of the dataset, by multiplying all counts with the same factor we did not change the relative distribution. Following the same example as before we calculate PAMH(G, E) for D_1', which yields PAMH ≈ 137. This value is over 10 times larger than the PAMH score calculated in the original example. The cause of this drastic increase of the score can be explained easily if we take a closer look at algorithm 3.1. Here the top of the fraction roughly increases by an order of r^2 if we multiply the strata with a factor r, where the bottom only increases with an order of r. We first encountered this problem when trying to replicate the work of the LIFT authors and applying it on our own datasets. We would obtain unrealistically high PAMH scores which would indicate a P value $< 10^{-16}$. This shows that comparing the PAMH scores to the critical values of the chi-square distribution with 1 degree of freedom is no longer

a suitable test statistic to determine the statistical significance when dealing with large datasets. This does not mean that using the PAMH score is completely useless. The comparison between gates and picking the gate with the highest PAMH score still yields the best tree, but the calculation of the actual significance as is done by the LIFT authors seems to be flawed. For large datasets, line 6 of Algorithm 1 of the LIFT paper where the PAMH score is compared to $\chi^2_{\alpha,1}$ always yields true for large strata. Besides, the LIFT algorithm uses a notation of the PAMH algorithm that does not consider multiple strata for each gate, but rather combines the strata into a single stratum. Although replication of some of their findings using the correct notation given in algorithm 3.1 does not drastically influence their result, not splitting the strata on potential confounders prohibits the intended operation of the PAMH algorithm.

3.2.2 Alternative to the PAMH approach

As an alternative to the Mantel-Haenszel test we can use the phi coefficient. The phi coefficient is equivalent to the Pearson correlation coefficient estimation for two binary variables. It returns a value between -1 and 1 where -1 means a perfect negative association and 1 a perfect positive association. We can calculate the phi coefficient of a 2 x 2 stratum such as the one shown in section 3.2 using:

$$\phi = \frac{AD - BC}{\sqrt{N_1 N_2 M_1 M_2}} \tag{7}$$

One of the advantages of using the phi coefficient is that it is in no way dependent on the datasize. Increasing the datasize without changing the relative distributions between the variables yields a constant phi coefficient. A drawback is that the phi coefficient is a method of measuring association, not causation. Even though we expected this to be a potential problem regarding the quality of our constructed fault trees, while comparing trees constructed using the PAMH score and the phi coefficient we found little differences in tree size and gates. Therefore we replaced the implementation of PAMH in LIFT with our implementation of the phi coefficient.

Example To illustrate how the phi coefficient works we consider the error e: *Warning low t1: 0.0*, which is taken from the Intergas dataset. We want to create the first gate for our fault tree, which will be placed directly below the top event (the error e). We consider the *min* of all variables provided as input, and use the C4.5 algorithm to first translate the continuous values to Boolean like we described in section 3.1. We consider all combinations (up to a user-chosen maximum gate size) of the variables in the Intergas dataset and calculate their respective phi coefficients. At some point the gate $g = heater\_temp < 7.12$ AND $s2\_temp < 21.0$ is considered, which yields the highest phi coefficient. For this gate g the following stratum was constructed:

s_k	$e = 1$	$e = 0$	*Total*
$g = 1$	32494	1901	34395
$g = 0$	4061	35965	40026
Total	36555	37866	74421

Using equation 7 the phi coefficient can be calculated:

$$\phi = \frac{32494 \cdot 35965 - 1901 \cdot 4061}{\sqrt{34395 \cdot 40026 \cdot 36555 \cdot 37866}} \approx 0.84 \tag{8}$$

We will place gate g directly below the top event since it has the highest coefficient, and remove the variables $heater\_temp < 7.12$ and $s2\_temp < 21.0$ as candidates for gates further down the tree.

3.3 Further adjustments to the LIFT algorithm

Due to some programming inefficiencies in the provided code of the LIFT paper we re-implemented some functions making the algorithm roughly 30 times faster.

In the original implementation of LIFT the authors used synthesized datasets for testing and made a difference between noise and significance. They calculated the significance using the PAMH score and defined the noise as the maximum value of either the top left or bottom right value of a stratum divided by the total count of that stratum. The idea behind this noise factor is to filter out incorrect datapoints (for example flipped bits), but we do not find this a suitable approach for our real-world dataset. We ignore the noise factor and only use the significance (the phi coefficient) as an criterion for choosing gates. We leave it up to the providers of a dataset to filter out as much noise and incorrect data before feeding it to our algorithms.

3.4 Description of complete algorithm

A full cycle of our algorithms works as follows. First the continuous variables of the input dataset are translated to Boolean based on the optimized threshold for each variable as described in section 3.1. This is a step that we introduced to our implementation, and is not present in LIFT. We then start a new fault tree with a single event, the system error (top event). For each iteration the algorithm chooses a variable from the tree that is not yet the output of a gate and tries to find the best gate for it. At the start this is only the top event. The algorithm exhaustively checks all combinations of unused variables up to a provided maximum size and calculates the significance for each gate. For our implementation we use the phi coefficient as significance as described in section 3.2.2, which differs from the implementation of the LIFT authors that use the PAMH score. Once all combinations of variables

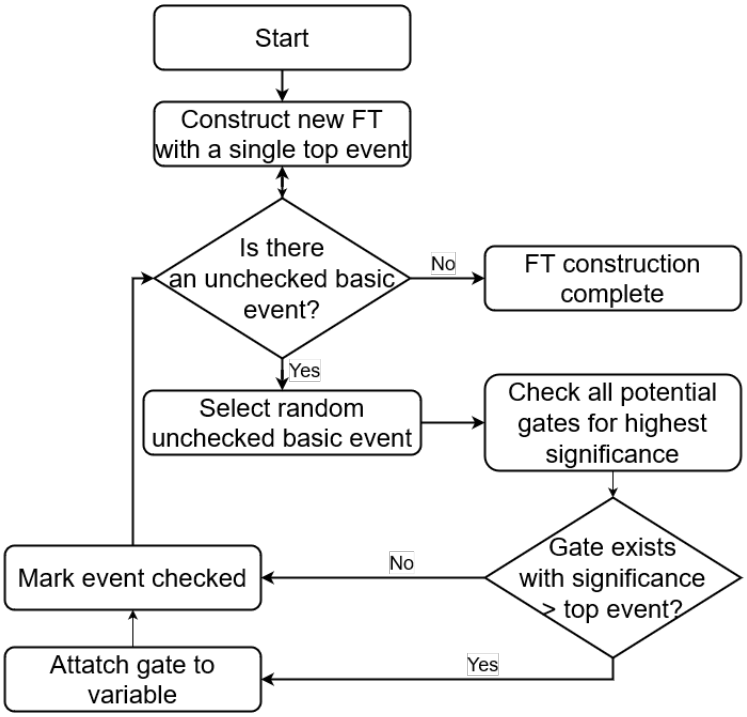

Figure 7: Basic flowchart of fault tree algorithm

are checked, the gate that yields the highest phi coefficient will be attached to the variable. The variables that make up the gate are no longer available for other gates. In the next iteration a new variable in the tree (that is not the output of a gate yet) will be considered and again the potential gates are formed and checked. As an additional criterion for each new gate we require that the significance is at least as high as the significance of the top gate (the gate with the top event as output). This prevents trees from becoming less significant further down the tree you go. This check is also new compared to the implementation of LIFT. This extra requirement also means that our algorithm has a high chance of stopping before all variables are part of the tree. At some point the remaining combinations of free variables will not have a gate yielding a higher or equal significance than the top gate, at which point no new gate will be added to the tree. If this does not happen our algorithm will stop as soon as all available variables are part of the fault tree. We provide a basic flowchart of our algorithm in Figure 7.

4 Experimentation

To illustrate the full operation cycle of our algorithm we take a single fault from our input data provided by Intergas. In this example we will be examining the variable *ch_pressure*, which reports the water pressure on a boiler. Depending on the type of boiler, a healthy pressure should be between 1 and 1.5 bar. As a safety measure, the boilers produced by Intergas have a mechanism in place to check whether the pressure drops below 0.49 bar. If this happens, the boiler is put into lockdown and a notification is sent with the message *Warning low ch_pressure: 0.49*. This notification is what we considered the error in this example. When this notification is sent, all operations of the boiler are halted, and a user either needs to refill the boiler manually or call technical support.

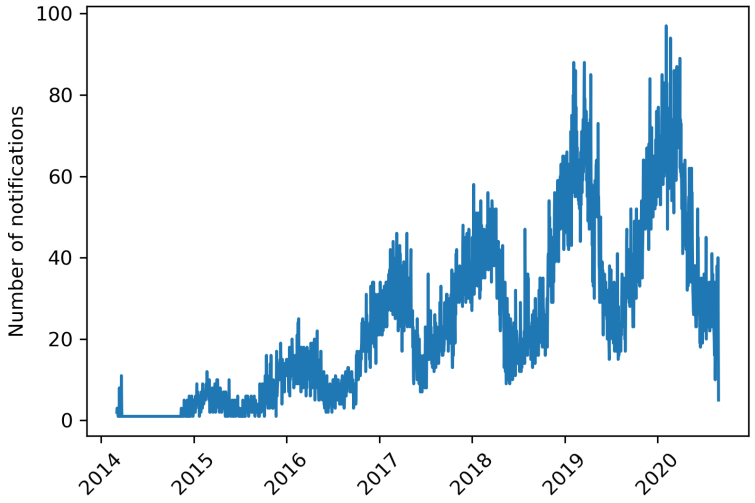

Figure 8: Low *ch_pressure* warning notifications per day for the Intergas dataset

To make a distinction between normal data and faulty data we have labelled the data collected on the day of the low pressure notification faulty, and the data of the day before it as normal. In total we have access to 30686 datapoints with faulty data that were triggered by a channel pressure below 0.49. This results in an average of 23 low pressure notification per day. A distribution of the notifications is shown in Figure 8. In this figure we also observe a pattern where more notifications are sent in months with lower outside temperatures on average in the Netherlands. This is easily explained by the fact that people tend to use their boilers more often and intensively to keep their homes warm during the winter. The increase of notifications throughout the years is explained by the increasing number of connected heaters. Combining this data with data of the days before a notification was sent leads us to a total of 103270 entries regarding the channel pressure dropping below 0.49. We use these datapoints as input for generating our tree.

Before we can reason about events and gates we need to translate our input dataset to Boolean values, which we do using our implementation of the C4.5 algorithm as described in section 3.1. To keep this example reasonable we only consider the parameter max. Since our dataset is large with over 100.000 entries, applying the C4.5 algorithm to convert our continuous variables to Boolean takes around 6 minutes on our test machine. See section 4.2 for further analysis of computational time and time complexity.

Variable	Split Value	Variable Description
$ch\_pressure$	0.49	Water pressure (bar)
$bc\_tapflow$	0	Water usage (l/min)
$heatertemp$	64.58	Boiler temperature (°C)
$s1\_temp$	4.31	Supply water temperature (°C)

Table 4: C4.5 Algorithm splits for selected variables

The C4.5 algorithm applied splits on the variables as given in table 4. We see that it split the $ch\_pressure$ variables on a value of 0.49, which agrees with the business process we know about. For the sake of this example we limited ourselves to the 4 variables given in the table, which are the 4 variables highest up in the eventual fault tree. The next step is generating a fault tree with the highest significance. To do this all gate possibilities are exhaustively checked. After one iteration of gate checking it is determined that the gate $ch\_pressure$ AND $bc\_tapflow$ yields the highest significance. Hence, this will be the top gate, and our fault tree looks like Figure 9. Note that we use the error *Warning low ch_pressure: 0.49* as the top event.

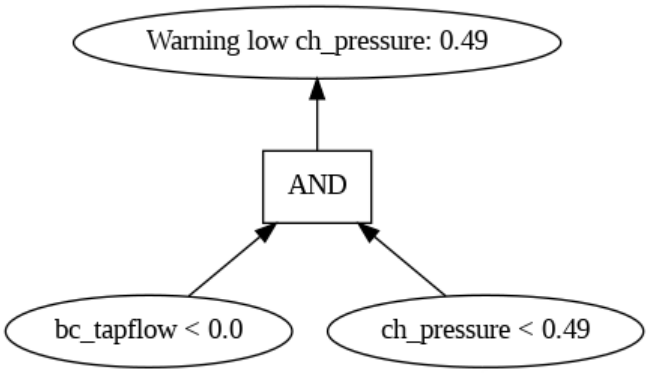

Figure 9: Fault tree for *Warning low ch_pressure: 0.49* after first gate is constructed

We can see in Figure 9 that we have a top event with the warning of low pressure, which is connected via an AND gate with $bc\_tapflow$ and $ch\_pressure$. We learn from this that the chances of a low pressure warning occurring are maximized if both the flow rate is 0, and the channel pressure is below or equal to 0.49. The

ch_pressure is expected to be present in the gate, since the system error is about the channel pressure. We can also explain the presence of the *bc_tapflow* variable in the gate because if the *ch_pressure* drops below a certain point the boiler will stop operation and no water will be used. Letting our algorithm continue generating a new layer results in Figure 10. Here a new layer connected to the channel pressure variable is constructed. We learn from this that the probability of the channel pressure getting below 0.49 is maximized if either the temperature of the heater drops below 64.58 degrees, or the temperature of the supply water drops below 4.31. The relation between these variables in the gate is less obvious, but we know that there is at least a significant association between these events and the *ch_pressure* dropping below 0.49. The algorithm continues generating these layers until no gate is found that contains unused variables and is significant enough to be accepted. We can also annotate the basic events with probabilities after the algorithm halts. We do this by simply dividing the number of datapoints where the basic event is active by the total count of datapoints for each basic event in the generated tree. For this example the complete tree including probabilities is found in Appendix B, Figure 33.

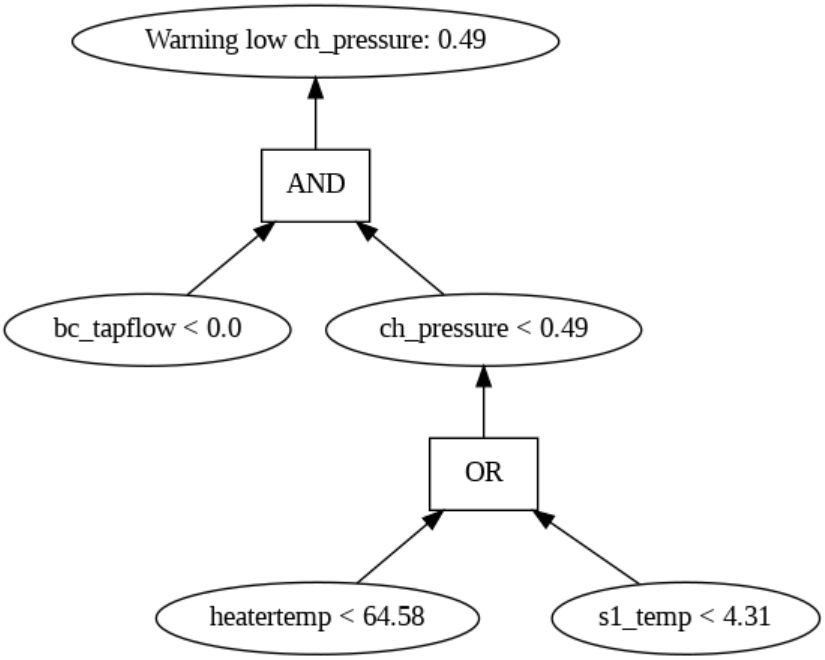

Figure 10: Fault tree for *Warning low ch_pressure: 0.49* after first two gates are constructed

4.1 Results for Intergas datasets

In our research we used the datasets provided by Intergas as a mean to test our algorithm on real-world data. We considered 11 different errors contained in the data (Table 2). Even though more errors are known to Intergas, not all of them have a

sufficient amount of data available to perform meaningful operations on. For each of these 11 errors we ran our algorithm and constructed 4 fault trees by taking the $min, max, average$ and $range$ of each of the samples in the input data, resulting in 44 fault trees. Even though the $range$ is not part of the provided dataset, it is easily calculated by subtracting min from max. We provide some of these fault trees in Appendix A, with their corresponding errors, significance and selected value for each variable. Afterwards we compared the significance of the 4 different fault trees per error to return the most significant tree. We got a range of results which we would like to discuss.

Tree Size The sizes of the trees greatly depended on the parameter and error chosen. We provide Figure 11 for a plot of all of the tree sizes. Here we defined the size of the tree as the number of layers. Some constructed fault trees only contained a single gate below the top event, others contained up to 6 layers. In some real-world scenarios fault trees can get up to thousands of nodes, but given the limited number of variables in our case study this was of course not expected for our research. In reality the layers further down the tree might be of less use for domain experts since they have less impact on the top event. This is because they are separated by other gates that need to activate as well before reaching the top event.

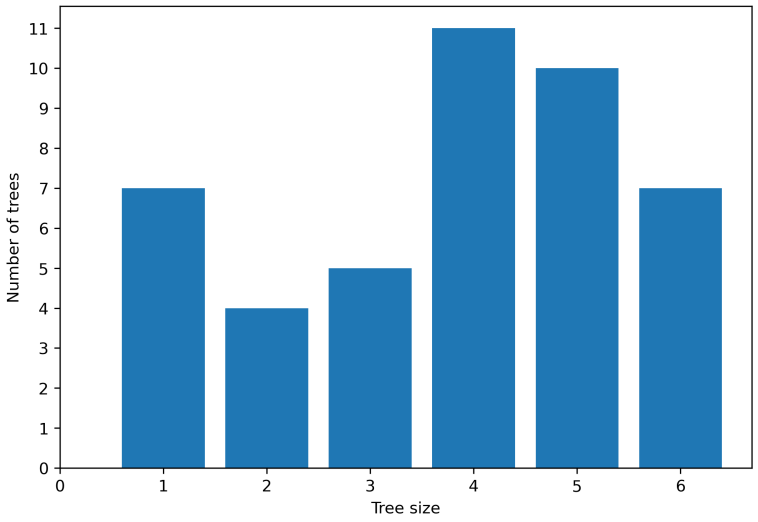

Figure 11: Count of constructed tree sizes (number of layers) for the Intergas dataset.

Significance Our constructed fault trees almost cover the entire range of significance, with trees having a significance value as low as 0.00, and trees having a significance as high as 0.95. This range of significance is expected when using real-world datasets. Even though our algorithm returns the best fault tree, it does not guarantee that a valuable fault tree exists for a given problem. It is possible that variables having great influence on a top event are simply not available in the dataset. The best trees for each of the errors and their corresponding significance are given in Figure 12. We see

that for the errors concerning low values of *t1* and *t2* we were able to produce trees with a very high significance. Most other trees fall within a range of a significance of 0.18 and 0.4.

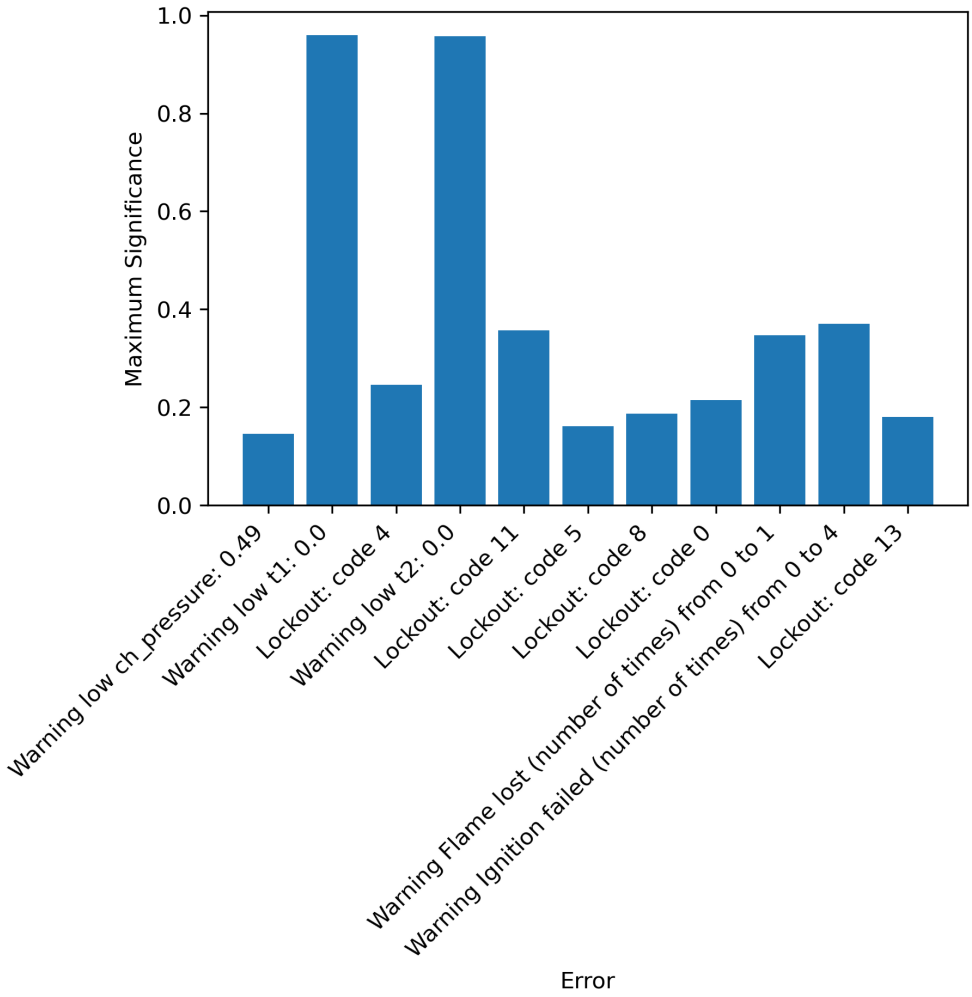

Figure 12: Maximum significance per error.

Value There is certainly value in our approach for industry. The results of our experiments on the Intergas dataset show that significant trees can be constructed. This does not mean that users should simply take the results for granted, there is still some manual evaluation needed. First of all the significance value should be considered, and if this is below a desired threshold the tree does probably not include useful insights. Trees with a high significance however do contain meaningful relations, as was confirmed by Intergas domain experts for the constructed trees. In cases where domain experts already had good understanding of how an error occurs, they confirmed that important variables were also in the tree. In our case study we were not able to provide Intergas with novel insights regarding faults in their products

through our automated fault trees. We believe this is not caused by an incorrect or insufficient implementation, but rather by the nature of the data and the large amount of domain knowledge already available at Intergas.

4.2 Time complexity

The time complexity of our algorithm is divisible in two parts. First the C4.5 algorithm, which often makes up the majority of the runtime, and secondly the FT construction algorithm, which we implemented using the phi coefficient.

We run our experiments in a Docker container with access to an AMD Ryzen 3900 and 128GB of RAM. The C4.5 algorithm has a time complexity of $O(m \cdot n^2)$, where m is the number of rows and n the number of variables. Applying the C4.5 algorithm on our entire dataset (100.000 records, 28 variables) took roughly 5 minutes. Thresholds are learnt for one variable at a time, which makes this process parallelisable in theory. Our algorithm does not implement this yet, so this remains a great opportunity for future work to improve upon.

On average, the computation time of a single run of our fault tree generation algorithm is faster than the translation step from continuous to Boolean data using the C4.5 algorithm. In Figure 13 we plot the runtime of sample datasets with different dimensions. We can see that increasing the dimensions of the data sometimes shortens the runtime. This is because our algorithm picks the tree with the highest significance. If for example a first layer with a high significance is constructed, chances are that further down the construction process no layers with an significance at least as high are found. This means that the algorithm can halt early, thus lowering the runtime.

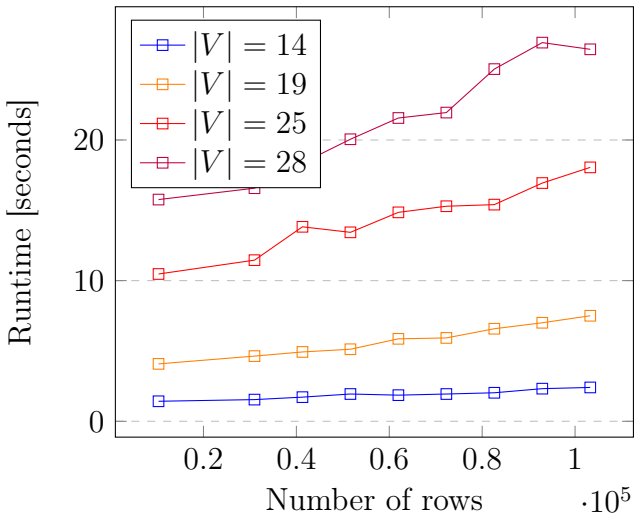

Figure 13: Runtime of FT algorithm for different data sizes using phi coefficient

5 Conclusions

In this research we presented an approach to automatically generate fault trees from continuous-valued data sets. We were able to use a real-world data set provided by Intergas and construct multiple fault trees. Our method differs from previous approaches such as the LIFT algorithm by allowing continuous data to be part of the input dataset. We introduce the usage of the C4.5 algorithm to perform Boolean splits in the continuous data before we automatically construct fault trees. We use the phi coefficient to determine the significance of each fault tree and to find the most suitable gates, which differs from the implementation of PAMH used in the LIFT algorithm.

5.1 Usability

Due to the nature of the Intergas dataset and the exhaustive approach of our algorithm both fault trees with a high and low significance were constructed. Fault trees with a low significance are probably best to be skipped, but the trees with high significance did encode the real-world problems that Intergas domain experts know about. This does not mean that we were able to give Intergas new insights in the causes of failures in their domestic boilers, but we were able to confirm some causes they already knew through domain experience from a data perspective. This leads us to believe that this approach is also suitable in other industries as well, if the provided dataset follows the requirements we described in this paper.

5.2 Limitations

To get the best result from our algorithm it is important that the input dataset contains minimum noise and incorrect entries. This preprocessing of the dataset is not part of our algorithm and is different for each dataset. This means that some manual work is required and domain experts are still needed before our algorithm can help. For our dataset we manually omitted a few boilers that were causing an unrealistic number of errors, since there was probably something fundamentally wrong with them. Another limitation comes from the implementation of the C4.5 algorithm. Even though execution times in the order of minutes is not problematic, for even larger datasets this can quickly grow out of hand. In contrast to the authors of the LIFT algorithm our approach also completely ignores the concept of causality. While this might not lead to drastic changes in the produced fault trees for the Intergas dataset, there exists datasets where causal relations are of more importance and simply skipping these relations could lead to incorrect fault trees.

5.3 Future work

Our suggestions for future work come directly from the limitations of our own work. First of all we think implementing the C4.5 algorithm can be parallelized. This

would mean that even larger datasets can be used as input in our algorithm without drastically increasing the execution time. A second direction for future work would be reimplementing causal checks to the algorithm whilst remaining scalable for large datasets. This would be very interesting for datasets containing lots of causal relations and could perhaps increase the accuracy of generated fault trees.

References

[1] Bin Gao and Yuehua Cui. Learning directed acyclic graphical structures with genetical genomics data. *Bioinformatics*, 31(24):3953–3960, 09 2015. ISSN 1367-4803. doi: 10.1093/bioinformatics/btv513. URL https://doi.org/10.1093/bioinformatics/btv513.

[2] Ruocheng Guo, Lu Cheng, Jundong Li, P. Richard Hahn, and Huan Liu. A survey of learning causality with data: Problems and methods. *ACM Comput. Surv.*, 53(4), July 2020. ISSN 0360-0300. doi: 10.1145/3397269. URL https://doi.org/10.1145/3397269.

[3] Intergas installation manual Combi-Compact-HRE. URL https://www.intergasheating.co.uk/app/uploads/2018/04/Combi-Compact-HRE-Installation-manual-88287806.pdf.

[4] Sohag Kabir. An overview of fault tree analysis and its application in model based dependability analysis. *Expert Systems with Applications*, 77:114 – 135, 2017. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2017.01.058. URL http://www.sciencedirect.com/science/article/pii/S0957417417300714.

[5] John T. Kent. Information gain and a general measure of correlation. *Biometrika*, 70(1), 1983. doi: 10.1093/biomet/70.1.163.

[6] Thuc Duy Le, Lin Liu, Anna Tsykin, Gregory J. Goodall, Bing Liu, Bing-Yu Sun, and Jiuyong Li. Inferring microRNA–mRNA causal regulatory relationships from expression data. *Bioinformatics*, 29(6):765–771, 01 2013. ISSN 1367-4803. doi: 10.1093/bioinformatics/btt048. URL https://doi.org/10.1093/bioinformatics/btt048.

[7] Jiuyong Li, Saisai Ma, Thuc Le, Lin Liu, and Jixue Liu. Causal Decision Trees. *IEEE Transactions on Knowledge and Data Engineering*, 29(2):257–271, February 2017. ISSN 1041-4347. doi: 10.1109/TKDE.2016.2619350. URL http://ieeexplore.ieee.org/document/7600471/.

[8] Alexis Linard, Doina Bucur, and Mariëlle Stoelinga. Fault trees from data: Efficient learning with an evolutionary algorithm. *Dependable Software Engineering. Theories, Tools, and Applications Lecture Notes in Computer Science*, 2019. doi: 10.1007/978-3-030-35540-1_2.

[9] Alexis Linard, Marcos L. P. Bueno, Doina Bucur, and Marielle Stoelinga. Induction of fault trees through bayesian networks. *Proceedings of the 29th European Safety and Reliability Conference (ESREL)*, 2019. doi: 10.3850/978-981-11-2724-3_0596-cd.

[10] N. Mantel and W. Haenszel. Statistical aspects of the analysis of data from retrospective studies of disease. *J. Natl. Cancer Inst.*, 22(4):719–748, Apr 1959.

[11] Meike Nauta, Doina Bucur, and Mariëlle Stoelinga. Lift: Learning fault trees from observational data. In Annabelle McIver and Andras Horvath, editors, *Quantitative Evaluation of Systems*, Lecture Notes in Computer Science. Springer, 2018. ISBN 978-3-319-99153-5.

[12] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986. doi: 10.1007/bf00116251.

[13] Vineet K. Raghu, Allen Poon, and Panayiotis V. Benos. Evaluation of causal structure learning methods on mixed data types. In Thuc Duy Le, Kun Zhang, Emre Kıcıman, Aapo Hyvärinen, and Lin Liu, editors, *Proceedings of 2018 ACM SIGKDD Workshop on Causal Disocvery*, volume 92 of *Proceedings of Machine Learning Research*, pages 48–65, London, UK, 20 Aug 2018. PMLR. URL `http://proceedings.mlr.press/v92/raghu18a.html`.

[14] Paul R. Rosenbaum and Donald B. Rubin. The central role of the propensity score in observational studies for causal effects. *Matched Sampling for Causal Effects*, page 41–55, 1983. doi: 10.1017/cbo9780511810725.016.

[15] Donald B. Rubin. Estimating Causal Effects from Large Data Sets Using Propensity Scores. *Annals of Internal Medicine*, 127(8_Part_2):757, October 1997. ISSN 0003-4819. doi: 10.7326/0003-4819-127-8_Part_2-199710151-00064. URL `http://annals.org/article.aspx?doi=10.7326/0003-4819-127-8_Part_2-199710151-00064`.

[16] Steven L. Salzberg. C4.5: Programs for machine learning by j. ross quinlan. morgan kaufmann publishers, inc., 1993. *Machine Learning*, 16(3):235–240, September 1994. doi: 10.1007/bf00993309. URL `https://doi.org/10.1007/bf00993309`.

[17] William E Vesely, Francine F Goldberg, Norman H Roberts, and David F Haasl. Fault tree handbook. Technical report, DTIC Document, 1981.

A PAMH gate construction example

To illustrate how the PAMH approach works we consider a hypothetical dataset X, which has already been converted to only contain Boolean variables. The contents of X are given by table 5. At the start of our fault tree construction we will consider all potential gates G and calculate their corresponding PAMH(g,e) score. We will select the gate with the highest score to be connected to the error e (labeled Error in table 5). We start with an empty fault tree, which means that all the gate possibilities with variables (A, B, C, D) are still potential candidates. Let's consider the potential gate g = A & B. Since our dataset contains 4 variables and our gate takes (A, B), we need to stratify with the remaining variables C and D. Doing this gives us the following strata as given in table 6.

A	B	C	D	Error	Count
1	1	0	0	1	10
1	1	0	0	0	1
1	1	0	1	1	3
1	1	0	1	0	1
1	1	1	0	1	10
0	1	0	0	1	2
0	0	0	0	0	5
0	0	0	1	1	2
1	0	0	1	0	4
0	1	1	0	1	7
0	1	1	0	0	5

Table 5: Example dataset X

(C, D) (0, 0)	$E=1$	$E=0$	*Total*
$A\&B=1$	10	1	11
$A\&B=0$	2	5	7
Total	12	6	18

(C, D) (0, 1)	$E=1$	$E=0$	*Total*
$A\&B=1$	3	1	4
$A\&B=0$	2	4	6
Total	5	5	10

(C, D) (1, 0)	$E=1$	$E=0$	*Total*
$A\&B=1$	10	0	10
$A\&B=0$	7	5	12
Total	17	5	22

Table 6: Strata for dataset D_1 and gate A & B

Because X contains no data for C=1 and D=1 we only have 3 strata. We can now calculate if the gate A & B is significant using the PAMH score:

$$PAMH = \frac{\left(\left| \ 10 - \frac{11*12}{18} + 3 - \frac{4*5}{10} + 10 - \frac{10*17}{22} \right| - \frac{1}{2}\right)^2}{\frac{11*7*12*6}{18^2*17} + \frac{4*6*5*5}{10^2*9} + \frac{10*12*17*5}{22^2*21}} \approx 11.05$$

If we take a α value of 0.05 for the χ^2 test we get a value of 3.84. Since $11.05 > 3.84$ we can safely say that the occurrence of A & B has a strong causal relationship with the occurrence of the error. If no other gates yield a higher PAMH score than 11.05, we will set the gate A AND B as the first gate in our fault tree.

B Constructed fault trees

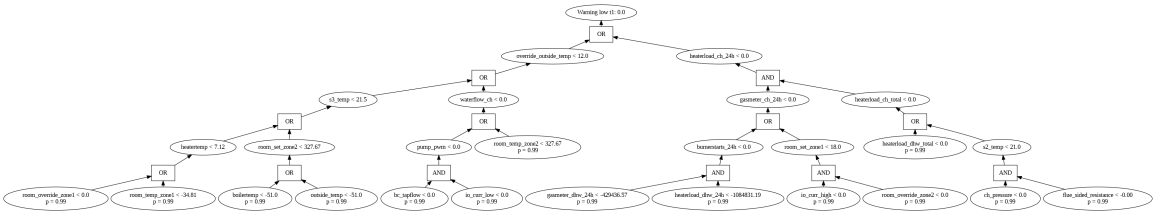

Figure 14: Fault tree for *Warning low t1*, with a significance of 0.0 and *min* selected as value for each variable.

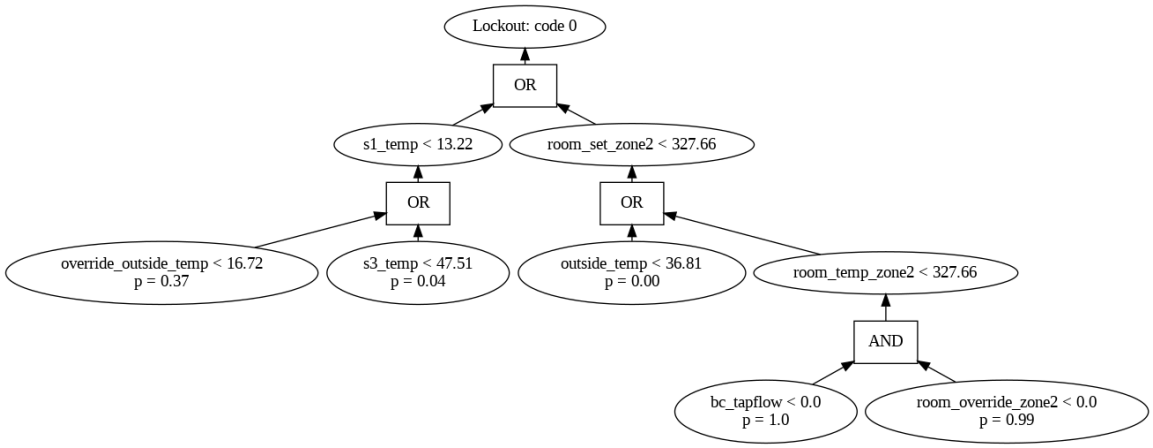

Figure 15: Fault tree for *Lockout code 0*, with a significance of 0.02 and *average* selected as value for each variable.

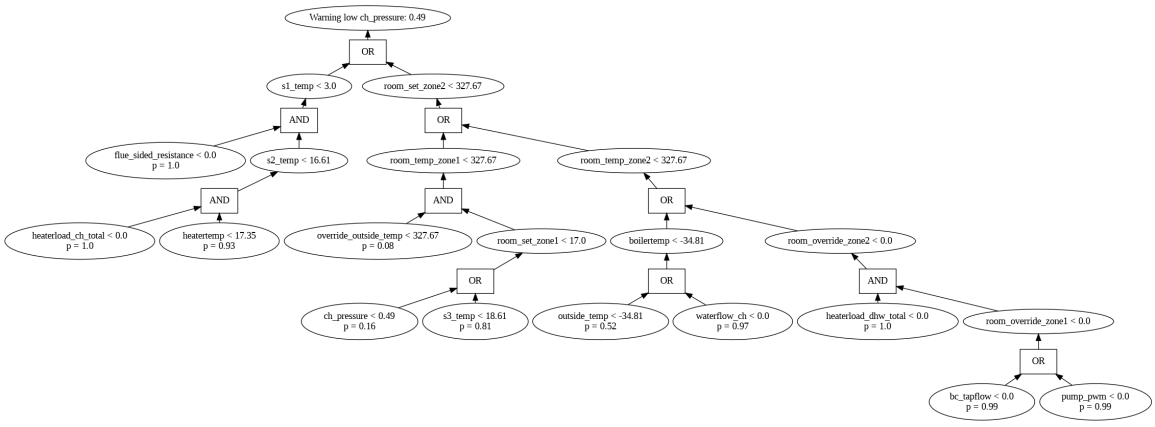

Figure 16: Fault tree for *Warning low ch_pressure: 0.49*, with a significance of 0.02 and *min* selected as value for each variable.

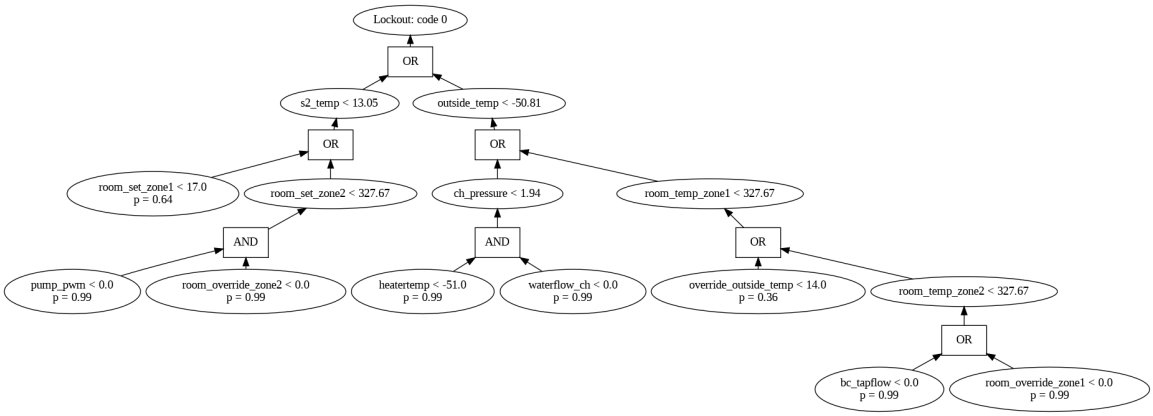

Figure 17: Fault tree for *Lockout code 0*, with a significance of 0.03 and *min* selected as value for each variable.

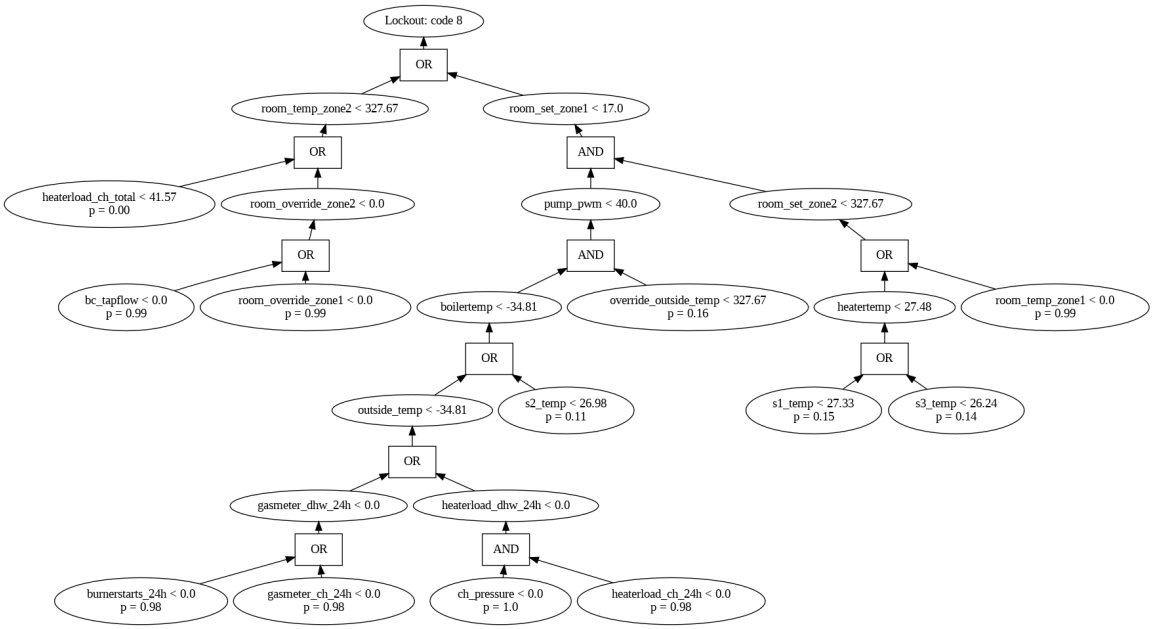

Figure 18: Fault tree for *Lockout code 8*, with a significance of 0.03 and *min* selected as value for each variable.

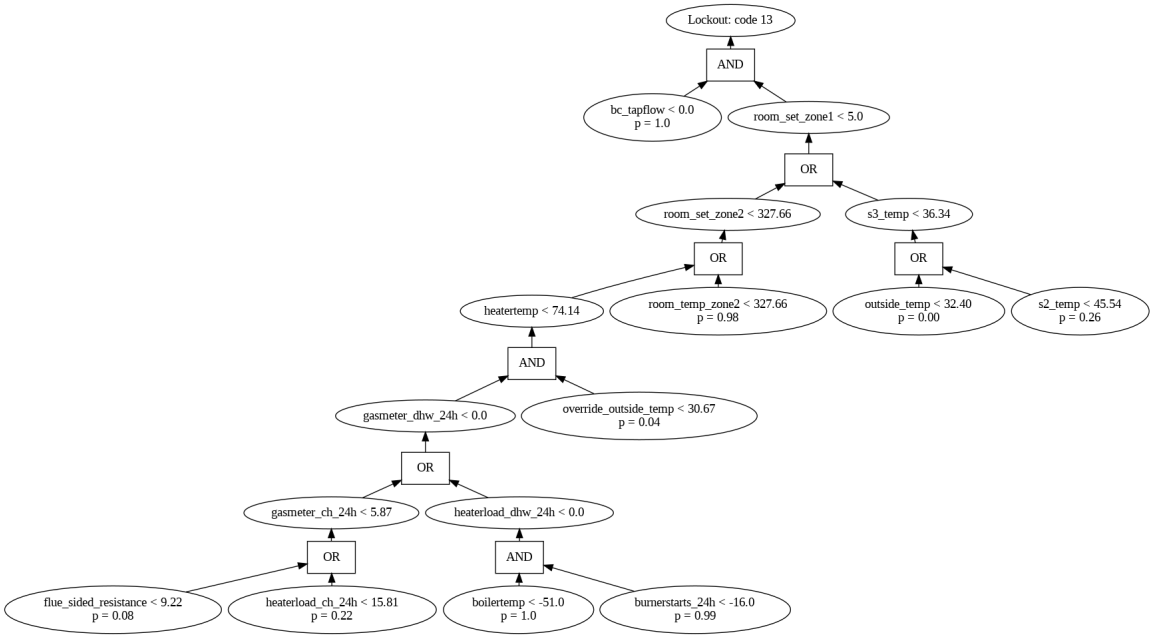

Figure 19: Fault tree for *Lockout code 13*, with a significance of 0.05 and *average* selected as value for each variable.

Figure 20: Fault tree for *Lockout code 13*, with a significance of 0.08 and *min* selected as value for each variable.

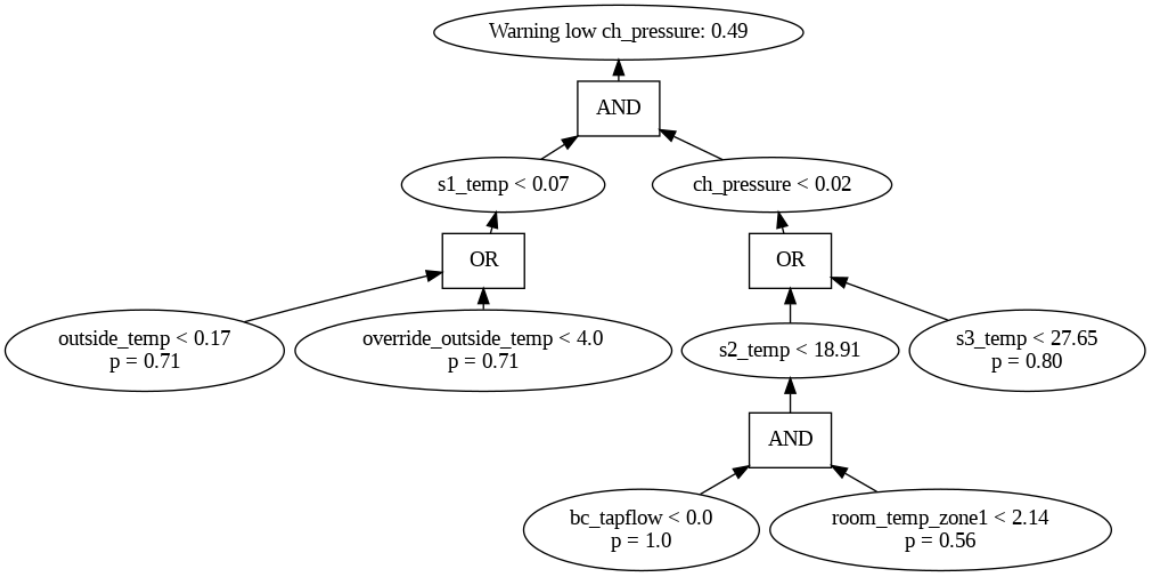

Figure 21: Fault tree for *Warning low ch_pressure: 0.49*, with a significance of 0.05 and *range* selected as value for each variable.

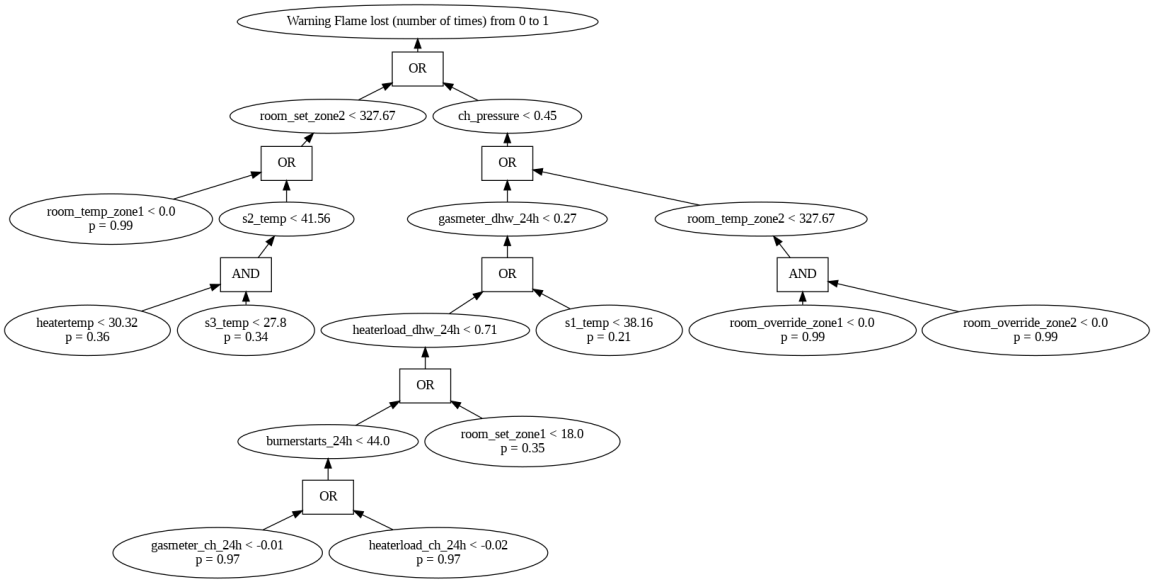

Figure 22: Fault tree for *Warning Flame lost (number of times) from 0 to 1*, with a significance of 0.05 and *min* selected as value for each variable.

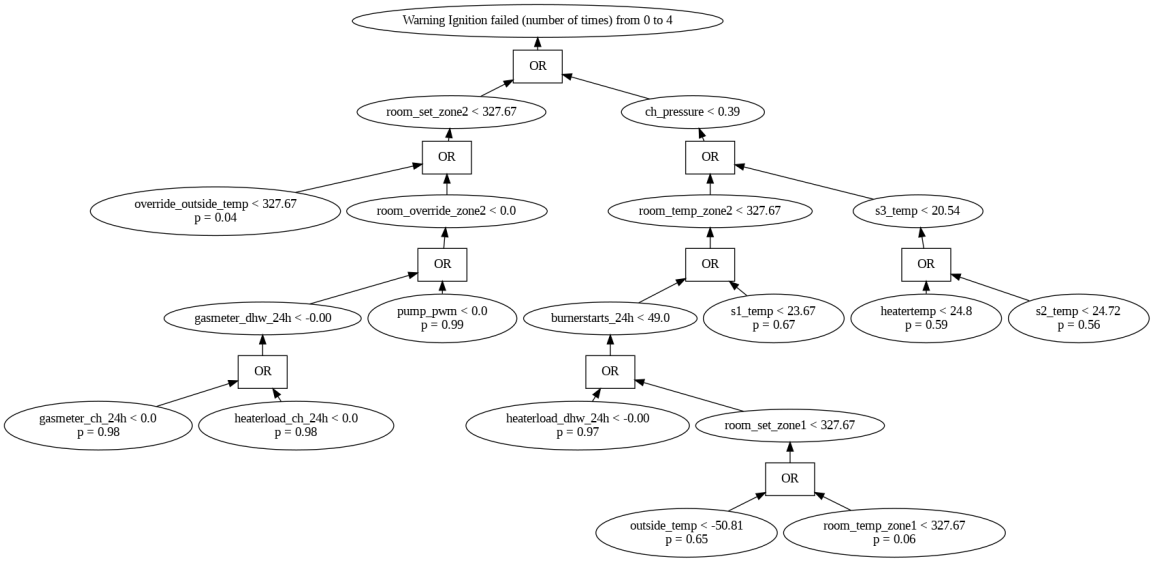

Figure 23: Fault tree for *Warning Flame ignition failed (number of times) from 0 to 4*, with a significance of 0.06 and *min* selected as value for each variable.

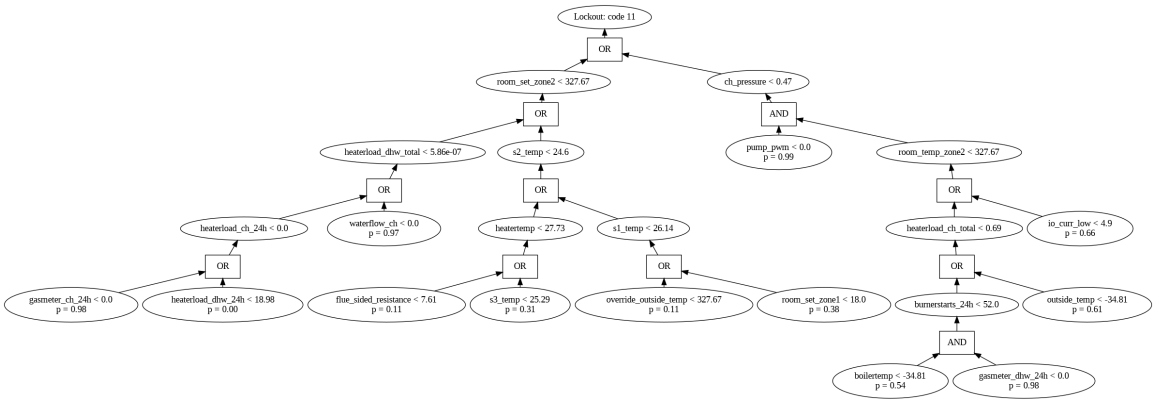

Figure 24: Fault tree for *Lockout code 11*, with a significance of 0.07 and *min* selected as value for each variable.

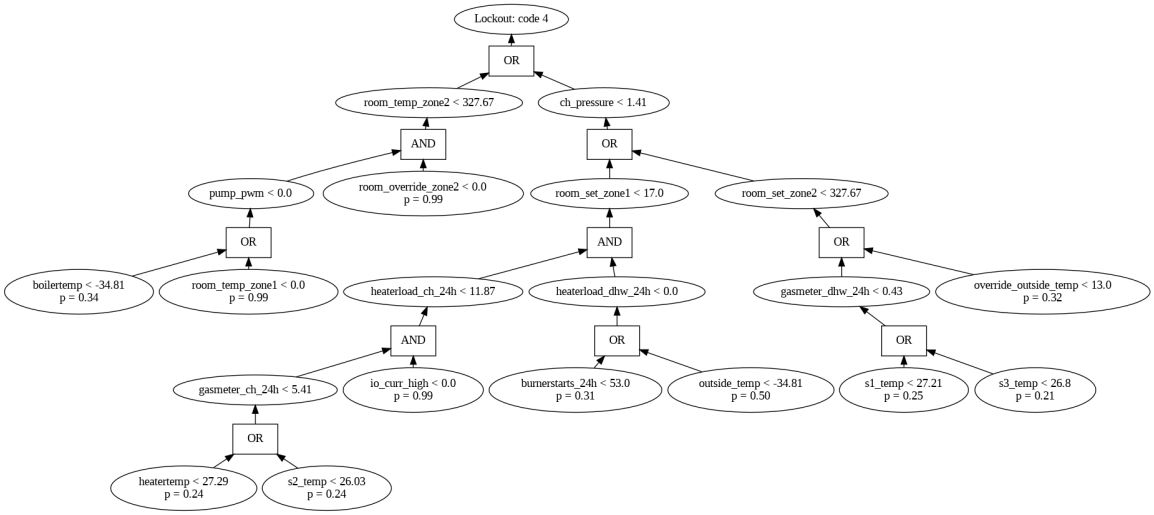

Figure 25: Fault tree for *Lockout code 4*, with a significance of 0.07 and *min* selected as value for each variable.

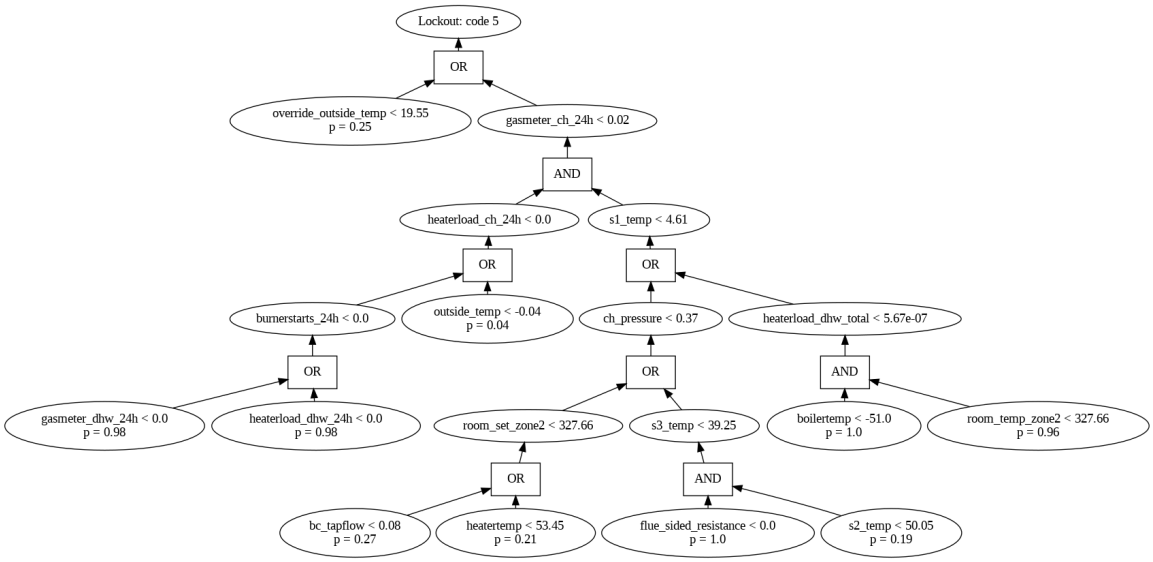

Figure 26: Fault tree for *Lockout code 5*, with a significance of 0.16 and *average* selected as value for each variable.

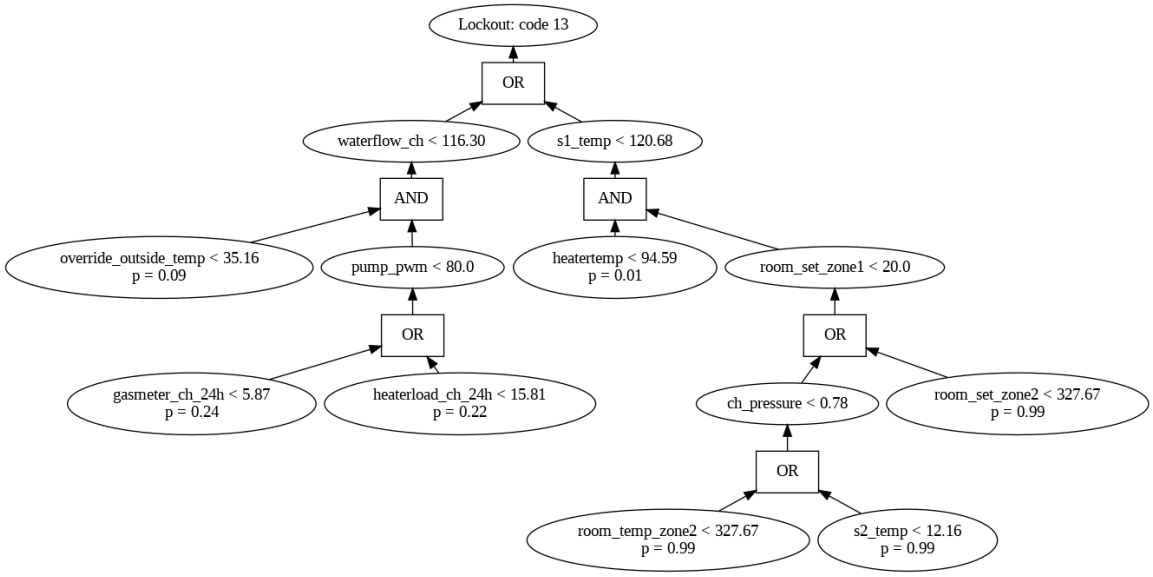

Figure 27: Fault tree for *Lockout code 13*, with a significance of 0.08 and *max* selected as value for each variable.

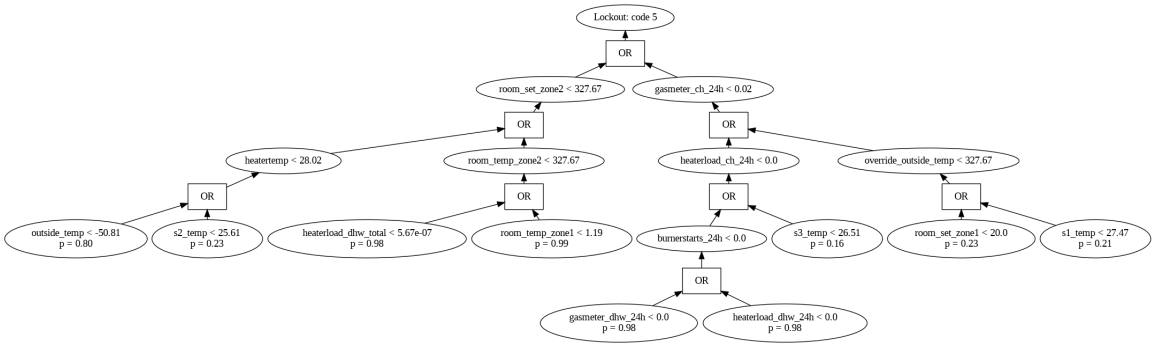

Figure 28: Fault tree for *Lockout code 5*, with a significance of 0.08 and *min* selected as value for each variable.

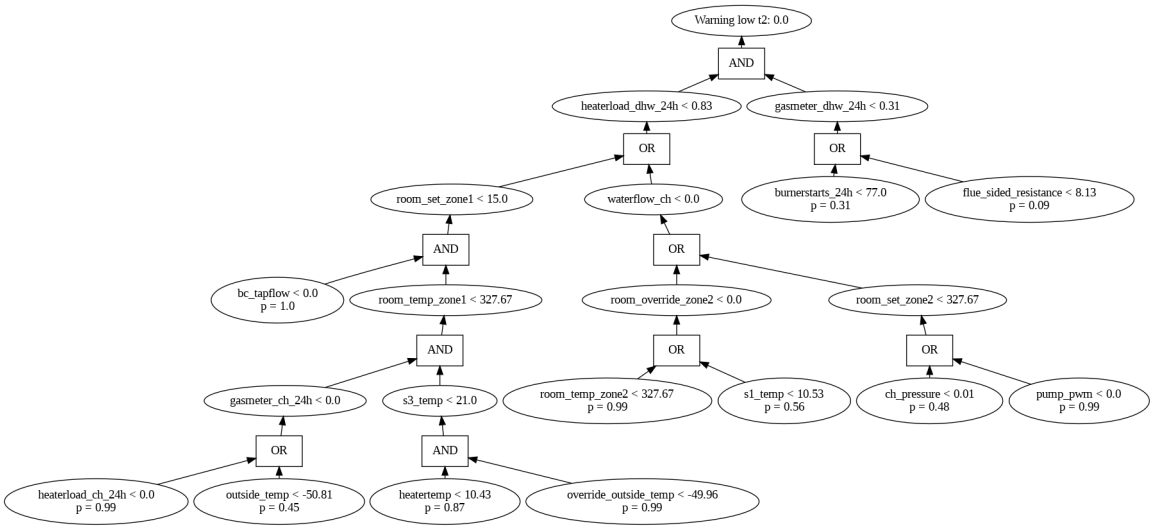

Figure 29: Fault tree for *Warning low t2*, with a significance of 0.08 and *min* selected as value for each variable.

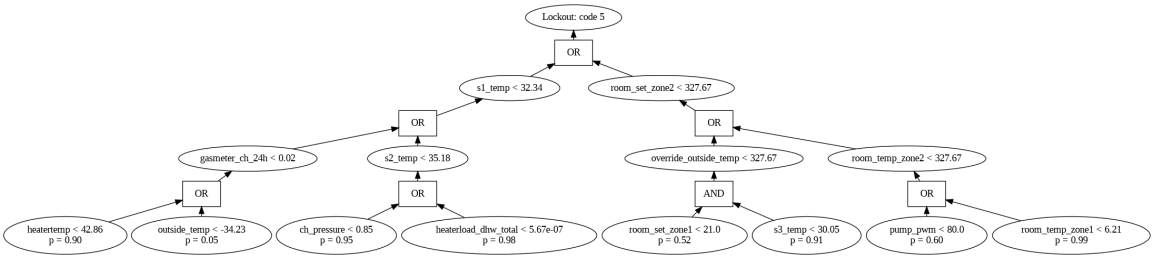

Figure 30: Fault tree for *Lockout code 5*, with a significance of 0.08 and *max* selected as value for each variable.

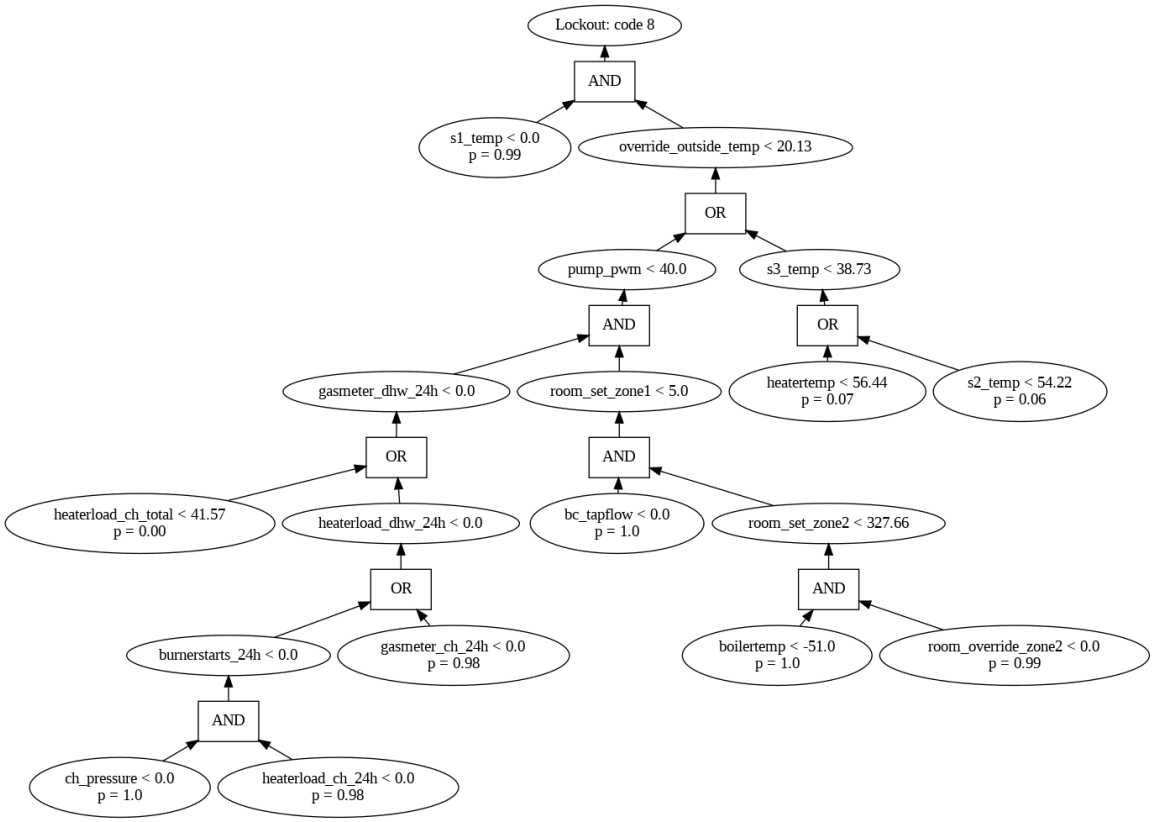

Figure 31: Fault tree for *Lockout code 8*, with a significance of 0.09 and *average* selected as value for each variable.

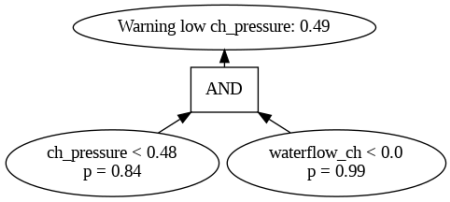

Figure 32: Fault tree for *Warning low ch_pressure: 0.49*, with a significance of 0.12 and *average* selected as value for each variable.

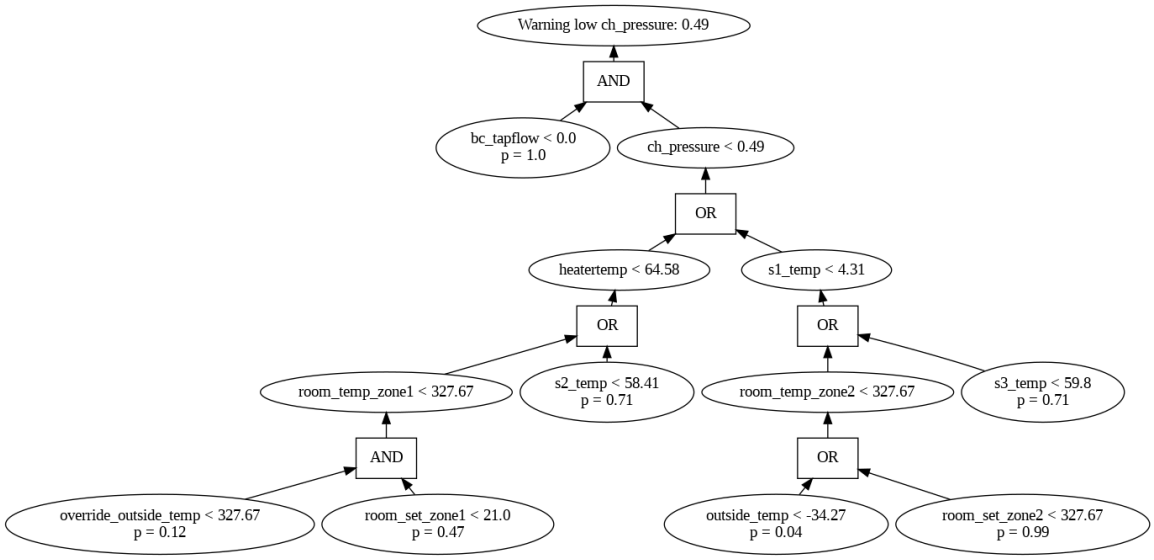

Figure 33: Fault tree for *Warning low ch_pressure: 0.49*, with a significance of 0.14 and *max* selected as value for each variable.

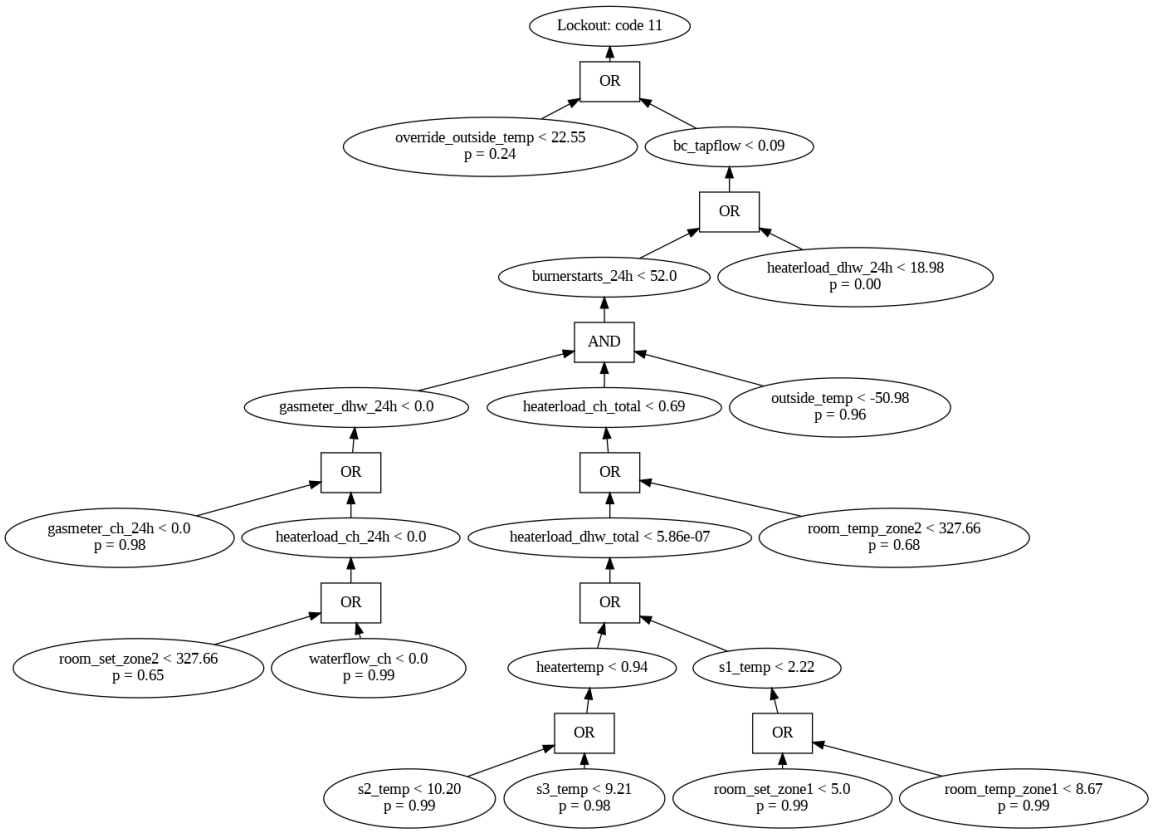

Figure 34: Fault tree for *Lockout code 11*, with a significance of 0.16 and *average* selected as value for each variable.

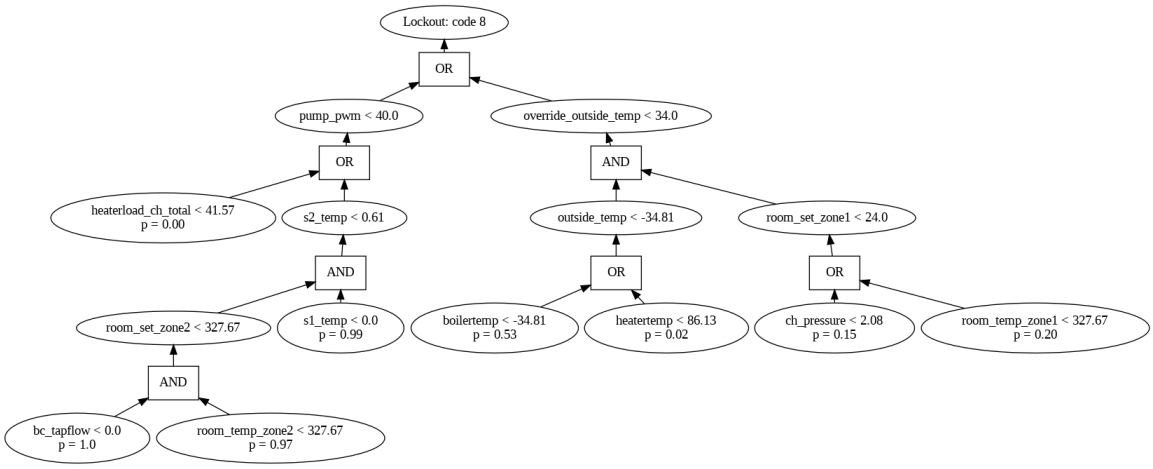

Figure 35: Fault tree for *Lockout code 8*, with a significance of 0.16 and *max* selected as value for each variable.

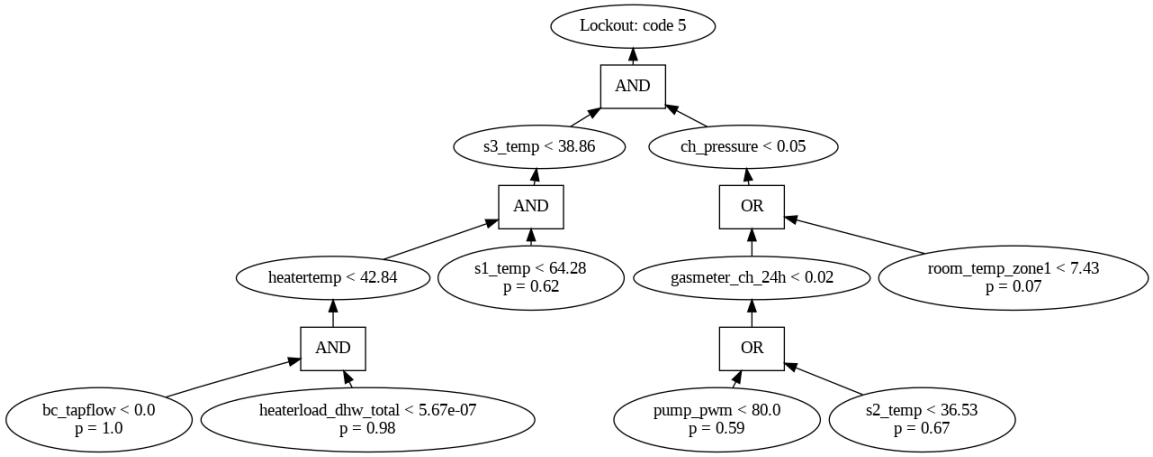

Figure 36: Fault tree for *Lockout code 5*, with a significance of 0.20 and *range* selected as value for each variable.

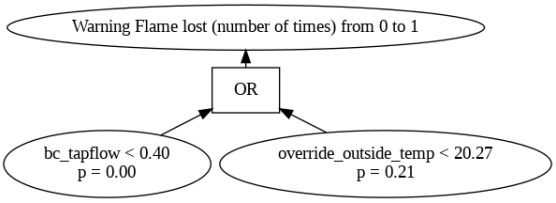

Figure 37: Fault tree for *Warning Flame lost (number of times) from 0 to 1*, with a significance of 0.17 and *average* selected as value for each variable.

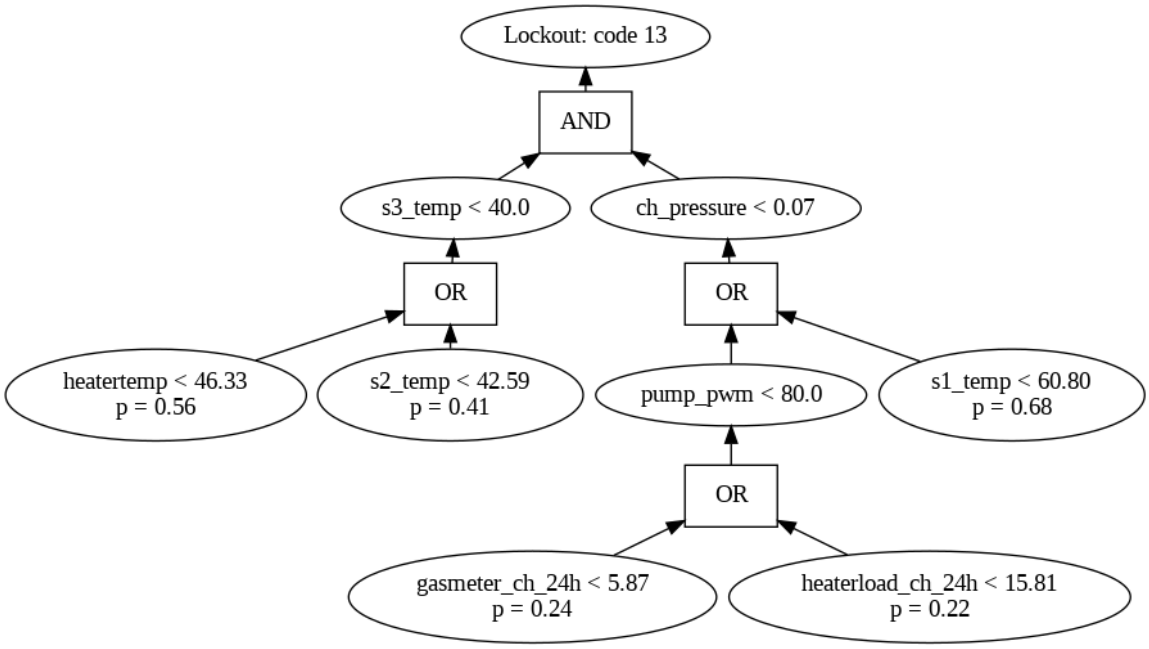

Figure 38: Fault tree for *Lockout code 13*, with a significance of 0.17 and *range* selected as value for each variable.

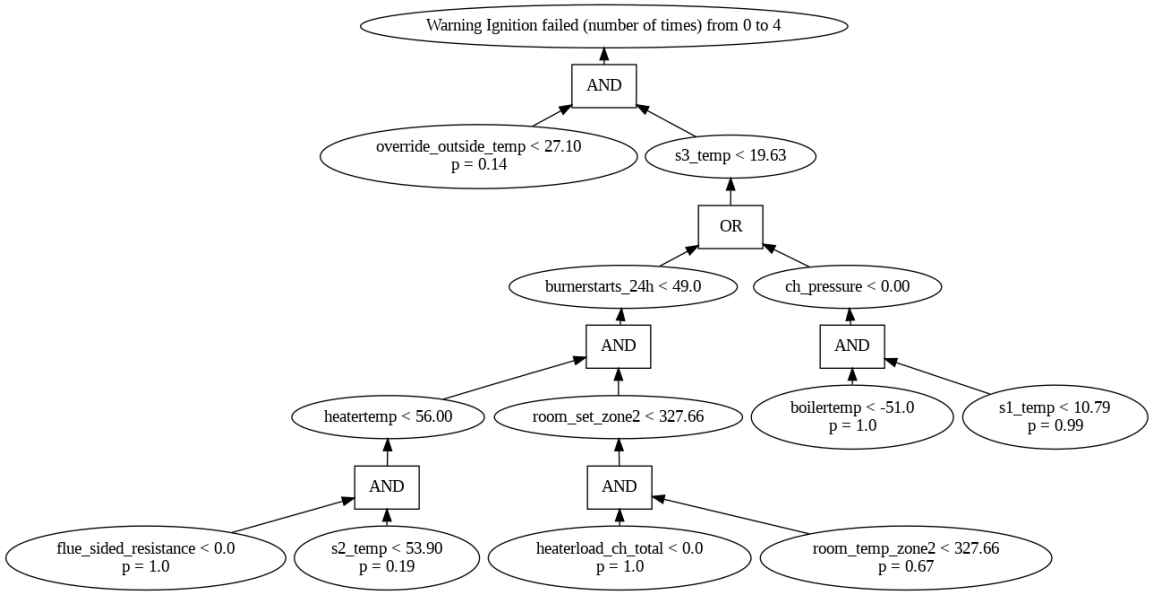

Figure 39: Fault tree for *Warning ignition failed (number of times) from 0 to 4*, with a significance of 0.18 and *average* selected as value for each variable.

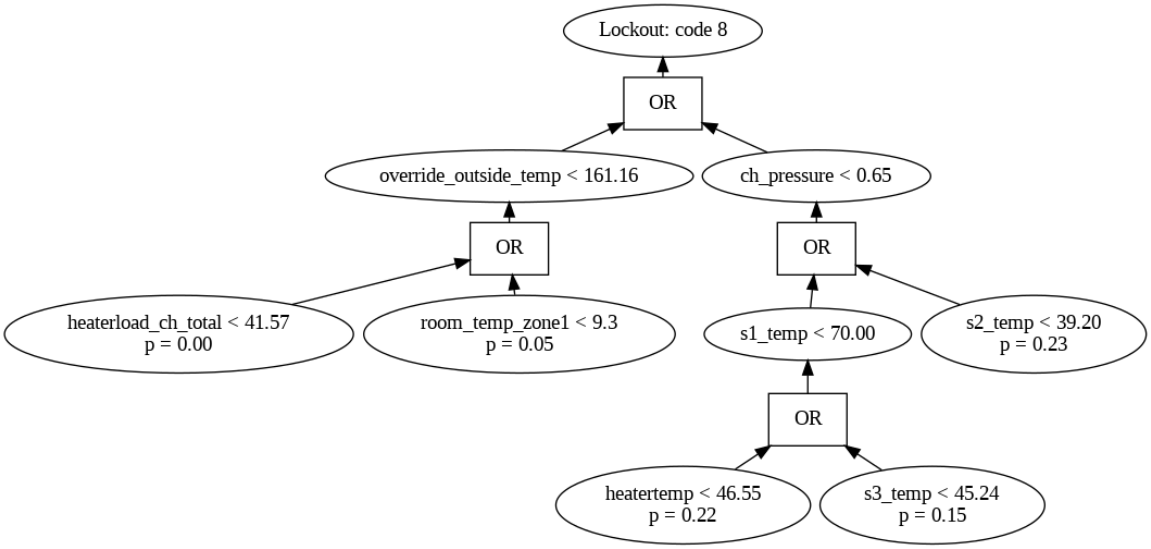

Figure 40: Fault tree for *Lockout code 8*, with a significance of 0.18 and *range* selected as value for each variable.

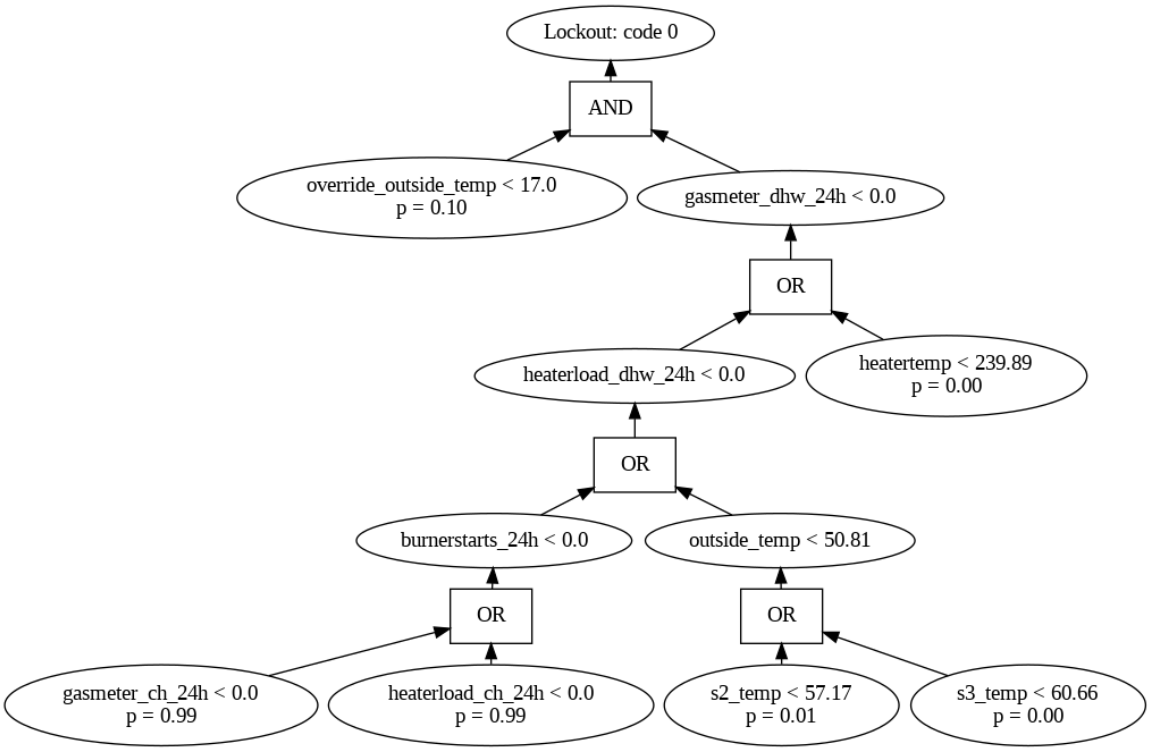

Figure 41: Fault tree for *Lockout code 0*, with a significance of 0.19 and *range* selected as value for each variable.

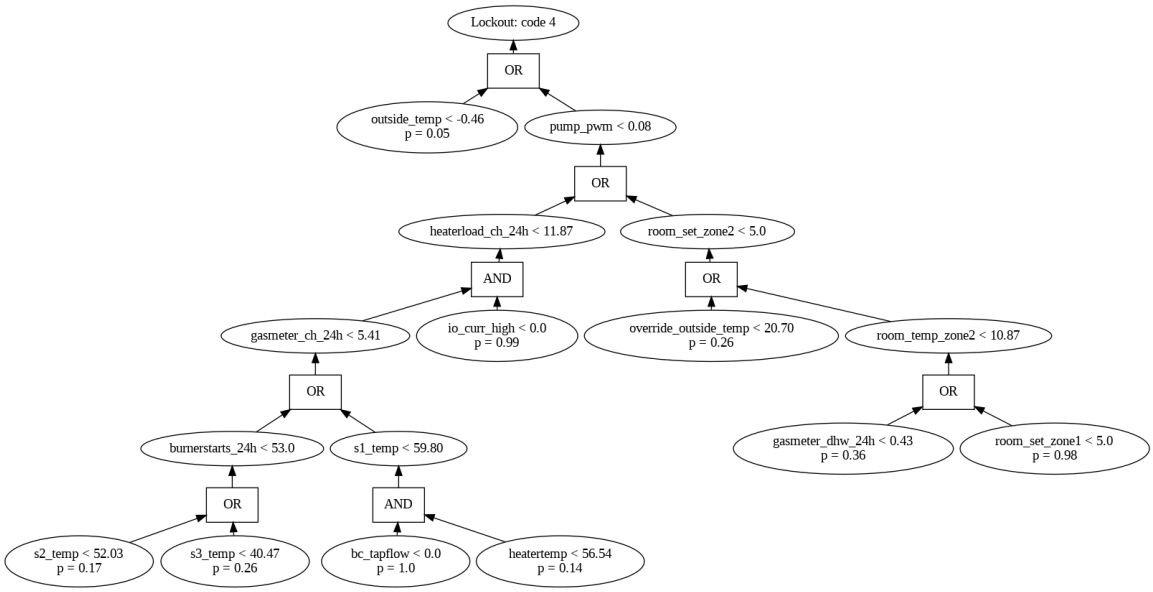

Figure 42: Fault tree for *Lockout code 4*, with a significance of 0.20 and *average* selected as value for each variable.

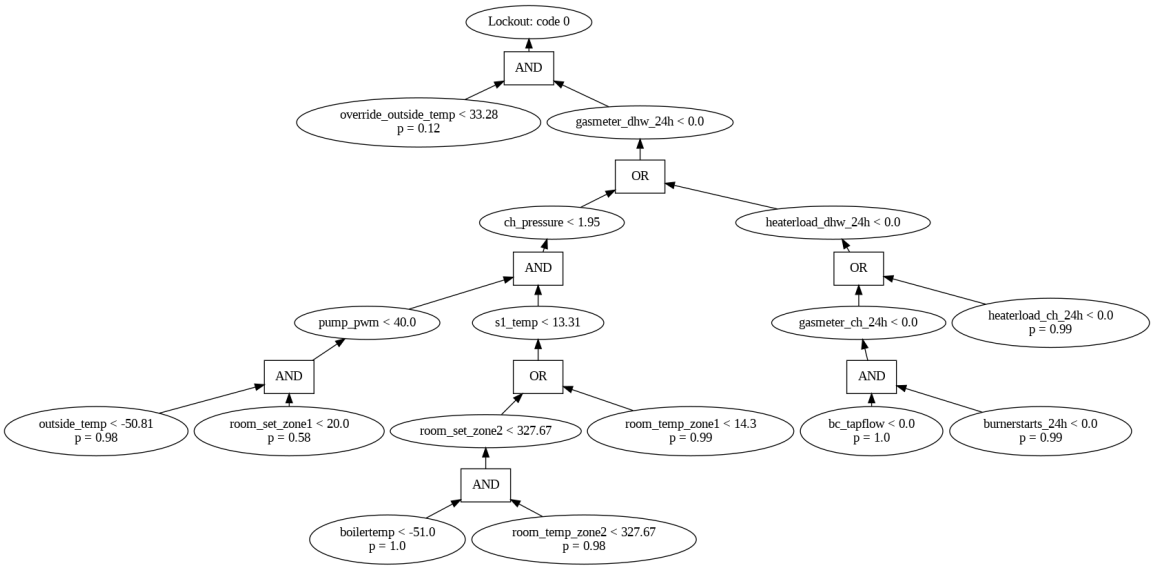

Figure 43: Fault tree for *Lockout code 0*, with a significance of 0.21 and *max* selected as value for each variable.

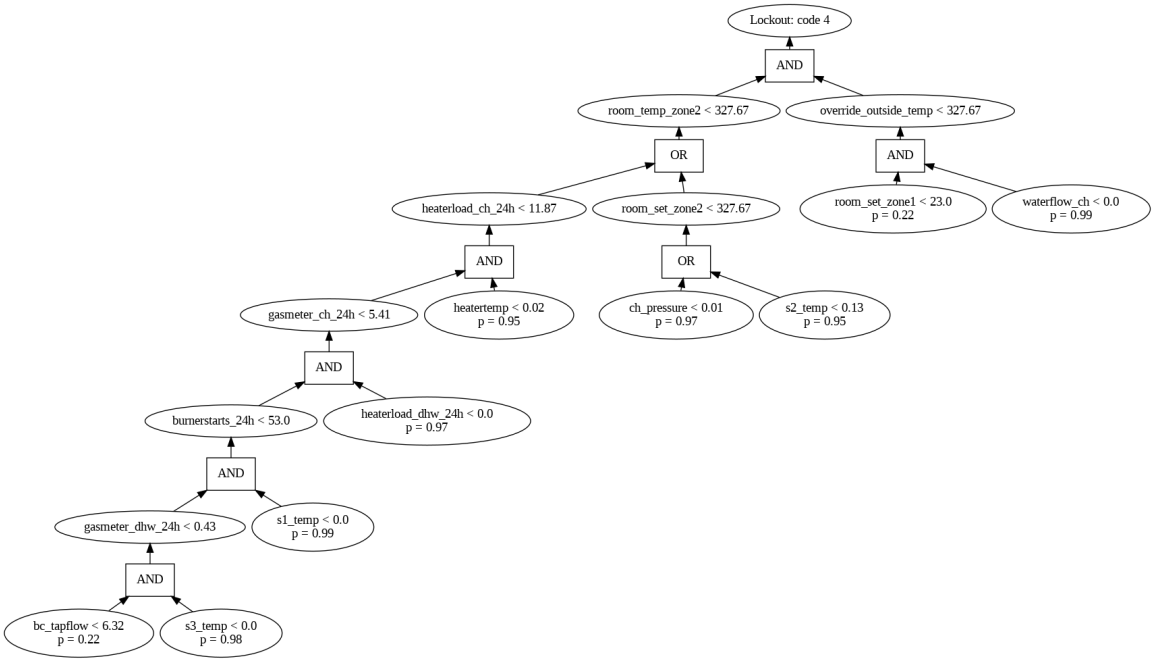

Figure 44: Fault tree for *Lockout code 4*, with a significance of 0.22 and *max* selected as value for each variable.

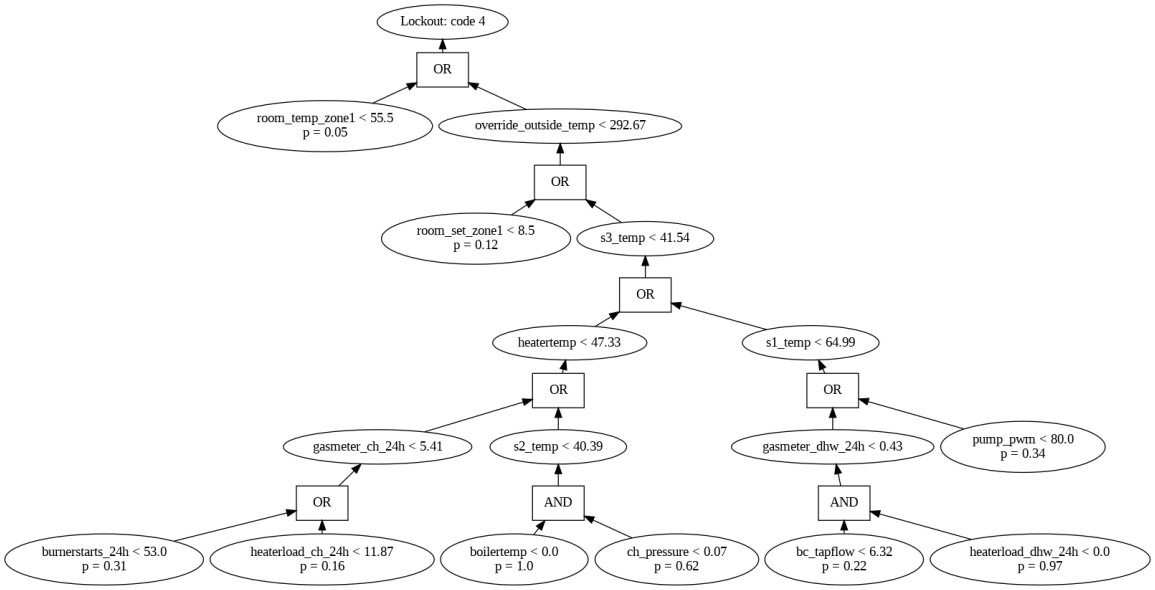

Figure 45: Fault tree for *Lockout code 4*, with a significance of 0.24 and *range* selected as value for each variable.

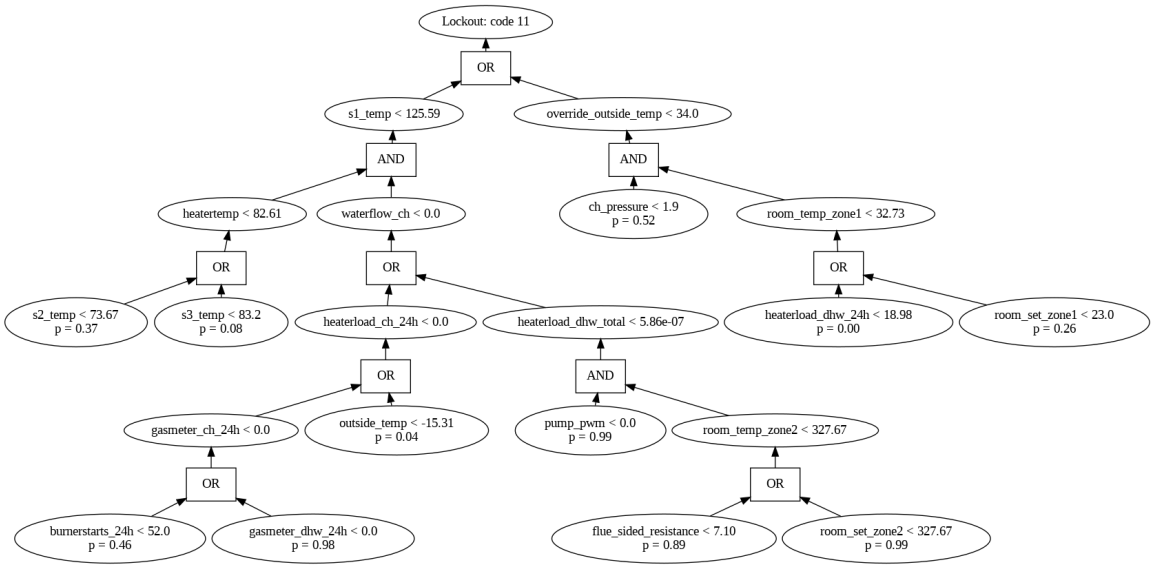

Figure 46: Fault tree for *Lockout code 11*, with a significance of 0.28 and *max* selected as value for each variable.

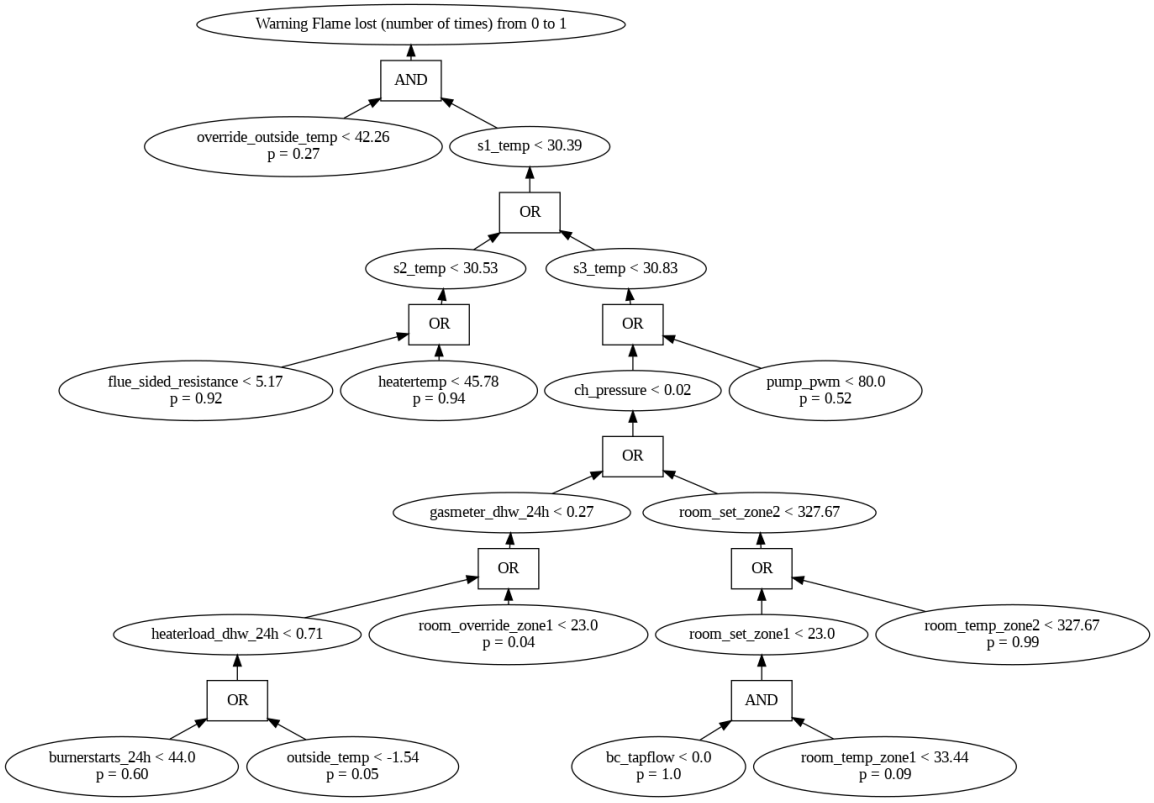

Figure 47: Fault tree for *Warning Flame lost (number of times) from 0 to 1*, with a significance of 0.32 and *max* selected as value for each variable.

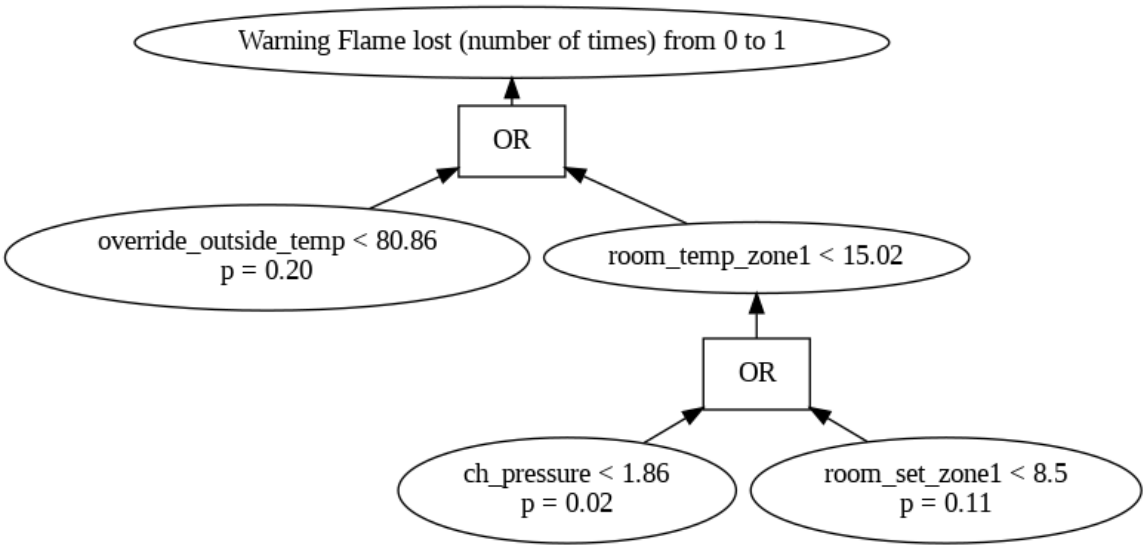

Figure 48: Fault tree for *Warning Flame lost (number of times) from 0 to 1*, with a significance of 0.34 and *range* selected as value for each variable.

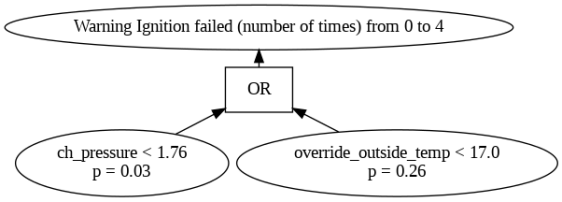

Figure 49: Fault tree for *Warning ignition failed (number of times) from 0 to 4*, with a significance of 0.34 and *range* selected as value for each variable.

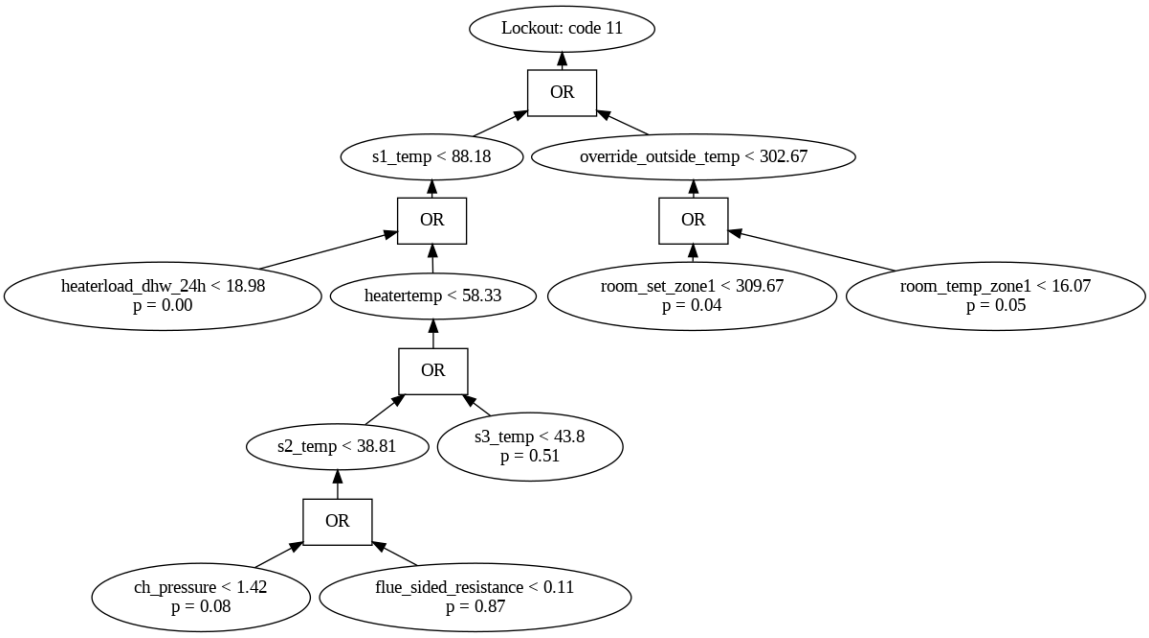

Figure 50: Fault tree for *Lockout code 11*, with a significance of 0.35 and *range* selected as value for each variable.

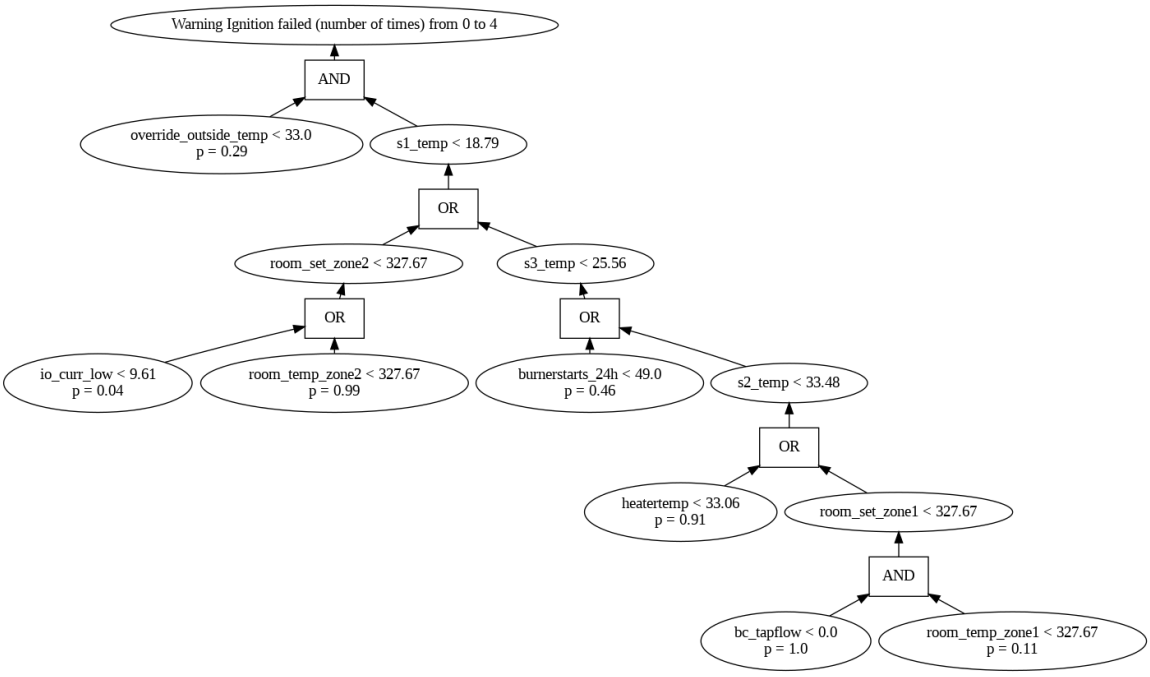

Figure 51: Fault tree for *Warning ignition failed (number of times) from 0 to 4*, with a significance of 0.36 and *max* selected as value for each variable.

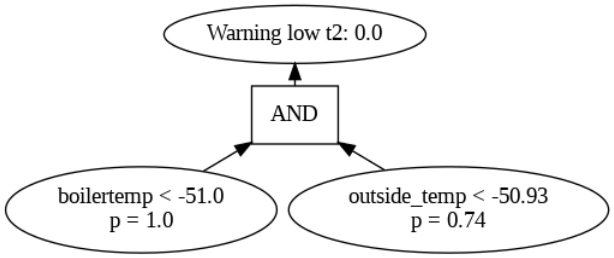

Figure 52: Fault tree for *Warning low t2*, with a significance of 0.59 and *average* selected as value for each variable.

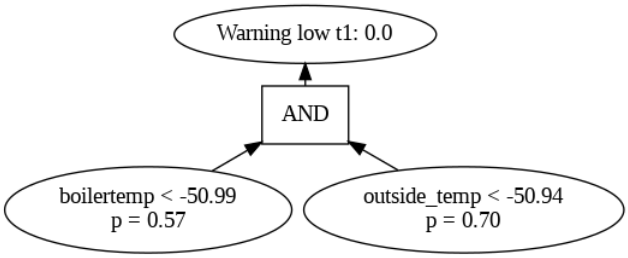

Figure 53: Fault tree for *Warning low t1*, with a significance of 0.87 and *average* selected as value for each variable.

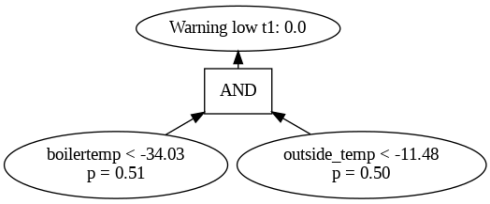

Figure 54: Fault tree for *Warning low t1*, with a significance of 0.95 and *max* selected as value for each variable.

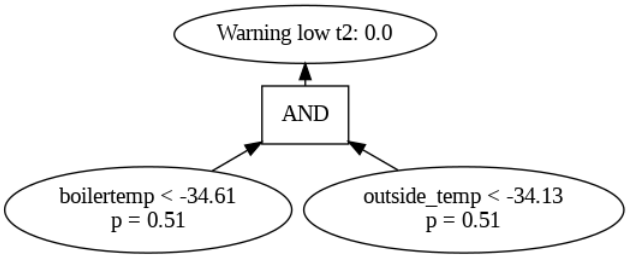

Figure 55: Fault tree for *Warning low t2*, with a significance of 0.95 and *max* selected as value for each variable.

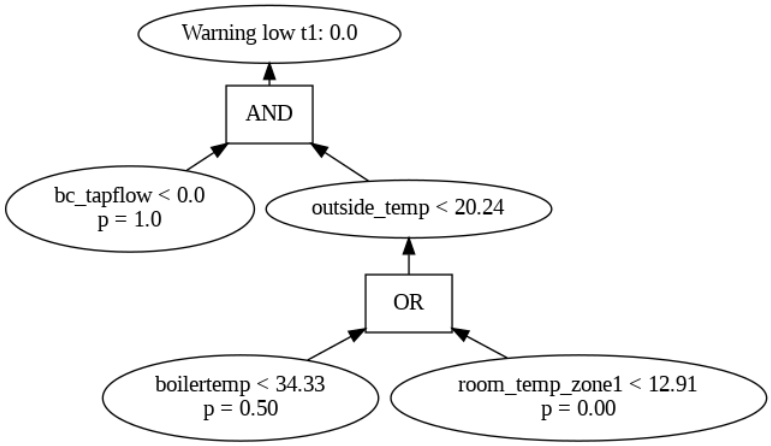

Figure 56: Fault tree for *Warning low t1*, with a significance of 0.95 and *range* selected as value for each variable.

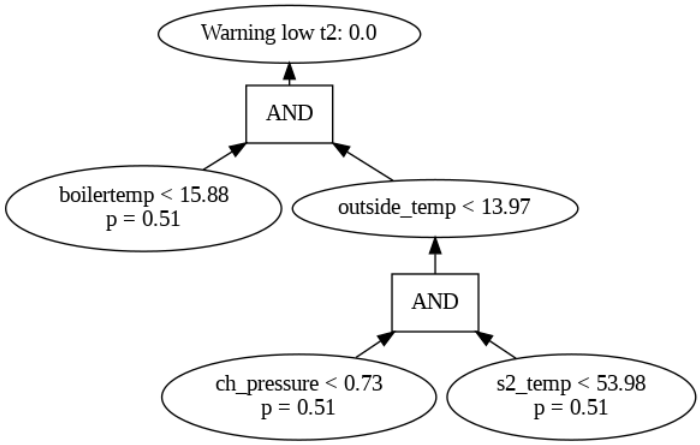

Figure 57: Fault tree for *Warning low t2*, with a significance of 0.95 and *range* selected as value for each variable.