Morse: Reducing the Feature Interaction Explosion Problem Using Subject Matter Knowledge as Abstract Requirements

Laure Millet, Nancy A. Day, and Jeffrey J. Joyce
Critical Systems Labs
Vancouver, BC, Canada
Email: {laure.millet, nancy.day, jeff.joyce}@cslabs.com

***Abstract*—The feature interaction problem appears in many different kinds of complex systems, especially systems whose elements are created or maintained by separate entities - for example, a modern automobile that incorporates electronic systems produced by different suppliers. Cross-cutting concerns, such as safety and security, require a comprehensive analysis of the possible interactions. However, there is a combinatorial explosion in the number of feature combinations to be considered. Our work approaches the feature interaction problem from a novel point of view: we seek to use the abstract subject matter knowledge of domain experts to deduce why some features will NOT interact, rather than trying to discover or resolve the interactions. In this paper, we present a method that can automatically reduce the required number of combinations and situations that have to be evaluated or resolved for feature interactions. Our tool, called Morse, rules out feature combinations that cannot have interactions based on traceable deductions from relatively simple abstract requirements that capture relevant subject matter knowledge. Our method is useful as a means of focusing attention on particular situations where more detailed functional requirements may be needed to avoid unacceptable risk arising from unintended interactions between features.**

I. INTRODUCTION

The feature interaction problem [1], [2] arises when components of a system are developed independently and then combined on a common platform where a feature can influence the behaviour of other features. While feature-oriented development facilitates modular development through separate suppliers, it requires comprehensive, deep integration testing and causes an explosion in the size of the verification task. Cross-cutting concerns, such as safety and security, of many complex systems make the search for solutions to this problem of paramount importance.

Most work on the feature interaction problem has focused on 1) how to define a feature and a feature interaction (*e.g.,* [3], [4]); 2) how to analyze combinations of features to detect interactions (*e.g.,* [5], [6], [7]); and 3) architectures/policies for governing/resolving feature interactions (*e.g.,* variable-specific [8], priority-based [9], Distributed Feature Composition (DFC) [10]) .

Our work approaches the feature interaction problem from a novel point of view. We seek to deduce why features will **not** interact from abstract requirements that domain experts often know about features. Much as a detective eliminates suspects before arriving at the only remaining culprit, our goal is to use the knowledge of domain experts, expressed abstractly, to **rule out** combinations of features in situations that cannot have interactions. We present a method (and tool), called **Morse**, that can automatically reduce the required number of combinations and situations that have to be evaluated or resolved for feature interactions, based on traceable deductions from requirements knowledge.

We focus on feature interactions that are caused by shared effects on features or data. Our approach considers collections of features, not just pairwise combinations. Of key importance in our approach is that we rule out combinations and situations based on abstract requirements from the domain experts. These stakeholders provide information about the relationships between features and the conditions under which a feature is relevant, which allow us to rule out combinations. These requirements provide a traceable argument for why features do not interact. We do not require a complete model of the behaviour of the feature, as is required for many approaches that search for interactions (*e.g.,* model checking). By using abstract information, the problem becomes tractable – we are unlikely to suffer from a state space explosion problem.

The contributions of our work are:

- A modelling approach that captures abstract requirements of features that are relevant to their interactions, including a novel representation of the model using a feature relationship graph.
- A formal definition for using these requirements to reduce the problem to regions or feature combinations, which are subsets of the entire set of features in a situation (set of conditions), that require further investigation for feature interactions.
- Two examples to demonstrate the reduction in the feature interaction problem accomplished by our approach: 1) a set of automotive active safety features; and 2) the 1993 crash of an Airbus 320.

We believe that by taking the novel point of view of ruling out feature combinations based on abstract requirements, we have made an important contribution to reducing the state space explosion problem in the verification of combinations of features. Our results focus attention on particular situations

© 2018 IEEE. L. Millet, N. A. Day and J. J. Joyce, "Morse: Reducing the Feature Interaction Explosion Problem using Subject Matter Knowledge as Abstract Requirements," 2018 IEEE 26th International Requirements Engineering Conference (RE), Banff, AB, 2018, pp. 251-261, doi: 10.1109/RE.2018.00033.

where more detailed functional requirements may be needed to avoid unacceptable risk arising from unintended interactions.

II. Overview

In this section, we provide a small example of a set of features and abstract requirements of subject matter knowledge to illustrate how our method allows us to remove some combinations of features from consideration for further analysis. Our example is based on the University of Waterloo Feature Model Set (UWFMS) [11], a set of non-proprietary automotive features modelled in Stateflow. Our example includes four features with the informal descriptions in Table I. For our purposes, these features are described succinctly by the abstract requirements in Table II. We chose the words *relevant* and *influence* as general words that can cover a number of meanings depending on the subject context.

TABLE I
Example UWFMS Features

- **CC:** Cruise Control controls the throttle to adjust the vehicle speed according to a target speed as set by the driver.
- **EVA:** The Emergency Vehicle Avoidance feature safely clears the road by pulling over when an emergency vehicle is detected.
- **RA:** The reverse assistant brakes to mitigate or prevent a collision when driving in reverse.
- **CA:** The Collision Avoidance feature mitigates or prevents a collision with an obstacle by braking.

We represent these statements graphically in Figure 1. We call this graph a *feature relationship graph (FR graph)*. It is similar to a typical dataflow or dependency diagram showing features (CC, EVA, RA, CA) and parameters that they influence (throttle, brakes, and speed), but also includes the conditions for when a feature (or parameter) is relevant (under the feature or parameter nodes), and the conditions for when a node can influence another node (listed on the edges).

From these influences and conditions, we can deduce that there are only two situations where two or more features can influence the speed (assuming the gear can only be in one of drive or reverse):

1) When the vehicle is in drive and runs at a speed above 25kmh and below 40kmh, then both EVA and CA influence the speed.
2) When CC is enabled and the vehicle is in drive at a speed greater than or equal to 40kmh, then CC, CA, EVA can influence the speed.

In all other interpretations of truth values for these conditions, only one feature can influence the speed. Thus, rather than considering all possible feature combinations for feature interaction analysis, we have reduced the problem to considering only the sets of features stated above under the situations (set of conditions) stated. For a set of four features each with a state space of size n, we have reduced the analysis problem from n^4 to $n^3 + n^2$. This reduction is even more significant because the situation reduces the amount of feature behaviour to analyze. In order to check for a feature interaction, it is sufficient to look at the possibly interacting features in the context of the situation only.

In the next sections, we describe our method precisely beginning with our first contribution: how to model subject matter knowledge as abstract requirements that we can use.

III. Relevance and Influence Requirements

The input language of Morse is a set of English-like sentences that specify abstract requirements of subject matter knowledge describing the system. These requirements depict subsystems we call **features** by describing their behavior in terms of their modification of the **parameters** of the system. The features are active elements and more particularly the elements of the system the modeller is concerned about interacting, whereas a parameter is passive element such as an actuator, a sensor or any kind of shared data. The term **node** means feature or parameter. There are two kinds of requirements that can be provided as input to our tool:

- **Relevance requirements:** A relevance requirement is an English statement about the conditions under which a node is relevant. A relevance requirement is of the form **N1 is relevant only if C1**, where N1 is a node name and C1 the condition under which N1 is relevant. The meaning of the term “relevant” needs to be disambiguated in each project. However, early in system development engineers often have a sense of when some part of the system is relevant. For example, a feature is relevant when it is enabled or running; a parameter is relevant when it can be modified. There can be only one relevance requirement for a node (because one condition can be composed of multiple expressions through logical operators).
- **Influence requirements:** An influence requirement provides information about how a node influences another node. An influence requirement is of the form **N1 influences N2 when C1**, where C1 is the condition under which this influence exists. For example, a feature influences a parameter if it modifies its value; a parameter influences a feature if it is an input to the feature. A parameter can influence another parameter either when the first one is an actuator that mechanically modifies the value of the second or if there is a physical relationship such that the value of the second is dependent of the first one. A feature can influence another feature if the first one sends information to the second one. We assume that the influence relation is transitive (*i.e.,* if A influences B and B influences C then A influences C). There can be only one influence requirement for a pair of nodes.

Morse is not strictly tied to a given level of specification. In particular, one could mix information from a high-level specification with more specific details as needed.

We have a simple grammar for describing the conditions, which can include propositional logic operators, numeric operators, equality, and constants. The order of the requirements

TABLE II
ABSTRACT REQUIREMENTS FOR UWFMS FEATURES

- CC is relevant only if CC is enabled.
- CC influences the throttle when *speed* ≥ 40 and *gear* $=$ *drive*.
- EVA influences the throttle when *speed* ≥ 0 and *gear* $=$ *drive*.
- EVA influences the brakes when *speed* ≥ 0 and *gear* $=$ *drive*.
- RA influences the brakes when $10 \leq speed \leq 25 \ \wedge \ gear = reverse$.
- CA influences the brakes when *speed* > 25 and *gear* $=$ *drive*.
- The throttle influences the speed.
- The brakes influence the speed.

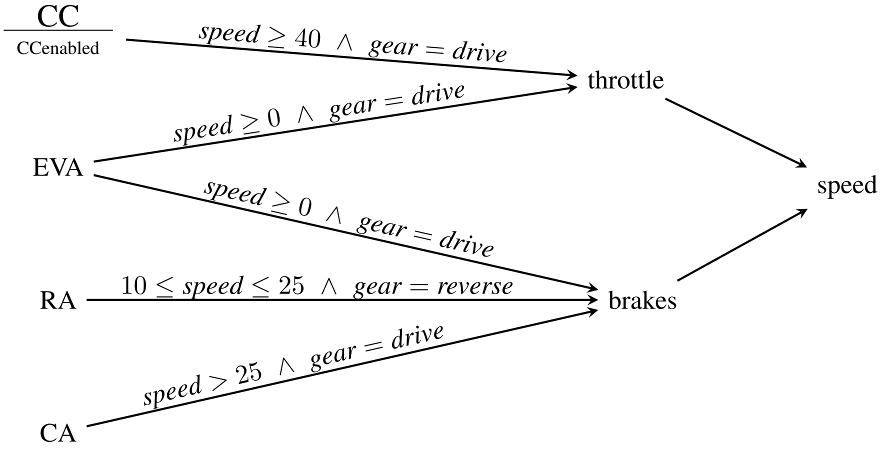

CC
CCenabled
$speed \geq 40 \ \wedge \ gear = drive$
throttle
$speed \geq 0 \ \wedge \ gear = drive$
EVA
speed
$speed \geq 0 \ \wedge \ gear = drive$
RA
$10 \leq speed \leq 25 \ \wedge \ gear = reverse$
brakes
$speed > 25 \ \wedge \ gear = drive$
CA

Fig. 1. Feature Relationship Graph for UWFMS Abstract Requirements

in the input file is not significant. We use the closed world assumption that the only relevance and influence requirements are those described in the input file.

From this information, we create a **feature relationship graph (FR graph)**. The graph nodes are features and parameters. The relevance requirements are the conditions on the nodes. The influence requirements are the edges and the conditions on the edges. There is a one-to-one relationship between an FR graph and its relevance and influence requirements.

IV. RULING OUT FEATURE COMBINATIONS

The goal of the Morse method is to use the relevance and influence requirements to separate the set of all features into groups, which we call regions, such that all feature interactions are contained within one of these groups. Combinations of features that are not in one of these groups do not need to be checked for feature interactions.

A **potential feature interaction** is a situation in which two or more features can both influence a node in the graph. A feature interaction can occur only if there is an interpretation that satisfies all of the conditions under which each feature in the region influences the node. If such an interpretation does not exist, we can rule out the possibility of a feature interaction, unless our closed world assumption is violated (*i.e.,* there is missing or new information about feature relationships).

From a set of features and parameters, and a set of relevance and influence requirements, the result of our method is a set of regions (sub-graphs) of the FR graph, each with an attached situation. A situation is a condition that describes a set of interpretations (of the variables used in the conditions) that satisfy it[1]. In *every* interpretation of the situation, there is a potential feature interaction between the features of the region. Our method guarantees that there is no interpretation in which a feature interaction can occur between features that are not together in one of the regions. However, it is possible for a feature to appear in multiple regions.

Three key challenges in creating our method are: 1) to combine related regions and situations such that we do not end up with an unmanageably large set of regions; 2) to limit the number of computationally expensive satisfiability checks; and 3) to avoid non-terminating analysis when there are loops in the FR graph.

A. Primitive Feature Relationship Graphs

We begin by considering a primitive feature relationship graph, which has no conditions (on nodes or edges). There is at most one edge in a direction between a pair of nodes. Without conditions, an FR graph is essentially a data flow graph and our goal is to break it into parts such that all the feature

[1] One condition can be used to express a set of conditions via logical operators.

interactions are contained within one of the parts. Figure 2 shows a set of influence requirements for features F1-F5 and its FR graph. The remaining nodes in the graph are parameters.

I1 influences F1
I1 influences F2
F1 influences O1
F1 influences X
X influences O2
X influences O3
F2 influences X
F3 influences Y
Y influences O4
F4 influences Y
F4 influences Z
F5 infuences Z
Z influences O4
Z influences O5

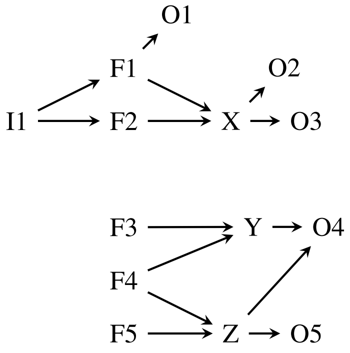


Fig. 2. Example FR Graph #1

An **influence path** is a chain of directed influence edges between two nodes in the FR graph. For example, there is an influence path from F2 to O2. We consider each node to have an influence path to itself. Two features cannot interact if there is no node that they both influence. An **interaction point** is a node that has an influence path to it from at least two features. F3 and F4 can both influence interaction point Y. F3, F4, and F5 can all influence O4. Our goal is to group features into regions based on their interaction points. We include multiple interaction points in the same region if there is an influence path between the interaction points, *i.e.,* a potential interaction at one can cause a potential interaction at another. For example, an interaction at Z can cause an interaction at O4. This grouping partly addresses challenge #1 stated above to avoid an explosion in the number of regions and keep together related interactions for analysis. A **maximal interaction point** of a graph of nodes is a node, n, in the graph, and every other node in the graph has an influence path to n. If there is a cycle in the FR graph, a set of nodes does not necessarily have a unique maximal interaction point.

Definition 1: A **region**, g, **of a *primitive* FR graph**, G, is a non-empty, full sub-graph (nodes and edges) of G such that:

1) there exists a node, n, in g that is a maximal interaction point for the region;
2) there is no node in $G \setminus g$ that has an influence path to node n, and there is no node in $G \setminus g$, such that n has an influence path to it;
3) g includes *at least two* feature nodes; and
4) g does not include a parameter node that does not have an influence path *from* a feature[2].

As a full sub-graph of the FR graph, a region includes all the edges between its nodes that are in the FR graph. A consequence of this definition is that all nodes within a region must be connected. The second part of the definition ensures that in the set of regions of a primitive FR graph, no region's set of nodes are a subset of another region's nodes.

The regions for the FR graph of Figure 2 are:

1) F1, F2, X, O2 (for maximal interaction point O2)
2) F1, F2, X, O3 (for maximal interaction point O3)
3) F3, F4, F5, Y, Z, O4 (for maximal interaction point O4)
4) F4, F5, Z, O5 (for maximal interaction point O5)

Each region includes all the edges in the original FR graph between the nodes present in the region.

A feature can appear in multiple regions. A region cannot consist of only one feature so the path from F1 to O1 is not in any region. Because of point 4 of the definition of region, I1 does not appear in any region because no feature influences it. Two regions can have the same set of features, but must have different maximal interaction points (as in regions R1 and R2). A **terminal node** is one with no outgoing edges. In our example, O1–O5 are all terminal nodes. Without cycles in the graph, there is a region for every terminal node that has two or more features that have influence paths to it. For a primitive FR graph, our result is the same as doing a cone-of-influence reduction [12] or slicing a data flow graph [13] from all the interaction points and combining the sub-graphs appropriately.

We find it useful to distinguish regions such as R1 and R2 in order to focus on different non-interacting maximal interaction points. However, a user may wish to reduce the regions to the set of features only and then merge subsets, which, for this example, would result in feature sets: { F1, F2 } and { F3, F4, F5 }.

If there are cycles in the FR graph, which we have found occur often, the definition of a region does not change because a region can have multiple maximal interaction points (all of which have influence paths between them), which addresses challenge #1 of avoiding producing too many regions.

Problem Reduction: In this example, from the original set of five features, we have reduced the feature interaction problem to checking sets of features of size three and size two. Every time we are able to remove even one feature from the set of features that must be considered together we get an exponential decrease in the size of the problem. For example, if the state space of every feature is size n, then checking all five features together has a state space of size n^5, whereas we have reduced the problem to sub-problems of size n^3 and n^2. Since the state space of a single feature model is usually exponential in its number of variables, this reduction can be substantial.

Justification for Reduction: Without conditions on the FR graph, a feature interaction is possible when there are two or more influence paths from different features to the same node. We justify our reduction with the argument that based on the definition of a region, there cannot be two paths that start at different features that lead to the same node that are not both contained within a region; thus all feature interactions must

[2] Here, the analysis distinguishes between features (active elements) and parameters (passive elements).

be contained within a region and we do not need to check combinations of features that are not within a region.

Problem Decomposition and Verification Obligations: As the development of a system progresses, the abstract requirements themselves can be verified on the individual components of the system to ensure that they remain valid. In this way, our approach promotes a decomposition of the feature interaction problem into the following verification obligations: 1) check the regions that result from Morse for feature interactions; 2) ensure the abstract requirements input to Morse remain valid; and 3) ensure that no new influences are found so that the closed world assumption remains valid.

Implementation: To calculate the regions, we traverse the FR graph backwards from each node with no outgoing edges and from each strongly-connected component (SCC) of the graph that has no outgoing edges. Each of these will result in a region. By working backwards from the SCCs and ensuring that we do not visit a node more than once, we address challenge #3 of avoiding non-terminating analysis. Regions with only one feature are eliminated and nodes that do not have an influence path from a feature are removed.

B. Feature Relationship Graphs with Conditions

Next, we consider an FR graph with conditions on the nodes (from relevance requirements) and conditions on the edges (from influence requirements). There is at most one edge in a direction between any pair of nodes. A primitive FR graph is an FR graph with all edges labelled with True and all nodes labelled with True. Figure 3 shows a set of conditional relevance and influence requirements and their FR graph.

F3 is relevant only if $\neg b$
F2 influences F3 when $\neg b$
F2 influences F1 when $\neg a$
X influences F2 when b
X influences F1 when $\neg a$
F1 influences X
F1 influences Y when a

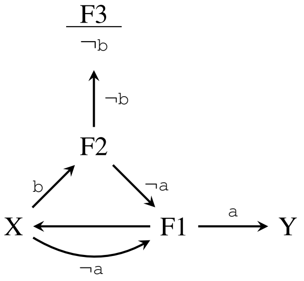


Fig. 3. Example FR Graph #2

An interpretation is a mapping from all the variables used in the conditions of the FR graph to values. A **situation** is a condition that describes a set of interpretations that satisfy it. Our goal is to find the parts of the graph that result under different interpretations and combine them appropriately. An interpretation reduces the FR graph by removing edges and nodes in the graph.

Definition 2: A **reduction** is the application of an interpretation, i, to an FR graph, g, that results in a primitive FR graph, g_i, by following these steps:

1) Label all edges and nodes in g whose conditions are True in the interpretation with True (or just drop the condition on the edge).
2) Remove all edges and nodes in g whose conditions are False in the interpretation.
3) Remove all of the incoming and outgoing edges of nodes in g that are removed.

Figure 4 shows the primitive FR graphs that result from the possible reductions under the four interpretations for the variables a and b used in the conditions of the FR graph of Figure 3. For each of these interpretations, we determine the regions of the primitive FR graph and then combine regions with the same nodes into a situation.

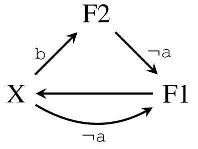
Definition 3: A **region**, g, **of an FR graph**, G, is a non-empty, full sub-graph of G, plus a situation, s, such that for every interpretation i, $i \in s$, the reduction of g under i results in a primitive FR graph, g_i, such that:

1) g contains all the nodes in g_i;
2) there exists a node, n, in g_i such that every other node in g_i has an influence path to n;
3) there is no node in $G \setminus g_i$ that has an influence path to node n in interpretation i, and there is no node in $G \setminus g_i$, such that n has an influence path to it in interpretation i;
4) g_i includes *at least two* feature nodes; and
5) g_i does not include a parameter node that does not have an influence path *from* a feature.

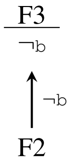
In addition, there is no interpretation j, $j \notin s$ such that the reduction of G under j has the same set of nodes as g (*i.e.,* the situation is maximal).

The regions for the FR graph of Figure 3 are:

1) F1, F2, X, situation $a \wedge b \vee \neg a \wedge b \vee \neg a \wedge \neg b \equiv \neg a \vee b$



2) F2, F3, situation $a \wedge \neg b \vee \neg a \wedge \neg b \equiv \neg b$



All nodes in a region are connected. Two regions cannot have the same FR graph, but one region can be a sub-graph of another region (because of different situations). Each region includes all the edges in the original FR graph between the nodes present in the region where the condition on the edge can be satisfied by *at least one* of the interpretations of the situation. But it excludes edges whose conditions cannot be satisfied by one of the interpretations of the situation. For example, even though Y is influenced by F2 and F1, there is no region that includes Y because the conditions on the edges F2-F1 and F1-Y cannot both be True at the same time. Therefore, there can never be a feature interaction at Y. Since the regions are determined from the primitive FR graphs resulting from reductions, a cycle in the FR graph does not change the definition of a region. The set of situations associated with all the regions will not necessarily cover all interpretations of the variables if there is an interpretation in which no feature interaction can occur.

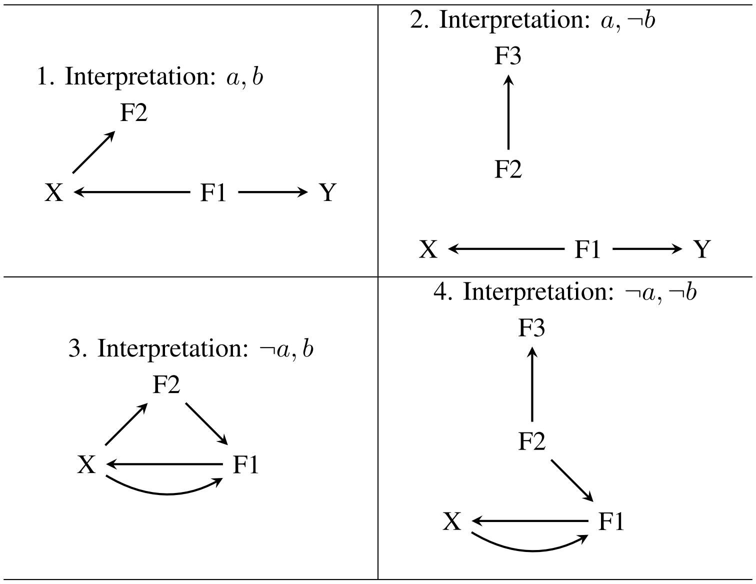

1. Interpretation: a, b
F2
X
F1
Y
2. Interpretation: $a, \neg b$
F3
F2
X
F1
Y
3. Interpretation: $\neg a, b$
F2
X
F1
4. Interpretation: $\neg a, \neg b$
F3
F2
X
F1

Fig. 4. Reductions of Fig 3

Problem Reduction: As for primitive FR graphs, every time we remove a feature from a combination that needs to be considered for feature interactions we get a substantial reduction in the size of the problem. With conditions, we have the additional reduction that the combination of feature behaviours only needs to be considered under a reduced set of interpretations.

Justification for Reduction: With conditions on the FR graph, a feature interaction is possible when there are two or more influence paths from different features to the same node *and all the conditions on the two paths are satisfiable together*. We justify our reduction with the argument that based on the definition of region there is not an interpretation in which there are two paths that start at different features that lead to the same node that are not both contained within a region, thus all feature interactions must be in a region and we do not need to check combinations of features that are not within a region.

Problem Decomposition and Verification Obligations: With situations, our verification obligations become: 1) check the regions that result from Morse for feature interactions *only under the interpretations included in the situation*; 2) ensuring the abstract requirements input to Morse remain valid; and 3) ensuring that no new influences are found so that the closed world assumption remains valid.

Implementation: We calculate the regions of an FR graph with conditions using backwards reachability from the terminal nodes and from the SCCs without outgoing edges. Each time we encounter a condition on a node or an edge, we branch to create regions for the positive and negative possibilities of the condition. The result of our algorithm is a set of subgraphs each with a set of conditions. Once we have concluded the backwards graph walk, we check the satisfiability of all the conditions in the set associated with each region using the SMT (Satisfiability Modulo Theories) solver Z3 [14]. If a set of conditions is not satisfiable that region is removed from the set of regions. By checking for satisfiability only at the end of the graph walking, we limit the number of computationally expensive satisfiability checks addressing challenge #2. If the set of conditions is satisfiable, the conjunction of the conditions in the set becomes the region's situation. If there are two regions with identical sets of nodes, we form a situation that is the disjunction of the situations of each of the regions, addressing challenge #1 so that we do not end up with an unmanageably large set of regions. In our examples so far, our implementation takes negligible time to run.

C. Unknown and Independent Features

We see a distinction between 1) an example where no influences are stated meaning all features can potentially interact and 2) an FR graph with conditional influences that has an interpretation in which the conditional edges are all False. In 2), there are always other interpretations in which there may be potential interactions. If there is no interpretation in which a condition can be True, it must be an erroneously stated influence.

Therefore, we consider what the role of an unconnected feature could be in an FR graph. There are two possible meanings: a) it has no influences (and would be in no region); or b) it may influence all other features because we lack information. We call a) an **independent feature** and b) an **unknown feature**. A feature that is not connected to another node in the FR graph must be labelled as unknown or independent. We have adopted syntactic conventions to designate whether a feature is unknown or independent. Unknown and independent features may have relevance requirements but are not involved in any influence requirements. We consider the notation for unknown features a syntactic shorthand for drawing edges to every other node, thus unknown and independent features do not require extensions to the definition of a region. An independent feature does not appear in any region. It is possible to use relevance requirements to control which nodes an unknown feature can influence. This technique is useful when a feature influences all but a few nodes in the FR graph and we want to avoid drawing too many edges.

V. Examples

Our technique is implemented in a tool called Morse. In this section, we describe two examples. Information used in these examples have been extracted from [11], and [15]. We focus on how much reduction in the feature interaction combination problem has been achieved using our tool.

A. Automotive Safety Features

In this section, we describe an extended version of the example presented in Section II based on the non-proprietary set of industrial automotive features called the "University of Waterloo Feature Model Set" (UWFMS) [11]. These "Active Safety" features are modelled in StateFlow [16] and use sensor inputs to control the vehicle's motion in order to improve the safety of its occupants. This example is composed of the four features of Table I (slightly extended) and the three following additional ones:

- **LG:** Lane Guide warns or assists the driver when unintentionally drifting from the lane.
- **PA:** Park Assist performs steering for a parallel park maneuver and requests gear changes when needed.
- **PSC:** Parking Space Centering moves the car to the center after a perpendicular parking maneuver.

Previously, Juarez-Dominguez [11] created and analyzed this set of features for feature interactions using model checking. She created a data flow graph (without loops) similar to a primitive FR graph. There are five system parameters as shown in Figure 5: brakes, throttle, and steering (which are direct actuators of the features), and speed and position of the vehicle (which are indirect actuators).

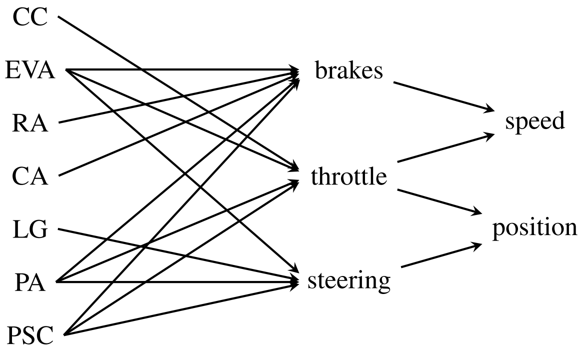


Fig. 5. UWFMS Primitive FR Graph (similar to Fig. 7.6 on p. 151 of [11])

Based on this primitive FR graph, she reduced the problem to three groups of features each of size four or five features (one group for each of throttle, brakes, and steering), where each group influenced the same parameter, ruling out feature combinations such as **CC** and **LG**. She then model checked all the possible pairs within these groups doing 22 model checking runs, which uncovered four pairs of features that had feature interactions, meaning they both influenced the same parameter at the same time: CA-EVA for the brakes, CC-EVA for the throttle and LG-EVA and EVA-PSC for the steering. Juarez-Dominguez's definition of an actual feature interaction is that two features both influence the same parameter and the difference between the effects on the parameters is within a certain threshold. She did not consider possible 3-way interactions.

We first compared our approach directly to Juarez-Dominguez's work, limiting ourselves to the direct actuators. We investigated whether there is subject knowledge that can reduce the number of model checking runs needed. After a quick examination the StateFlow models of the features, we extracted the brief abstract requirements found in Table III to input to Morse, which include the conditions under which a feature influences a parameter. Graphically, each Stateflow model was about a page in size so our abstract requirements are much more concise than a full behavioural model. By using our tool, Morse, we reduced the number of combinations to be verified for potential feature interactions to only six as described in Table IV. Our regions cover the four actual feature interactions found by Juarez-Dominguez, and the regions were found at much lower cost in human effort and computing time by using our methodology (*i.e.,* 22-6=16 model checking runs that were avoided). Our regions also allow for the possibility of a three-way feature interaction. Additionally, these abstract requirements would probably have been known by subject matter experts prior to creating the detailed Stateflow models.

Next, we added the following requirements about the parameters speed and position to Table III:

- The throttle influences the speed.
- The brakes influence the speed.
- The throttle influences the position.
- The steering influences the position.

Based on these abstract requirements, Morse reduces the feature interaction problem to the regions found in Table V, which we can compare to the features interactions found by Juarez-Dominguez (CC-CA, CA-EVA, and CC-EVA for the speed, and CC-LG for the position). Again, our regions cover the four actual feature interactions found by Juarez-Dominguez, and our methodology allow us to find these regions at low cost in human effort and computing time. Further knowledge by subject matter experts of *when* the throttle influences the speed, *etc.* could reduce the problem even further. Moreover, our regions also allow for the possibility of a three-way feature interaction. In addition, Morse provides results that allow an analyst to focus in on the situation in which an interaction could occur reducing the number of cases that need to be considered.

B. Airbus A320-211 Crash

In our second example, we used Morse to analyze the 1993 crash of an Airbus 320 [15]. A relevant factor in this crash was unintended effects of interactions between the different aircraft systems and the operation of the ground spoilers, reverse

TABLE III
ABSTRACT REQUIREMENTS FOR UWFMS

- CC is relevant only if the $\mathit{speed} > 0$.
- CC influences the throttle when $\mathit{speed} \geq 40$, $\mathit{gear} = \mathit{drive}$.
- EVA is relevant only if $\mathit{accel} < 30$.
- EVA influences the throttle when $\mathit{speed} \geq 0$, and $\mathit{gear} = \mathit{drive}$.
- EVA influences the brakes when $\mathit{speed} \geq 0$, and $\mathit{gear} = \mathit{drive}$.
- EVA influences the steering when $\mathit{speed} \geq 0$, and $\mathit{gear} = \mathit{drive}$.
- RA is relevant only if $\mathit{acc} \leq 0$.
- RA influences the brakes when $10 \leq \mathit{speed} \leq 25$ and $\mathit{gear} = \mathit{reverse}$.
- CA is relevant only if $\mathit{acc} < 35$.
- CA influences the brakes when $\mathit{speed} > 25$ and $\mathit{gear} = \mathit{drive}$.
- LG influences the steering when $\mathit{speed} \geq 40$, and $\mathit{gear} = \mathit{drive}$.
- PA influences the throttle when $\mathit{speed} = 0$, and $\mathit{gear} = \mathit{reverse}$.
- PA influences the brakes when $0 < \mathit{speed} \leq 5$, and $\mathit{gear} = \mathit{reverse}$ or $\mathit{gear} = \mathit{drive}$.
- PA influences the steering when $0 < \mathit{speed} \leq 5$, and $\mathit{gear} = \mathit{reverse}$.
- PSC influences the throttle when $0 < \mathit{speed} \leq 5$, and $\mathit{gear} = \mathit{drive}$.
- PSC influences the brakes when $0 < \mathit{speed} \leq 5$, and $\mathit{gear} = \mathit{drive}$.
- PSC influences the steering when $0 < \mathit{speed} \leq 5$, and $\mathit{gear} = \mathit{drive}$.

TABLE IV
UWFMS REGIONS COMPARED TO JUAREZ-DOMINGUEZ

Brakes	CA - EVA	$(\mathit{speed} > 25)$ *and* $(\mathit{gear} = \mathit{drive})$ *and* $(\mathit{acc} < 30)$
	PA - EVA - PSC	$(\mathit{speed} > 0)$ *and* $(\mathit{speed} \leq 5)$ *and* $(\mathit{gear} = \mathit{drive})$ *and* $(\mathit{acc} \leq 0)$
Throttle	CC - EVA	$(\mathit{speed} \geq 40)$ *and* $(\mathit{gear} = \mathit{drive})$ *and* $(\mathit{acc} < 30)$
	EVA - PSC	$(\mathit{speed} > 0)$ *and* $(\mathit{speed} \leq 5)$ *and* $(\mathit{gear} = \mathit{drive})$ *and* $(\mathit{acc} \leq 0)$
Steering	LG - EVA	$(\mathit{acc} < 30)$ *and* $(\mathit{speed}) \geq 40$ *and* $(\mathit{gear} = \mathit{drive})$
	EVA - PSC	$(\mathit{speed} > 0)$ *and* $(\mathit{speed} \leq 5)$ *and* $(\mathit{gear} = \mathit{drive})$ *and* $(\mathit{acc} \leq 0)$

TABLE V
UWFMS REGIONS WHEN INCLUDING SPEED AND POSITION PARAMETERS

Speed	CC - CA	$\mathrm{not}(\mathit{acc} < 30)$ *and* $(\mathit{speed} \geq 40)$ *and* $(\mathit{gear} = \mathit{drive})$ *and* $(\mathit{acc} < 35)$
	CA - EVA	$\mathrm{not}(\mathit{speed} \geq 40)$ *and* $(\mathit{speed} > 25)$ *and* $(\mathit{gear} = \mathit{drive})$ *and* $(\mathit{acc} < 30)$
	CC - EVA- CA	$(\mathit{gear} = \mathit{drive})$ *and* $(\mathit{speed} \geq 40)$ *and* $(\mathit{acc} < 30)$
	EVA - PSC- PA	$(\mathit{speed} > 0)$ *and* $(\mathit{speed} \leq 5)$ *and* $(\mathit{gear} = \mathit{drive})$ *and* $(\mathit{acc} <= 0)$
Position	CC - LG	$(\mathit{speed} \geq 40)$ *and* $(\mathit{gear} = \mathit{drive})$ *and* $\mathrm{not}(\mathit{acc} < 30)$
	CC- EVA - LG	$(\mathit{speed} \geq 40)$ *and* $(\mathit{gear} = \mathit{drive})$ *and* $(\mathit{acc} < 30)$
	EVA - PSC	$(\mathit{speed} > 0)$ *and* $(\mathit{speed} \leq 5)$ *and* $\mathit{gear} = 3$ *and* $(\mathit{acc} \leq 0)$

thrusters and brakes upon landing at Okecie International Airport in Warsaw Poland. As reported in [15]: "Very light touch of the runway surface with the landing gear and lack of compression of the left landing gear leg to the extent understood by the aircraft computer as the actual landing resulted in delayed deployment of spoilers and thrust reversers." We constructed our example on the following aircraft features:

- **Sp:** Spoilers are used during flight to increase the descent rate or to obtain more stability. They are also used during landing to increase drag.
- **Sl:** Slats increase lift allowing the plane to fly at a "slow" speed.
- **RT:** Reverse thrusters redirect the engine's thrust so that it is directed forward, rather than backward.
- **Fl:** Flaps, like the slats, increase lift to allow the plane to fly at "slow" speed.
- **Bk:** Brakes slow down the plane.

For the purpose of our study, we added the following two weather features: Wind and Rain. Our model has 20 abstract requirements, shown in Table VI (input to Morse), capturing the influences and relevancy of the features and parameters. In addition to describing relationships between aircraft systems, they also describe influences on parameters in the physical world, such as Lift and Unbalanced distribution of aircraft weight. Features and parameters are capitalized. Wheel speed is used as both part of the parameter, "Wheel speed less than 72kt at landing gears", and as a constant in a condition. Note that *kt* is the abbreviation of the nautical mile per hour unit.

Based on these abstract requirements, Morse reduces the feature interaction problem to the regions found in Table VII. When a weather feature, like wind or rain is included in a region it means that this meteorological situation is required for there to be a potential interaction. For example, the last region depicts a situation where the three features responsible for decreasing the velocity were less effective during bad meteorological conditions that yield the plane to be hydroplaning and unbalanced.

In addition to reducing the number of features to consider in combinations, most notably, these regions highlight critical situations when decreasing velocity upon landing where the features can interact and affect the software's understanding of the flight state (*i.e.,* flying vs. on-ground). A better understanding of these situations by engineers responsible for ensuring the airworthiness of this aircraft might have helped avoid this catastrophe.

VI. Related Work

There is a variety of definitions of a "feature" (*e.g.,* [1], [3], [17]). Our method does not rely on any particular definition of a feature; we leave it to the subject matter expert to decide what constitutes a feature. The feature interaction problem was first studied in telephony (*e.g.,* [1][18]), but has been recognized in other domains, such as internet applications [19], embedded systems [20], security [21], and the automotive domain [22]. The most common definition of a feature interaction is from Cameron and Velthuijsen [3] where a feature interaction means that a feature behaves differently in the presence of other features than it behaves by itself (which could be desirable or undesirable). For such a feature interaction to occur there must be a situation where two features can influence the same parameter (or each other directly), which is what our analysis searches for as a potential feature interaction.

Approaches to handling the feature interaction problem were categorized as detection, resolution, and avoidance by Cameron and Velthuijsen [3]. There has been considerable work on detecting feature interactions through a variety of static analysis (off-line) methods such as model checking (*e.g.,* [5], [23]) and other formal methods (*e.g.,* [6]). The abstract models of Felty and Namjoshi [7] are almost as abstract as our abstract requirements but include a temporal element so that a model of feature behaviour is created, and they use model checking to find actual feature interactions. Our method is complementary to these approaches because, with little effort, it narrows down the possible feature combinations to consider for detection of actual feature interactions.

Resolving feature interaction at run-time can be done using a feature interaction manager (*e.g.,* variable-specific resolutions [8]). Avoidance of feature interactions can be accomplished through architectures that automatically resolve the interaction through the organization of features in the system (*e.g.,* Distributed Feature Composition (DFC) [10]). Our method could be complementary to feature interaction managers and architectures by reducing the set of features to monitor for feature interactions at run-time.

Without conditions, our method is similar to slicing [13] a data dependence graph based on its terminal points (and cycles). The two prior efforts that are most closely related to our work (but do not use conditions) are that of Juarez-Dominguez [11], which we compared to directly in a case study and Metzger [20]. Juarez-Dominguez used a primitive FR graph without loops to limit the number of pairwise feature combinations of state machine models of features to model check for feature interactions. The model checking produces actual cases of feature interactions. We use conditions in the FR graph (which may have loops) to reduce the number of feature combinations and situations that need to be checked for feature interactions. We work with much less detailed models and much more lightweight analysis to gain our reduction results. Metzger [20] used similar ideas to our primitive FR graph (but without loops) to describe relationships among requirements to determine locations of potential feature interactions. To refine these potential interactions, he created refined data models or usage types. We use conditions and stay at an abstract level of requirements to reduce the feature interaction problem. FR graphs are similar to the proposed framework of Classen et al. [4] that extends Jackson's problem frames [24]. However, FR graphs are data flow diagrams and not control flow diagrams. Our approach looks at the situation in which interactions could occur and not how such interactions could occur.

VII. Conclusions and Future Work

We have presented a method and tool to use abstract subject matter knowledge to deduce why some features will NOT interact and reduce the combinations of features and situations that have to be evaluated or resolved for feature interactions. Our novel contribution is in using conditions on the influences and relevance of nodes expressed abstractly in order focus on situations in which a feature interaction might occur and rule out situations that cannot occur. We described a simple language for writing this subject matter knowledge as abstract requirements. We used these requirements in a graph walking algorithm connected to an SMT solver to reduce the feature interaction problem to sets of features each with an attached situation that require further analysis. Our method handles loops in the FR graph; groups related potential feature interactions together to avoid ending up with an unmanageably large set of regions; and limits the number of computationally expensive satisfiability checks. Our examples show interesting results in terms of reducing the feature interaction combinatorial explosion and in focusing attention to particular situations more analysis is needed.

TABLE VI
ABSTRACT REQUIREMENTS FOR AIRBUS A320-211 CRASH

- Sp is relevant only if (false indication on landing or wheel speed $\leq$ 72 or in flight)
- RT is relevant only if false indication on landing
- Landing gear is relevant only if false indication on landing and not in flight
- Landing gear influences Sp
- Landing gear influences RT
- Wheel speed less than 72kt at landing gears influences Sp when wheel speed $\leq$ 72 and not in flight
- Wind influences Lift when in flight
- Sp influences Lift when in flight
- Sl influences Lift when in flight
- Fl influences Lift when in flight
- Wind influences Unbalanced distribution of aircraft weight when false indication on landing
- Unbalanced distribution of aircraft weight influences Landing gear when false indication on landing
- RT influences Velocity not decreasing when false indication on landing
- Sp influences Velocity not decreasing when (false indication on landing or wheel speed $\leq$ 72) and not in flight
- Rain influences Hydroplaning
- Hydroplaning influences Wheel speed less than 72kt at landing gears
- Hydroplaning is relevant only if wheel speed $\leq$ 72
- Hydroplaning influences Bk
- Bk influence insufficient primary braking when wheel speed $\leq$ 72
- Insufficient primary braking influences Velocity not decreasing

TABLE VII
AIRBUS A320-211 CRASH REGIONS

Lift	Wind - Sp - Sl - Fl	*in flight*
Velocity	Rain - Bk - Sp	*not in flight and wheel speed* $\leq$ 72 *and not false indication on landing*
	Wind - RT - Sp	*not in flight and not wheel speed* $\leq$ 72 *and false indication on landing*
	Rain - Wind - Bk - Sp - RT	*not in flight and wheel speed* $\leq$ 72 *and false indication on landing*

We found it remarkable that from the limited set of vague statements we worked with, we deduce very useful information to reduce the feature interaction problem quickly. These statements are requirements of the individual features and become a strategy for decomposing the verification problem. As long as these requirements remain valid (with common definitions of relevance and influence across all features) and no new relevance or influence requirements are introduced then our reduction of the feature interaction problem is sound. Our method is independent of the logic used to write the conditions since an alternative satisfiability checker can be used.

Our analysis reduces the feature interaction problem but does not conclude that a feature interaction does occur in the region, which depends on the behaviour of the feature. An important element of Morse is its ability to yield useful analysis results without providing complete models of the feature behaviour. Our results are conservative in that they do not take into account the temporal order of the influences. The conditions under which one feature influences a node may not be true at the same time that another feature influences the node, but our analysis cannot rule out this possibility.

It is also important to recognize that the omission of certain kinds of information could adversely affect the accuracy of our analysis results. We must include all potential influences of a feature on other features and parameters, and the conditions must be accurate for our results to be valid. However, some kinds of information about the system can be omitted without an adverse effect on the accuracy of our analysis results. The absence of an explicit condition on an influence or node means it is applicable in all situations. Omission or weakening of a condition on an influence or node might yield less precise results, but will not yield inaccurate results.

We plan to continue our work in the following directions:

- Automatically extracting our abstract requirements from Stateflow or other types of feature models to reduce the search for feature interactions in these models;
- Study the applicability of the method under the closed world assumption in an industrial context;
- Presenting the argument regarding why a feature interaction cannot occur in certain situations as a logical argument that can be checked by a proof checker and used as verification obligations; and
- Exploring the decomposition providing by our method as downstream development verification obligations.

References

[1] T. F. Bowen, F. S. Dworack, C. H. Chow, N. Griffeth, G. E. Herman, and Y. J. Lin, "The feature interaction problem in telecommunications systems," in *Seventh International Conference on Software Engineering for Telecommunication Switching Systems*, pp. 59–62, Jul 1989.

[2] S. Apel, J. M. Atlee, L. Baresi, and P. Zave, "Feature Interactions: The Next Generation (Dagstuhl Seminar 14281)," *Dagstuhl Reports*, vol. 4, no. 7, pp. 1–24, 2014.

[3] E. J. Cameron and H. Velthuijsen, "Feature interactions in telecommunications systems," *IEEE Communications Magazine*, vol. 31, pp. 18–23, Aug 1993.

[4] A. Classen, P. Heymans, and P.-Y. Schobbens, "Whats in a feature: A requirements engineering perspective," in *International Conference on Fundamental Approaches to Software Engineering*, pp. 16–30, Springer, 2008.

[5] P. K. Au and J. M. Atlee, "Evaluation of a State-Based Model of Feature Interactions," *Feature Interactions in Telecommunications Networks IV*, pp. 153–167, 1997.

[6] M. Frappier, A. Mili, and J. Desharnais, "Defining and detecting feature interactions," in *Algorithmic Languages and Calculi: IFIP TC2 WG2.1 International Workshop on Algorithmic Languages and Calculi*, pp. 212–239, Springer US, 1997.

[7] A. P. Namjoshi, Felty and K. S. Namjoshi, "Feature Specification and Automated Conflict Detection," *ACM Transactions on Software Engineering and Methodoloy*, vol. 12, no. 1, pp. 3–27, 2003.

[8] C. Bocovich and J. M. Atlee, "Variable-specific Resolutions for Feature Interactions," *Foundations of Software Engineering*, pp. 553–563, 2014.

[9] K. C. Wong, J. G. Thistle, and R. P. Malhame, "Conflict resolution with flexible priority in modular control," in *Canadian Conference on Electrical and Computer Engineering*, vol. 2, pp. 797–800, Sep 1995.

[10] M. Jackson and P. Zave, "Distributed feature composition: a virtual architecture for telecommunications services," *IEEE Transactions on Software Engineering*, vol. 24, pp. 831–847, Oct 1998.

[11] A. L. Juarez Dominguez, *Detection of Feature Interactions in Automotive Active Safety Feature*. PhD thesis, University of Waterloo, David R. Cheriton School of Computer Science, 2012.

[12] R. P. Kurshan, *Computer-Aided Verification of Coordinating Processes: The Automata- Theoretic Approach*. Princeton University Press, 1995.

[13] M. Weiser, "Program slicing," *IEEE Transactions on Software Engineering*, vol. SE-10, pp. 352–357, July 1984.

[14] L. M. de Moura and N. Bjørner, "Z3: an efficient SMT solver," in *Tools and Algorithms for the Construction and Analysis of Systems*, vol. 4963 of *Lecture Notes in Computer Science*, pp. 337–340, 2008.

[15] Main Commission Aircraft Accident Investigation Warsaw, "Report on the Accident to Airbus A320-211 Aircraft in Warsaw." http://www.rvs.uni-bielefeld.de/publications/Incidents/DOCS/ComAndRep/Warsaw/warsaw-report.html. transcripted by Peter Ladkin, 6 March 1996.

[16] "MathWorks Stateflow." http://www.mathworks.com/help/stateflow/.

[17] C. R. Turner, A. Fuggetta, L. Lavazza, and A. L. Wolf, "A conceptual basis for feature engineering," *Journal of Systems and Software*, vol. 49, no. 1, pp. 3 – 15, 1999.

[18] D. Amyot, T. Gray, R. Liscano, L. Logrippo, and J. Sincennes, "Interactive conflict detection and resolution for personalized features," *Journal of Communications and Networks*, vol. 7, no. 3, pp. 353–365, 2005.

[19] R. G. Crespo, M. Carvalho, and L. Logrippo, "Distributed resolution of feature interactions for internet applications," *Computer Networks*, vol. 51, no. 2, pp. 382–397, 2007.

[20] A. Metzger, "Feature interactions in embedded control systems," *Computer Networks*, vol. 45, no. 5, pp. 625–644, 2004.

[21] A. Nhlabatsi, R. Laney, and B. Nuseibeh, "Feature interaction: The security threat from within software systems," *Progress in Informatics*, no. 5, pp. 75–90, 2008.

[22] A. L. Juarez Dominguez, N. A. Day, and J. J. Joyce, "Modelling feature interactions in the automotive domain," in *International Workshop on Modeling in Software Engineering*, pp. 45–50, ACM, 2008.

[23] A. Classen, P. Heymans, P.-Y. Schobbens, and A. Legay, "Symbolic model checking of software product lines," *International Conference on Software Engineering*, pp. 321–330, 2011.

[24] M. Jackson, *Problem Frames: Analyzing and Structuring Software Development Problems*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001.