PAPER Special Section on Formal Approaches

Evaluation the Redundancy of the IoT System Based on Individual Sensing Probability

Ryuichi TAKAHASHI[†a)], ***Member***

SUMMARY In IoT systems, data acquired by many sensors are required. However, since sensor operation depends on the actual environment, it is important to ensure sensor redundancy to improve system reliability in IoT systems. To evaluate the safety of the system, it is important to estimate the achievement probability of the function based on the sensing probability. In this research, we proposed a method to automatically generate a PRISM model from the sensor configuration of the target system and calculate and verify the function achievement probability in the assumed environment. By designing and evaluating iteratively until the target achievement probability is reached, the reliability of the system can be estimated at the initial design phase. This method reduces the possibility that the lack of reliability will be found after implementation and the redesign accompanying it will occur.

key words: *IoT system, autonomous driving system, sensing probability, PRISM model checker*

1. Introduction

1.1 Safety Verification of IoT System

In recent years, research and development related to IoT have been actively promoted all over the world, and it is growing rapidly [1]. IoT system equips several sensors into any object in our environment. By acquiring various types of information, control that was not possible before will be possible. The autonomous driving system is a typical IoT system. It aims to realize advanced automatic control with data acquired from vehicles and road sensors. However, some fatal accidents also have occurred. An autonomous driving vehicle needs to acquire various information such as the vehicle ahead, the opposing vehicle, pedestrians, and location information by sensors, and determine the behavior according to it. Compared to conventional vehicles, a huge number of sensors are mounted on vehicles, and these sensors are directly connected to vehicles' behavior control. Therefore, the design complexity increases at an accelerated rate. On the other hand, it is difficult to solve with a heuristic approach based on the developer's experience. It is essential to manage and verify design information using engineering methods to ensure the safety of the IoT system.

Highly reliable data is required for the safe driving of autonomous vehicles. However, no matter how sophisticated the sensor, false detection due to noise or the like always occurs. For example, if data is acquired once a second by a sensor with a sensing probability of 99.999%, a data acquisition error or false detection may occur once every 28 hours. If this occurs in a sensor used for collision avoidance, it can lead to a serious accident. In particular, in autonomous driving, where false control affects human life, it is very important to understand the sensing probability of the sensor from the design stage and to evaluate the safety.

To guarantee the sensing probability, it is necessary to consider two aspects. One is to ensure redundancy by multiplexing. One solution is to replace sensors with higher performance and higher sensing probability to prevent accidents due to sensor misdetections. However, the cost rises, and no matter how expensive the sensor, 100% sensing probability cannot be guaranteed. Therefore, it is possible to improve overall safety by mounting multiple sensors of the same type and detecting only one of them. When two sensors with a sensing probability of 99.999% are mounted, the probability that the two sensors cannot acquire data simultaneously is $0.001\% \times 0.001\% = 10^{-10}$. This means that there will be one failure every 317 years. If the number of sensors is increased further, safety can be improved. Also, if the acquired data is noisy and unreliable, mounting an odd number of sensors and taking a majority vote can improve the reliability of the data. Thus, redundant configuration by sensor multiplexing is very important in realizing automatic operation. However, it is very difficult to determine the configuration to secure the target sensing probability/reliability. If there is only one type of sensor, it can be calculated easily using the above formula, but this calculation is much more difficult for autonomous vehicles because many types of sensors depend on each other.

Another way to guarantee the sensing probability is a sensor alternative/degenerate configuration. Some sensors are not suitable for specific environments and purposes. For example, obstacle recognition by an optical camera can determine not only the distance to the object but also the type and state of the object. However, it has the disadvantage of being greatly affected by the weather such as rain and fog. On the other hand, if it is an ultrasonic sensor, the distance to the object is less affected by the weather and can detect the distance to the obstacle. Based on these characteristics, safety can be ensured by switching multiple types of sensors according to the situation. For example, it is possible to avoid obstacles with a stereo camera in normal times and

Manuscript received October 8, 2019.
Manuscript revised February 6, 2020.
Manuscript publicized May 14, 2020.

[†]The author is with the Dep. of Computer and Information Sciences, College of Engineering, Ibaraki University, Hitachi-shi, 316–8511 Japan.
a) E-mail: ryuichi.takahashi.office@vc.ibaraki.ac.jp

DOI: 10.1587/transinf.2019FOP0001

switch to an ultrasonic sensor in the rain or when the camera sensor fails. Functions and safety may be degraded by changing the sensor used, but functions such as autonomous driving can continue to be provided.

1.2 Problem to Determine Sensor Configuration

If the sensor configuration described in Sect. 1.1 is used, if N sensors of a single type should be mounted, and at least one of them needs to be operated, the probability of function failure can be calculated by $1 - (1 - P)^N$. The P specifies the sensing probability of a sensor unit. However, to realize autonomous driving, it is necessary to combine functions of various granularities hierarchically and to use multiple types of sensors in combination to realize each function. In such a complex configuration, it is very costly to estimate any property satisfaction probabilities (ex. probability of achieving a specific function) based on each sensor's sensing probability. Because, in a system where various sensors and functions work in a complex manner, the constraints to be verified are various, and it is a great task to manually construct the calculation formulas for all the constraint verifications. Ideally, a tool that automatically calculates the achievement probability by specifying the state to be achieved, rather than constructing the calculation formula, is desired.

Also, when designing an alternative/degenerate configuration of sensors, it is important to understand the dependency between functions and sensors. Depending on the operating status of other sensors, it may be necessary to limit the function or substitute another function. In the IoT system where multiple types of sensors are complexly dependent, the availability of a single sensor may affect multiple functions, and it is very important to manage its traceability. We think IoT systems also have variability similar to the conventional system. For example, autonomous vehicles may have different grades depending on the price range. Even if the basic functions are the same, there are variations in the types of mounted sensors and degenerative functions. There is Software Product Line (SPL) as a method for engineering product variation engineering, but it cannot manage sensor/function substitution or degeneration. Therefore, there is a need for a method for engineering management of sensor configurations including alternative and degenerate configurations.

In the following sections, we will explain the details of the proposed method, mainly on the autonomous driving system. Section 3 shows an overview of our proposed method. Section 4 explains how to determine the sensor configuration with considering redundancy. Sections 5 and 6 explains a method to evaluate system safety. Section 7 shows a case study and Sect. 8 discusses the effectiveness of our method. Section 9 shows related works and Sect. 10 concludes this paper.

2. Overview of the Proposed Method

To solve the problem described in Sect. 1.2, this paper proposes a sensor configuration design and evaluation method for IoT systems. The contribution of this method is as follows.

1. Proposal of a model that can represent redundant sensor configurations based on SPL
2. Convert sensor configuration to the PRISM model and automatically evaluate sensing probability with a tool
3. Exhaustively evaluate the sensing probability according to the operating environment

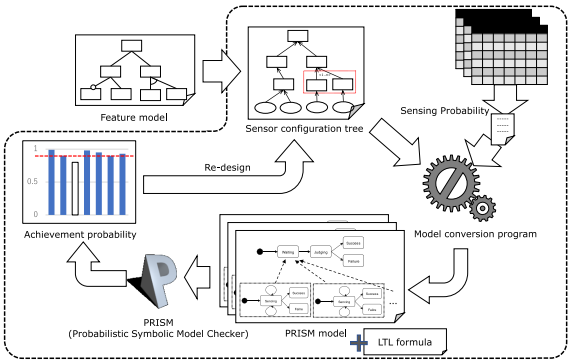

Fig. 1 Overview of proposed process

The first is to propose a model that structurally represents and manages the components of the IoT system. The model is based on the feature model used in SPL and expresses the sensor configuration by considering redundancy. In this model, function substitution/degeneration can also be regarded as a kind of product variability in SPL and systematically constructed and managed.

In the second, the above model is converted into a state transition diagram with probability. The converted model is input to the model checker PRISM for a state search. PRISM tool can evaluate not only reachability to the specific state, but also the probability to reach that state. The function achievement probability of the entire system can be calculated from the sensing probability of each sensor.

Third, we propose a method to comprehensively evaluate the function achievement probability of the system including the influence of environmental changes on the sensor. Since the sensor greatly depends on the actual environment, we comprehensively assume the operating environment of the system and evaluate whether the target achievement probability is satisfied in all target environments.

Figure 1 shows the overall process of the proposed method. This method first constructs a sensor configuration model based on the feature tree. After that, the sensor configuration model is converted to the PRISM model. In conversion, sensing probabilities are estimated from operating environment, and multiple PRISM models are acquired. The obtained PRISM model automatically evaluates the function achievement probability of the system by the tool. If the calculated probability does not reach the target level, review the sensor configuration and redesign it until the level is reached.

The proposal of this method makes it possible to en-

gineeringly manage the design of IoT systems considering redundancy. Furthermore, it becomes possible to evaluate the sensing probability quantitatively and automatically. As a result, design costs can be reduced and safety can be easily evaluated.

3. Sensor Configuration Tree

This section explains CBFM which is an existing feature model and proposes a sensor configuration tree based on CBFM.

3.1 Cardinary Based Feature Model

A feature model is a model used in the Software Product Line (SPL). SPL analyzes the products to model and manages the commonality of products and the variability that characterizes each product. The autonomous driving vehicles targeted in this research have common essential functions and various optional functions are prepared depending on the product grade. The feature model can manage these commonalities and variability.

After the feature model was proposed in Feature-Oriented Domain Analysis (FODA) in 1990 [2], several models have been proposed. In either model, the commonality and variability of the system are captured as features, and the relationship between features is represented hierarchically using a tree structure. In our research, Cardinary Based Feature Model (CBFM) [3] is used as a basic model to express the functions required for the system and the sensors required to execute them.

Figure 2 shows an example of CBFM feature model. Features are represented by rectangles on the model. There are three type features (mandatory, optional, and OR). These features are classified by analyzing the variability and commonality of product groups. Mandatory features are equipped in all products. Optional features are equipped with only some products. OR is a feature that is selected and equipped from a plurality of target features. Besides, two types of cardinality are defined as features of CBFM. First, group cardinality is the cardinality associated with the selection relationship and describes the number of minimum/maximum features are selected from the target child features. Another cardinality is feature cardinality. It associates with individual features. This cardinality describes the number of times the feature entity appears in the target system.

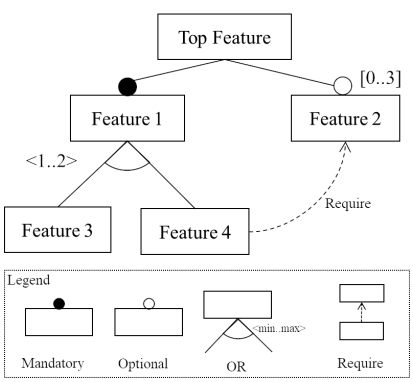

Fig. 2 Example of CBFM

These cardinalities are the reason why CBFM was adopted in this research. In sensor configuration design considering redundancy, alternative configurations and multiplexing configurations of sensors are important. By using the alternative feature and the group cardinality, it is possible to express an alternative configuration of the sensor to achieve a specific function. Also, by using feature cardinality, it is possible to represent multiplexing configuration for sensors. By using these two types of cardinality, the redundancy of the target system can be expressed as a kind of system variability.

Besides, we extend the "require" relationship into CBFM. The "require" expresses a direct dependency that selecting a feature requires that another feature be selected at the same time. It is especially used to express dependencies that traverse the hierarchical structure of feature models. In the autonomous driving system, it is considered that the operation of a specific sensor is indispensable for the realization of a certain function. The "require" is used to express such relationships.

3.2 Determing Sensor Configuration with Considering Redundancy

This section explains how to determine a sensor configuration of the target system that is developed from the feature model in Sect. 3.1. In the conventional feature model, the variability is analyzed, and after evaluating the requirements and costs, the features to be equipped in the system are determined. A decision table is often used for this determination, and feature configuration is reflected in the system configuration. In this research, we do not refer to the process of configuration determination, and only focus to define a model that represents the determined configuration. In the sensor (system) configuration determined by the conventional method, information such as optional and alternative nature is lost, and only a set of features equipped to the system can be obtained. However, the lost information is useful to manage system redundancy. In this research, we define a model that expresses the function and the sensor configuration by considering redundancy.

The feature model expresses the maximum possibility that the target system can take. Therefore, the elements determined as system specifications are a subset. Figure 3 shows determined sensor configuration based on the model of Fig. 2. Certain features require data acquired by sensors. Therefore, it is necessary to assign related sensors to the determined feature structure in consideration of redundancy. Specifically, edge features and a specific sensor model for realizing them are connected with relational lines. This assignment means not assigning individual sensors but assigning sensor types. Therefore, even if one type of sensor model is assigned, the product configuration may

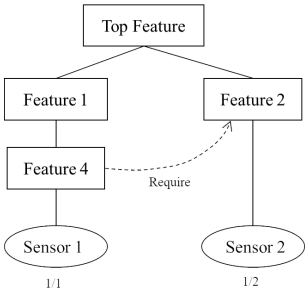

Fig. 3 Determing sensor configuration

include multiple sensors of the same product. This quantity relationship is expressed using the cardinality of the sensor. When sensors are multiplexed, not all sensors need to be in operation. For example, if we have three of the same sensors and two of them are operating normally, we can get the correct information by majority vote and determine the correct function to activate the function. Such sensor multiplexing information is added to the sensor model in the form of M/N. In the case of the previous example, the multiplexing information is $2/3$. The relationship between the sensor multiplexing information M/N and the feature cardinality $[f_{min}..f_{max}]$ is as follows.

$$f_{min} \leq M \leq N \leq f_{max} \tag{1}$$

Based on this multiplexed information, the sensing probability is calculated for each sensor type in the next section.

Alternative configurations for the sensor are determined using alternative features. When we determine an alternative configuration of features, we need to determine the number of features X to be equipped in the range of group cardinality $< g_{min}..g_{max} >$. However, like the sensor, not all X features need to be active. For example, multiple methods such as optical cameras and ultrasonic radars are used to detect obstacles in autonomous driving. If one or more of these different methods operate normally when obstacles can be detected, the redundant information is expressed in the form of a supplementary relationship. The relationship between the redundancy information X and the group cardinality is as follows.

$$g_{min} \leq X \leq g_{max} \tag{2}$$

This redundant information will be used to integrate the sensing probabilities in the next section.

4. Evaluation of Sensor Configuration

This section explains how to evaluate the safety by calculating the function (sensor) execution probability of the target system based on the sensor configuration determined in the previous section.

4.1 Model Checker PRISM

Model checking is a method for describing the system behavior formally and checking the correctness of the operation algorithmically. It is very effective as a method to improve system reliability because it can check the system design before the implementation of the target system, and can eliminate dependency on individual skills from the inspection and can perform a comprehensive inspection according to the algorithm.

PRISM [4] is one of the model checkers. In PRISM, the behavior of the system and the accompanying internal state change can be expressed by a state transition diagram. The determination of the transition destination by behavior can be expressed with probability. For example, in the case of network communication, an environment can be assumed in which packets arrive at 99.9%, and requests and ack are lost at 0.1%. When a packet does not arrive, it is necessary to execute retransmission processing up to a specified number of times. The probability of reaching a specific state can be calculated by expressing the local behavior of the system with probability and exhaustively searching the model. In the previous example, the tool can calculate the final communication success probability considering retransmission processing.

4.2 Sensing Probability Model

In this research, the sensor configuration tree defined in Sect. 3.2 is automatically converted into a PRISM model. The converted model consists of a sensing success/failure model for each sensor type and a model that expresses the achievement probability of each function of the system by integrating the sensing results of multiple sensor types. In the converted model, the probability of function failure (system safety) due to sensor failure is quantitatively evaluated by the tool calculating the transition probability to the function execution success state.

4.2.1 Sensor Model

First, a model for each sensor type is created with the PRISM model based on the sensor configuration tree designed in the Sect. 3.2. In the sensor configuration tree, the edge of the tree represents a single sensor type. Each sensor is assumed to succeed in detecting environmental information with probability p. Up to N units of this sensor are operated, and M of them are sensed successfully. In other words, if the relationship of M/N is $1/1$, only one target sensor is operating, and the target environment data cannot be obtained unless that sensor is successfully operated. Also, $2/3$ indicates that three sensors of the same type are operating and the correctness of the sensed data is being verified by majority vote. In this case, if up to one sensing failure occurs, system operation will not be hindered.

The Fig. 4 shows the sensing behavior of each sensor type as a state transition diagram with probability. First, each sensor evaluates its preconditions at the “Idle” state. By using the “requires” relationship expressed in the sensor configuration tree, it is possible to express a relationship in which the operation of one sensor depends on the opera-

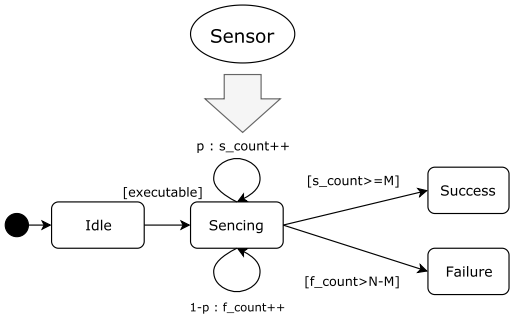

Fig. 4 State transition in sensor model

tion status of other sensors. In the "Idle" state, when a sensor with a "require" relationship becomes a success state, it transitions to the "Sensing" state. In the " Sensing " state, the sensor operates normally with a probability of p. By performing this success judgment N times, the simultaneous operation state of N sensors is reproduced. The results of sensor success and failure are totaled using a counter. If the number of successes is greater than or equal to M, it is assumed that the data has been sensed correctly by sensing, and if it is less than M, it is assumed that the sensing has failed and transitions to the "Failure" state.

list 1 Sensor Model

```
1  const int SUCCESS = 1;
2  const int FAILURE = 0;
3  const int N_sensor=1;
4  const int M_sensor=1;
5  const double p_sensor=0.9;
6
7  module sensor
8      s_sensor : [0..3] init 0;
9      result_sensor : [0..1] init 0;
10     s_count_sensor : [0..N] init 0;
11     e_count_sensor : [0..N] init 0;
12
13     [](s_sensor=0)-> (s_sensor'=1);
14     [](s_sensor=1)&!(s_count_sensor>=M_sensor)
           &!(e_count_sensor>N_sensor-M_sensor) ->
            p_sensor : (s_count_sensor'=
           s_count_sensor+1) + (1-p_sensor) : (
           e_count_sensor'=e_count_sensor+1);
15     [](s_sensor=1)&(s_count_sensor>=M_sensor)
           -> (s_sensor'=2)&(result_sensor'=
           SUCCESS);
16     [](s_sensor=1)&(e_count_sensor>N_sensor-
           M_sensor) -> (s_sensor'=3)&(
           result_sensor'=FAILURE);
17 endmodule
```

List 1 shows the sensor model in the PRISM language. The internal state of each sensor is managed by the state variable *s_sensor* (0: Idle state, 1: Sensing state, 2: Sensed success, 3: Sensed failure). The sensing results of each sensor are counted using *s_count* and *e_count* variables. In the Sensing state, it increments *s_count* with probability *p_sensor* and *e_count* with probability $1 - p\_sensor$. When the number of *s_count* become equal to *M_sensor*, it shifts to the sensed success state, and conversely, when *e_count* exceeds $N\_sensor - M\_sensor$, it transits to the sensed failure state.

Judgment, when the operation of a sensor depends on the state of other sensors, is described by the expression in the "Idle" state on line 7. The success status of other sensors

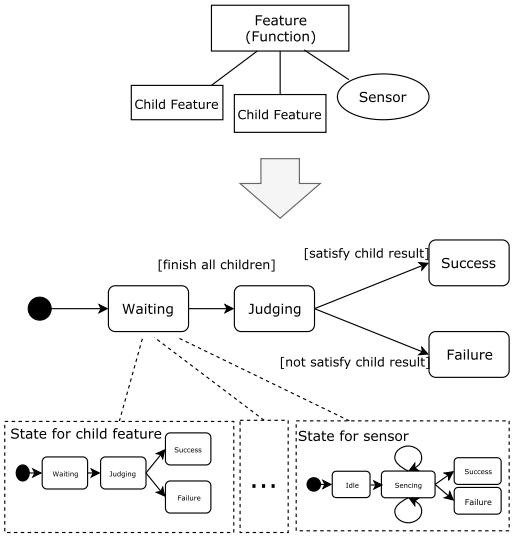

Fig. 5 State transition in integration model

is described as a transition condition. For example, if the target sensor operates only when sensor 2 is successfully sensed, write a conditional expression like follows.

```
1  [](s_sensor=0)&(result_sensor2=SUCCESS) -> (
       s_sensor'=1);
```

4.2.2 Integration of Sensor Results

After generating the individual sensor models, integrated models are generated that determines the success/failure of features by integrating the sensing results of each sensor. The integrated model is generated from the composition of intermediate features in the sensor configuration tree. Figure 5 shows the state transition diagram of the integrated model.

An integrated model is generated for each feature in the sensor configuration tree. Each integrated model consists of four states. In the "Waiting" state, it judges the states of the child nodes (feature/sensor) relating to its feature and waits until judges of all the child nodes are finished. Next, in the "Judging" state, the results of the child nodes are integrated. If the number of child nodes equal to or greater than the achievement condition X is successful, it transitions to the success state, otherwise it transitions to the failure state. The achievement condition X is determined when designing the sensor configuration tree.

list 2 Integration Model

```
1  module integration
2      s_integration : [0..3] init 0;
3      result_integration : [0..1] init 0;
4
5      [](s_integration=0)&(s_sensor1>=2)&...&(
           s_sensorN>=2) -> (s_integration'=1);
6      [](s_integration=1)&(result_sensor1+...+
           result_sensorN>=X)-> (s_integration'=2)
           &(result_integration'=SUCCESS);
7      [](s_integration=1)&!((result_sensor1+...+
           result_sensorN>=X))-> (s_integration'=3)
           &(result_integration'=FAILURE);
8  endmodule
```

List 2 shows the integrated model in PRISM language. At first, the sensor integration model judges whether all the child features (sensors) have been finished. The conditional expression on line 5 confirms that all the target sensor models (sensor1 - sensorN) have reached the end state. After that, in line 6-7, the sensor results are integrated. Each sensor judges the success/failure of detection considering the multiplexing of M/N. If the target sensor is successful, 1 is stored in the variable *result_sensorN*. If the values of all sensor variables *reuslt_sensorN* are summed and exceed the threshold value X, it means that the data necessary to achieve the feature is available. As a result, the feature is achieved and the state transitions to the success state ($result\_integration' = SUCCESS$).

5. Management of Environmental Information and Estimation of Sensing Probability

The autonomous driving system uses various sensors. Environmental information is acquired using a sensor, and the operation is determined based on that information. It can be said that the operation of the system depends on environmental information. This dependency can be applied to the sensor itself that acquires environmental information. Various sensors have strengths and weaknesses concerning the environment in terms of detection accuracy. For example, optical cameras can obtain information on the surrounding environment with high accuracy when the weather is clear during the daytime. On the other hand, the sensing accuracy and performance are greatly reduced when the lighting is poor at night or in rainy weather or fog. Thus, to evaluate the designed sensor configuration, it is necessary to identify the environment in which the target system operates. This section explains how to calculate and evaluate the sensing probability of the target system based on the environmental information that affects the sensors.

5.1 Management of Environment Information and Sensing Probability

In this study, we assume that the performance of each sensor used in the system is evaluated and managed in advance. This means that the characteristics of each sensor are measured by the catalog information of the sensor manufacturer or the preliminary analysis by the developer. The sensing probability of sensor s is determined depending on the environment $E =< e_1, e_2, \ldots, e_n >$ in which it operates. Environment E consists of environmental elements e_n (eg, illuminance, temperature, humidity, etc.). In the implementation of this research, each environmental element is divided into arbitrary M_n intervals, and the sensing probabilities are recorded and managed in the form of multidimensional arrays of $M_1 \times M_2 \times \ldots \times M_n$. The multidimensional array stores the minimum guaranteed value of sensing probability in the target environment.

The sensing probability of each sensor can be estimated by determining the environmental factors based on the operation scenario of the target system. For example, consider the operation of an autonomous driving system when testing a driving scene on a highway at night in rainy weather. It can be expected that the illuminance is low at night, the humidity is high due to rain, and the temperature is low, and specific value in that environment can be assumed. As a result, the sensing probability of each sensor in the target environment can be estimated. The estimated value is assigned to the detection probability p of each sensor model. By enumerating operational scenarios in advance, a PRISM model reflecting the sensing probability corresponding to each scene can be automatically generated.

5.2 System Safety Evaluation

This section describes how to calculate the probability of achieving any property of the system using the model checker PRISM. In PRISM, the probability of reaching any state can be calculated by expressing any state to be achieved using LTL (Linear Temporal Logic) formula. For example, to verify that the entire system works properly, calculate the probability that the top node of the sensor configuration tree will be in a successful state. In this case, the LTL formula can be described as follows.

```
1 P=?[(F result_top=SUCCESS)]
```

The above formula is the most typical one for evaluating safety, and makes the PRISM tool calculate the probability of achieving the function of the entire system. The leading "P =?" instructs the PRISM tool to calculate the probability of transition to the target state. In the above formula, by specifying ($F \quad result\_top = SUCCESS$), the probability that the value of result_top will be eventually SUCCESS is calculated. "result_top" is a variable that records the execution result of the top node of the sensor configuration tree. By checking that variable, it can be judged whether the objective of the entire system has been achieved.

It is possible to evaluate not only the entire system but also partial functions and the probability of occurrence of special states. For example, to verify whether the probability that the specified function will be achieved is 10% or less when a specific sensor is not working, verify the following formula.

```
1 P<=0.1 [ !(result_sensor=SUCCESS) U (result_func
      =SUCCESS) ]
```

In the above formula expresses the state in which detection by the sensor does not be achieved (not operating normally) until the specified function is executed normally. PRISM can set a threshold for the probability P, and can verify whether the condition is satisfied. In the above example, it verifies whether the probability is less than 10% by setting "P<=0.1".

In this way, by constructing the model of the target system and specifying the state which wants to be satisfied by the LTL formula, PRISM can calculate probability reaching a specified state. This process makes easy to verify any

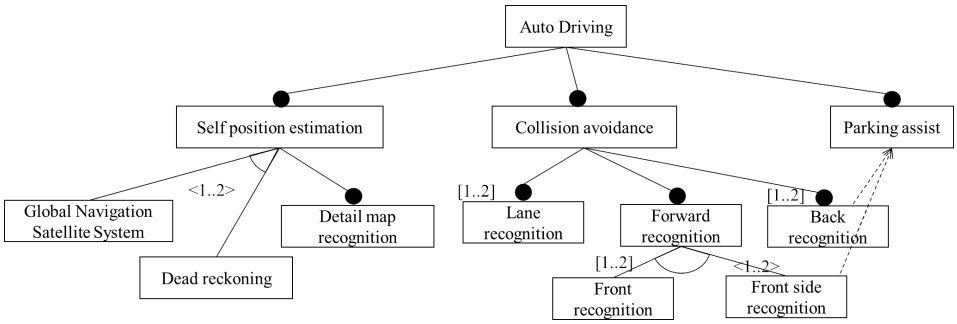

Fig. 6 Feature model for autonomous drive system

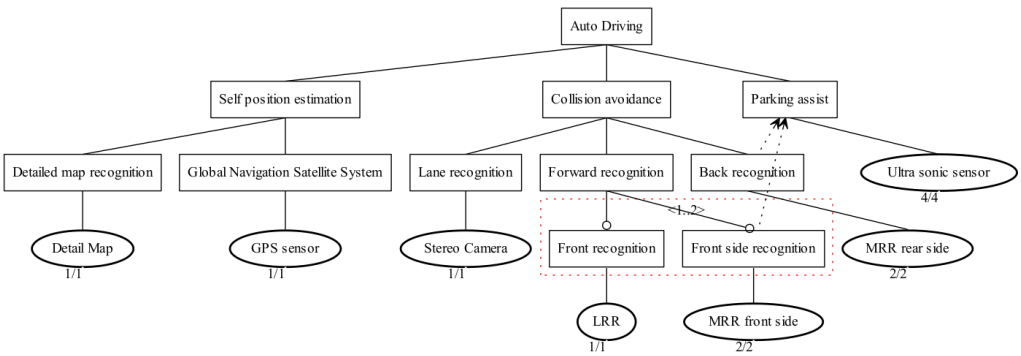

Fig. 7 Sensor configuration tree for autonomous drive system

system safety.

6. Case Study

In this section, we evaluate the effectiveness of the proposed method using an autonomous driving system as an example. First, the domain acknowledgment of the target system is shown below.

- To satisfy all requirements of the system, self-position estimation, collision avoidance and parking assistant must all active.
- Self-position estimation consists of two functions: acquisition of map information and current position measurement.
- Current position measurement can be achieved by GPS positioning or dead reckoning based on in-vehicle sensor information.
- To avoid collisions, three recognitions are required: lane recognition, forward recognition, and backward recognition.
- Although forward recognition is performed by front recognition using a long-range radar, it can be substituted by front-side recognition using a medium-range radar.
- Parking assistant requires to use both functions: back recognition and front side recognition.
- The autonomous driving failure probability due to sensor errors is aimed at 0.1% or less (success rate 99.9% or more).

Figure 6 shows the relationship between the functions that make up the system based on the above knowledge in a feature model. It is essential to achieve the three sub-functions “Self-position recognition”, “Collision avoidance” and “Parking assist” necessary to achieve the top-level function “Auto driving”. The two functions for achieving self-positioning are connected in a selective relationship with group multiplicity < 1..2 > so that one or more of the two needs to be achieved. Similarly, long-distance “Front recognition” and medium-distance “Front side recognition” are connected in a selective relationship for forward recognition, indicating that one or more of them are required. “Parking assist” also requires “Back recognition” and “Front side recognition”. The requirements are expressed by require relation.

Next, the sensor configuration of the system is determined based on the feature model. First, redundancy at the functional level is determined from the selective relationship between features. The two features of self-position estimation and forward recognition have a selective relationship on the feature model. The group cardinality of self-position estimation is < 1..2 >. Here, we will use only the Global Navigative Satelite System for self-position estimation. Similarly, forward recognition also has group cardinality < 1..2 >, but this is installed with both functions for providing redundancy. This means that there is no problem if either one is operating normally. In this way, the function to be installed is determined by referring to the selective relationship part on the feature model and selecting a subset from the candidates. After determining the functional configuration as described above, the sensors are determined and assigned to the functions. Refer to the feature (func-

Table 1 Sensing probability in each situation

Sensor Type	Situation 1	Situation 2
GPS sensor	0.99999	0.97
Detail Map	0.99999	0.99999
Sterao Camera	0.99999	0.97
LRR	0.99999	0.99999
MRR front side	0.99999	0.99999
MRR rear side	0.99999	0.99999
Ultra sonic sensor	0.99999	0.99999
Wheel encoder	-	0.99995

tion) at the edge of the feature model, and determine and assign the sensor required to achieve the function. At this time, referring to the feature cardinality of the target feature, the number of sensors to be mounted in that range is determined. For example, the MRR rear side sensor is assigned to achieve back recognition. It is assumed that these sensors are mounted one by one on the left and right rear of the vehicle, and that both of them operate normally, and that back recognition is achieved. Therefore, $M/N = 2/2$ is determined as the minimum number of operating sensors and the number of installed sensors. The sensor configuration of the target system is determined by completing the sensor assignment for other edge features in the same way.

Next, we assume the verification environment of the determined system. Two environments are assumed this time. The first is for driving on ordinary roads when the weather is clear during the day, and the second is for driving on highways when it is raining. To estimate the sensing probability of the sensor in the assumed environment, the environment is decomposed into environmental elements. Assuming speed range, brightness, and weather as environmental elements, element values in the assumed environment are determined. For example, in the first environment, it can be converted into environmental element values in the form of $E_1 = \{speedrange = medium, lightness = high, weather = clear\}$. Similarly, the second environment can be converted to $E_2 = \{speedrange = high, lightness = low, weather = rain\}$. Based on these environmental element values, the sensing probability is obtained from the sensing probability array of each sensor. For example, the sensing probability of the GPS sensor can be estimated as X% in the first environment and Y% in the second environment. By repeating this ""number of sensor types" × "number of environments"" times, the sensing probability necessary for system verification is obtained. Table 1 shows the sensing probability of each sensor estimated in this case study.

list 3 Model for Autonomous Driving

```
---(omission)---

//// Model of GPS_sensor ////
const int N_GPS_sensor=1;
const int M_GPS_sensor=1;
const double p_GPS_sensor=0.99999;

module GPS_sensor
    s_GPS_sensor : [0..3] init 0;
    result_GPS_sensor : [0..1] init 0;
    s_count_GPS_sensor : [0..1] init 0;
    e_count_GPS_sensor : [0..1] init 0;

    [](s_GPS_sensor=0)-> (s_GPS_sensor'=1);
    [](s_GPS_sensor=1)&!(s_count_GPS_sensor>=
        M_GPS_sensor)&!(e_count_GPS_sensor>
        N_GPS_sensor-M_GPS_sensor) ->
        p_GPS_sensor : (s_count_GPS_sensor'=
        s_count_GPS_sensor+1) + (1-p_GPS_sensor
        ) : (e_count_GPS_sensor'=
        e_count_GPS_sensor+1);
    [](s_GPS_sensor=1)&(s_count_GPS_sensor>=
        M_GPS_sensor) -> (s_GPS_sensor'=2)&(
        result_GPS_sensor'=SUCCESS);
    [](s_GPS_sensor=1)&(e_count_GPS_sensor>
        N_GPS_sensor-M_GPS_sensor) -> (
        s_GPS_sensor'=3)&(result_GPS_sensor'=
        FAILURE);
endmodule
//////////////////////////////

//// Model of Self_position_estimation ////
module Self_position_estimation
    s_Self_position_estimation : [0..3] init 0;
    result_Self_position_estimation : [0..1]
        init 0;

  [](s_Self_position_estimation=0)&(
      s_Detailed_map_recognition>=2)&(
      s_Global_Navigation_Satellite_System>=2)
      -> (s_Self_position_estimation'=1);
  [](s_Self_position_estimation=1)&(
      result_Detailed_map_recognition=SUCCESS)
      &(
      result_Global_Navigation_Satellite_System
      =SUCCESS)-> (s_Self_position_estimation
      '=2)&(result_Self_position_estimation'=
      SUCCESS);
  [](s_Self_position_estimation=1)&!((
      result_Detailed_map_recognition=SUCCESS)
      &(
      result_Global_Navigation_Satellite_System
      =SUCCESS))-> (s_Self_position_estimation
      '=3)&(result_Self_position_estimation'=
      FAILURE);
endmodule
//////////////////////////////

---(omission)---
```

The sensor configuration model and the sensing probability are input to the model conversion program that implements the proposed method and converted to the PRISM model. A part of the converted PRISM model is shown in list 3. For this model, specify the property to be verified and calculate its achievement probability. This time, we calculate the achievement probability of the top feature "Auto driving". The formula given is

```
P =? [(F result_Auto_driving = SUCCESS)]
```

By repeating the conversion to the PRISM model and the verification of properties, the verification of the system can be achieved comprehensively.

6.1 Revision and Revaluation

Once the sensing probabilities are determined based on the environment, PRISM can calculate the achievement probability of the specified constraint. To calculate manually the probability, it can be obtained by calculating the achievement probability of sensors and functions from the bottom up. For example, the achievement probability of "Forward

recognition" P_{for_rec} can be obtained by the following calculation.

$$
\begin{aligned}
P_{lrr} &= P_{fro\_rec} = 0.99999 \\
P_{mrr\_fs} &= 0.99999 \\
P_{fs\_rec} &= (P_{mrr\_fs})^2 = 0.9999800001 \\
P_{for\_rec} &= 1 - ((1 - P_{fro\_rec}) * (1 - P_{fs\_rec})) \\
&= 0.9999999998
\end{aligned}
$$

The difference between the calculation result and the PRISM result is less than 10^{-7}. The numerical errors are caused by differences in approaches. We believe that it is accurate enough to estimate the probability of achievement in the early phase of development. By performing the same calculation up to the root node of the system, the achievement probability of the entire system can be calculated. However, the calculation must take into account the calculation order based on the dependencies. In the case study, "Back recognition" and "Front side recognition" must be calculated before calculating "Parking assist".

In the initial configuration, the achievement probabilities of entire system in the two environments were 99.989% and 94.082%. Figure 8 shows a screenshot of the PRISM tool evaluating the rainy environment. The probability of the first sunny environment exceeds 99.9%, but the target value is not reached in the second rainy environment. This is thought to be because the communication accuracy of the GPS sensor is reduced due to rain clouds and the sensing probability of the stereo camera is reduced due to water droplets. Considering the variability of the feature model, we aim to increase the achievement probability by strengthening redundancy. First, instead of relying solely on the GPS sensor to estimate self-position, we install dead reckoning as GPS support. Dead reckoning uses a wheel encoder to estimate self-position. Also, as a countermeasure against a decrease in the accuracy of lane recognition cameras due to water droplets, two cameras are installed at different positions. If either camera can recognize the lane, it can work without problems. The revised sensor configuration is shown in Fig. 9. The achievement probability is calculated again for the revised model. With this revision, the achievement probability in rainy weather became 99.901%, and it was reached that the target value. By repeating the designing, evaluation, and revision of the sensor configuration as described above, the safety of the target system can be evaluated at the early stage of system design.

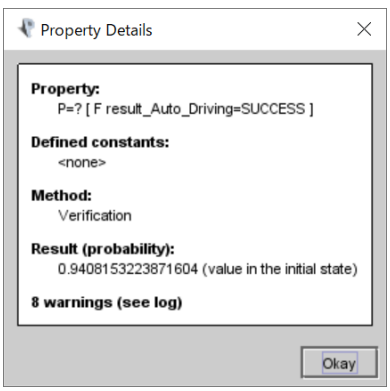

Fig. 8 Evaluation result in rainy environment

7. Discussion

This section describes the effectiveness and limitations of the proposed method.

7.1 Advantage of Using Model Checker

The verification time in the case study was 20.994 seconds for the initial configuration and about 385.226 seconds for the revised configuration. This verification time increases as the number of features and sensors that construct the model increases. If only the probability of achievement of the top function is calculated, it is possible to calculate faster than the PRISM model by designing a dedicated program. However, we chose using PRISM tool to calculate them for two reasons.

The first reason is that we do not need to consider the calculation order due to dependencies. If it is a simple tree structure, it is sufficient to calculate in order from the leaf to the root. When multiple functions share the same sensor, cross-cutting dependency on the tree occurs, and it is necessary to determine the calculation orders. However, in using of PRISM, it automatically started the calculation from the part that can be calculated, so there is no need for the developer to consider such order.

The second reason is the ease of constraint verification. The most typical constraint is whether the entire sys-

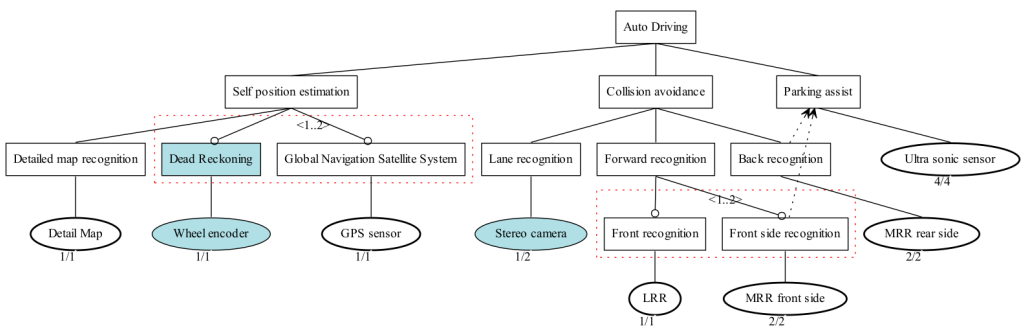

Fig. 9 Revised sensor configuration tree

tem works properly. However, actual system verification requires verification of various constraints. As described in Sect. 5.2, in this method, various verifications can be performed simply by describing constraint expressions. If we want to manually verify the achievement probabilities for constraints, we must design a formula to calculate them. Similarly, the cost is high when implementing a calculation program. Our method using a model checker facilitates probability verification.

7.2 Computational Cost

As mentioned in Sect. 7.1, the proposed method takes a lot of time to calculate a large model, and there is still a problem in scalability. In this paper, conversion to the PRISM model and evaluation process are a major contribution, and we think the scalability of the method is out of the scope. However, scalability improvement is considered essential for actual operation, and one strategy for improvement is described in this section.

Converting an entire system into a single PRISM model requires enormous computational costs. Therefore, the computation time can be dramatically reduced by decomposing the system into multiple subtrees and calculate them in a step by step. For example, if the revised configuration can be decomposed into three subtrees ("self-position estimation", "collision avoidance and parking assist" and "Integration into top function"), they can be calculated in about 0.5 seconds in total by PRISM. When decomposing into subtrees, it is important to decompose into appropriate sizes, taking into account the cross-cutting dependencies. There is another benefit to decomposing into subtrees. When redesigning the sensor configuration, if it is divided into subtrees, it only revaluates the changed subtrees and the upper subtrees, so the results of subtrees that do not need to be changed can be reused. Thus, the cost of redesign can be reduced. Proposal of specific algorithm is future work.

7.3 Verification Ability

In this method, the achievement probability of the function is the object of verification, and for this purpose, the model checking method PRISM that can handle the probability is adopted. The ability to handle probabilities is a major feature of PRISM, but it is also possible to verify whether or not a specific state can be achieved. Depending on the verification formula, flexible verification can be performed. However, real-time property cannot be verified as a model limitation. For example, depending on the situation, when the correct value cannot be obtained, the old value may be temporarily used. In the case of position estimation, the restriction can be expressed as property such as "use position information acquired within the past 10 seconds". This method cannot handle real-time information such as "within the past 10 seconds". To verify this, it is necessary to express the sensing time and the interval in the model, and also use time information in the constraint equation. We are planning the model extension for the verification of real-time properties.

The purpose of this method is to estimate the probability of achievement of functions at the initial stage of design. Even if this verification is passed, it is possible that the function cannot be achieved in the actual environment due to the influence of sensor placement and unexpected environmental factors. The unexpected behaviors can be dealt with by dynamically changing the functional configuration using the self-adaptive system technique.

7.4 Estimation of Sensing Probability

To use this method, it is necessary to evaluate the sensing probability of the individual sensor in advance. It is necessary to manage environmental elements and sensing probabilities. It is necessary to determine what environmental elements should be considered and how much the environmental elements should be divided based on domain knowledge. When new environmental elements are added, the effects of these elements need to be remeasured. It is costly. As for the estimation of the sensing probability in a specific environment, instead of clearly defining and managing the environmental elements to be considered as in this method, a method of estimating by using machine learning is also conceivable. With machine learning, we expect that the potential influences from the environment can be estimated.

8. Related Works

Model checking is useful for verifying system operation before implementation. Various model checkers that have different characteristics have been proposed [5]. The SPIN [6] is particularly good at expressing communication between parallel processes, and it is possible to check whether constraints can be satisfied by writing LTL expressions. This is especially effective for protocol verification. The NuSMV [7] is a model checker called a symbolic model checker. This also can verify the reachability of a specific state by describing the constraints by the LTL expression. A large number of states can be inspected efficiently. The UPPAAL [8] is a model checker that can express time information on a model. This makes it possible to verify systems with time constraints. In our method, the PRISM [4] is adopted as a model checker. This is because a transition probability can be expressed in the model. Since the sensing probability of the sensor can be handled explicitly and the probability of normal operation of the system based on it can be calculated, it is suitable for this research.

The autonomous driving system in this research realizes collision avoidance by integrating information from multiple sensors. Machine learning has been attracting attention to realize these information integrations and various researches have been proposed [9], [10]. Machine learning is suitable for automatically modeling behaviors that are difficult to rule manually. However, generated models become black boxes. They are difficult to be proved own safety. By using our method, it is possible to perform comprehen-

sive inspections and mathematical quantitative evaluations, which can be used to explain the safety of these systems.

In this research, the redundancy of the software system is modeled using the variability of the feature model. The purpose of this research is to assist the designing of redundancy, and even if the target probability is achieved by this research, it does not guarantee that the system operates as a safe system. Even if there is sufficient system redundancy, it does not make sense if it cannot be switched operations properly in runtime. To continue to achieve the objectives, it is considered that the combination with the research of the self-adaptive system is effective. In a self-adaptive system, the system configuration and behavior are dynamically determined and changed according to the runtime environment. For example, Ghezzi et al. [11] have proposed a method of changing the behavior appropriately by judging the timing when the system can be safely switched. Ramirez et al. [12] proposed a method to prevent the occurrence of fatal problems as a system by allowing some of the requirements to be relaxed. To effectively operate the redundant configuration handled in our method in an actual environment, it is effective to combine these techniques.

9. Conclusion

In IoT systems, data acquired by many sensors are required. However, since sensor operation depends on the actual environment, it is important to ensure sensor redundancy to improve system reliability in IoT systems. To evaluate how many sensors are multiplexed and what is effective as an alternative function, it is important to estimate the achievement probability of the function based on the sensing probability at the initial design phase. In this research, we proposed a method to automatically generate a PRISM model from the sensor configuration of the target system and to calculate and verify the function achievement probability in the assumed environment. By designing and evaluating iteratively until the target achievement probability is reached, the reliability of the system can be estimated at the initial design phase. This method reduces the possibility that the lack of reliability will be found after implementation and the redesign accompanying it will occur.

Future works include strengthening verification abilities. We believe that it is possible to verify the properties and constraints for the operation cycle of IoT systems by extending real-time information to the model. This extension makes it possible to design reliable IoT systems more easily. Also, as a problem inherent to model checking, scalability needs to be improved. We think that it is possible to verify large-scale complex systems composed of a large number of sensors by using model decomposition and aggregation methods.

References

[1] L.D. Xu, W. He, and S. Li, "Internet of things in industries: A survey," IEEE Trans. Ind. Informat., vol.10, no.4, pp.2233–2243, Nov. 2014.
[2] M. Kwiatkowska, G. Norman, and D. Parker, "Prism: Probabilistic symbolic model checker," Computer Performance Evaluation: Modelling Techniques and Tools, ed. T. Field, P.G. Harrison, J. Bradley, and U. Harder, Berlin, Heidelberg, vol.2324, pp.200–204, Springer Berlin Heidelberg, 2002.
[3] K. Czarnecki, S. Helsen, and U. Eisenecker, "Staged configuration using feature models," Software Product Lines, ed. R.L. Nord, Berlin, Heidelberg, vol.3154, pp.266–283, Springer Berlin Heidelberg, 2004.
[4] M. Kwiatkowska, G. Norman, and D. Parker, "Prism: Probabilistic symbolic model checker," Computer Performance Evaluation: Modelling Techniques and Tools, ed. T. Field, P.G. Harrison, J. Bradley, and U. Harder, Berlin, Heidelberg, vol.2324, pp.200–204, Springer Berlin Heidelberg, 2002.
[5] S. B and S. Jayadevappa, "Model checkers - tools and languages for system design - a survey," 5th International Conference on Advanced Information Technologies and Applications, pp.39–51, 11 2016.
[6] G. Holzmann, The SPIN Model Checker: Primer and Reference Manual, 1st ed., Addison-Wesley Professional, 2011.
[7] A. Cimatti, E.M. Clarke, F. Giunchiglia, and M. Roveri, "Nusmv: A new symbolic model verifier," Proc. 11th International Conference on Computer Aided Verification, CAV '99, London, UK, UK, vol.1633, pp.495–499, Springer-Verlag, 1999.
[8] J. Bengtsson, K. Larsen, F. Larsson, P. Pettersson, and W. Yi, "Uppaal — a tool suite for automatic verification of real-time systems," Proc. DIMACS/SYCON Workshop on Hybrid Systems III: Verification and Control: Verification and Control, Secaucus, NJ, USA, vol.1066, pp.232–243, Springer-Verlag New York, Inc., 1996.
[9] S. Hoermann, M. Bach, and K. Dietmayer, "Dynamic occupancy grid prediction for urban autonomous driving: A deep learning approach with fully automatic labeling," 2018 IEEE International Conference on Robotics and Automation (ICRA), pp.2056–2063, May 2018.
[10] Q. Rao and J. Frtunikj, "Deep learning for self-driving cars: Chances and challenges," 2018 IEEE/ACM 1st International Workshop on Software Engineering for AI in Autonomous Systems (SEFAIAS), pp.35–38, May 2018.
[11] C. Ghezzi, J. Greenyer, and V.P.L. Manna, "Synthesizing dynamically updating controllers from changes in scenario-based specifications," 2012 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), pp.145–154, June 2012.
[12] A.J. Ramirez, B.H.C. Cheng, N. Bencomo, and P. Sawyer, "Relaxing claims: Coping with uncertainty while evaluating assumptions at run time," Model Driven Engineering Languages and Systems, ed. R.B. France, J. Kazmeier, R. Breu, and C. Atkinson, Berlin, Heidelberg, vol.7590, pp.53–69, Springer Berlin Heidelberg, 2012.

Ryuichi Takahashi received the B.E., M.E. and D.E. from Waseda University, Tokyo, Japan in 2007, 2008 and 2012. During 2011–2017, he was an assistant professor at Waseda University. He is now with Ibaraki University, Japan. His research interests include software engineering especially the design of interactions for distributed systems.