Two Techniques to Optimize the Detection of False-Optional Features in Propositional Feature Models

Wei Zhang, Li Yi, Haiyan Zhao, Zhi Jin, and Hong Mei

**Abstract**—Feature models provide an approach to organizing and reusing reusable requirements in a software product line. Propositional feature models (PFMs) are a basic kind of feature model, which can be practically transformed into propostional formulas. For this reason, the analysis of PFMs usually can be transformed into the analysis of the corresponding propostional formulas, and more particularly, into satisfiability (SAT) problems related to the transformed formulas. This paper is concerned with the optimized detection of *false-optional* features in PFMs. A false-optional feature is an optional feature *in syntax*, but actually a dead feature or a mandatory feature *in semantics*. Our research focuses on two optimizing goals: the *scale-decrease* goal (i.e. to decrease the scale of the SAT problems reflecting whether a feature is false-optional), and the *invoking-times-decrease* goal (i.e. to detect all false-optional features in a PFM by invoking a SAT-Solver as few times as possible). Based on the two optimizing goals, we proposed two techniques: *atomic sets*, and *reasoning-based detection*. The former technique contributes to the *scale-decrease* goal, by mergeing a mandatory feature with its parent; the latter technique contributes both to the *invoking-times-decrease* goal and the *scale-decrease* goal, by integrating a reasoning-based approach into the the detection of false-optional features, and by removing those reasoning-decidable features from a PFM, respectively. The correctness and effectiveness of the two techniques are formally proved and analyzed.

**Index Terms**—Requirements analysis, reuse model, verification.

xxxx-xxxx/0x/$xx.00 © 200x IEEE

————————————————

1. F.A. Author is with the National Institute of Standards and Technology, Boulder, CO 80305. E-mail: author@ boulder.nist.gov.
2. S.B. Author Jr. is with the Department of Physics, Colorado State University, Fort Collins, CO 80523. E-mail: author@colostate.edu.
3. T.C. Author is with the Electrical Engineering Department, University of Colorado, Boulder, CO 80309. On leave from the National Research Institute for Metals, Tsukuba, Japan E-mail: author@nrim.go.jp.

*Manuscript received (insert date of submission if desired). Please note that all acknowledgments should be placed at the end of the paper, before the bibliography.*

—————————— ◆ ——————————

# 1 Introduction

F

EATURE models provide an approach to *organizing* and *reusing* reusable requirements in a software product line (SPL). The organizing responsibility is achieved by partitioning reusable requirements into a set of features and explicitly modeling constraints among features. The reusing responsibility is carried out through configuration – that is, selecting a subset of features from a feature model, while ensuring no violation of any constraint between/among features.

The concept of feature models is first proposed in the FODA methods. After that, several variants of feature models are proposed. For example, 1, tree🡪DAG; 2. Cardinality; 3. Attribute feature model.

In this paper, we focus on a *basic* kind of feature model, called *propositional* feature models (PFMs). This kind of feature model is basic, because most of the existing variants of feature models are based on PFMs (either by relaxing some constraints in PFMs or by adding some new constructs into PFMs). This kind of feature model is propositional, because a PFM can be practically and easily transformed into propostional formulas.

In particular, this paper is concerned with the optimized detection of false-optional features in a PFM. A false-optional feature is an optional feature (i.e. a feature that can either be selected or removed when its parent feature is selected) *in syntax*, but actually a dead feature (i.e. a feature that can not be selected when its parent is selected) or a mandatory feature (i.e. a feature that can only be selected when its parent is selected) *in semantics*. False-optional features are a kind of deficiency of a PFM, and the existing of false-optional features usually indicates the existing of semantic errors in a PFM. If false-optional features are not detected explicitly, the indicated errors will not be uncovered and then be transmitted implicitly into the subsequent core-asset and product developing activities in SPL engineering.

Fig. 1 shows the structure of this paper in an abstract level. The rest of this section will give a brief introduction to the elements and their relationships in this structure.

文章结构

Fig. 1. An abstract-level structure of this paper. To deal with the two *difficulities* in the *problem* of efficiently detecting all false-optional features in a PFM, this paper sets two *optimizing goals*, and proposes two *techniques*, correspondingly.

The key problem of detecting false-optional features is not whether they can be detected or not, but rather whether they can be detected *efficiently*. To detect false-optional features, we can simply traverse each optional feature in a PFM, and check whether it is false-optional. To detect them efficiently, howerver, there are two difficulities:

1. *Efficient feature-level checking*: The difficulty in checking whether a single optional feature is false-optional *efficiently*, and
2. *Efficient model-level detecting*: The difficulty in detecting all false-optional features in a PFM *efficiently*.

The first difficulty lies in the fact that checking whether a feature is false-optional involves serval satisfiability (SAT) problems; the NP-complete nature of SAT problems determines the difficulty. The second difficulty emerges when a PFM contains a large number of optional features; in such a situation, it is time-consuming and inefficient to detect all flase-optional features in a PFM by using a simple traversal-based approach to each feature in the PFM.

To deal with the two difficulties above, our research in this paper focuses on two optimizing goals: the *scale-decrease* goal, and the *invoking-times-decrease* goal. The former goal deals with the first difficulty, and aims to decrease the scale of the SAT problems reflecting whether a feature is false-optional, as small as possible. The latter goal deals with the second difficulty, and aims to completely detect all false-optional features in a PFM by invoking a SAT-Solver as few times as possible.

To operationalize the two optimizing goals above, we propose two techniques (the main contributions of this paper): *atomic sets*, and *reasoning-based detection*. The former technique contributes to the *scale-decrease* goal, by mergeing a mandatory feature with its parent and therefore decreasing the number of propositional variables in the related SAT problems. The latter technique contributes both to the *invoking-times-decrease* goal and the *scale-decrease* goal, by integrating a reasoning-based approach into the detection of false-optional features, and by removing those reasoning-decidable features from a PFM, respectively. We provide formal proofs and systematic analysis to the correctness and effectiveness of the two techniques.

The rest of this paper is organized as follows. Section 2 provides some preliminaries about prositional feature models (PFMs) and false-optional features. Section 3 presents the *atomic sets* technique and the *reasoning-based detection* technique, with the formal proofs of their correctness. Section 4 gives an analysis to the effectiveness of the two techniques by applying them to a set of public-available feature models. Section 5 discusses some issues in our research. Seection 6 introduces related work to this paper. Finally, Section 7 concludes this paper with a short summary.

# 2 Preliminaries

## 2.1 A Metamodel of PFMs

Fig. 2 shows a metamodel of PFMs following the UML notation. Generally, a PFM consists of a set of features and relations between/among features, and satisfies a set of OCL constraints.

FM Metamodel

Fig. 2. A metamodel of PFMs. A PFM consisits of a set of featues, a *refinement* relation between features, and a *constraint* relation among features. There are four typical kinds of *constraint* relation, namely, *requires*, *excludes*, *OR*, and *XOR*. In addition, a feature model also satisfies 10 kinds of OCL constraint (an exception is the 10th constraint, in which, we use “c.formula[f\f.binding-state, f∈c.involve]” to denote “the replacement of each feature in a propositional formula with the feature’s binding-state”; such a replacement syntax is not provided in OCL).

In a PFM, a feature has four attributes:

1. *Name*: a naming string of a feature.
2. *Description*: one or more paragraphs of text, describing the requirements denoted by a feature.
3. *Binding-state*: a three-valued Boolean, denoting whether a feature is selected (*true*), removed (*false*), or to be decided (*unknown*) in a configuration.
4. *Optional*: a Boolean, denoting whether a feature can be removed (*true*) or not (*false*), when its parent feature is selected. A feature with a *true*/*false* value about this attribute is simply called an *optional*/*mandatory* feature.

There are two relations between/among features: *refinement* and *constraint*. Through the refinement relation, features in a PFM form tree structures. That is, a feature, as the *parent* role, can be refined to zero, one or more children features, and as the *child* role, can be refined from zero or one feature.

The constraint relation captures the propostional constraints among features. All constraints have two attributes:

1. *Formula*: a propostional formula that capture the propositional constraints among features. Without loss of generality, we assume that the formula of a constraint is in the conjunction normal form (CNF).
2. *State*: a three-valued Boolean, denoting whether a constraint is satisfied (*true*), violated (*false*), or to be decided (*unknown*).

Table 1 gives explanations of the set of OCL constraints satisfied by a feature model.

TABLE 1  
Explanations of OCL Constraints

| ID | Explanation |
| --- | --- |
| **①** | A feature model should contain exactly one root feature. Usually, the root feature has the same name with the software product line under consideration. |
| **②** | The root feature is a *mandatory* feature and should always be *selected*. The underlying principle is that any product of a software product line should contain at least one feature; if the root feaure is *optional* and can be *removed*, an empty product could be customized from a feature model by simply removing the root feature and all its offspring features. |
| **③** | In a *refinment* relation between two features, if the *child* feature is *selected*, the *parent* feature should also be *selected*. |
| **④** | In a *refinment* relation, if the *child* feature is *mandatory*, this feature must be *selected* when the *parent* feature is *selected*. |
| **⑤** | In a *requires* constraint between two features, if the *requirer* feature is *selected*, the *requiree* feature should also be *selected*, in order not to violate this constraint. |
| **⑥** | In an *excludes* constraint between two features, at most one feature can be *selected*, in order not to violate this constraint. |
| **⑦** | In a *VP-variant-constraint*, each *variant* feature should also be a *child* of the *variation-point* feature. That is, a *VP-variant-constraint* is a kind of constraint between a feature and its children, in which, the *parent* feature plays the role of *variation-point*, and each involved *child* plays the role of *variant*. |
| **⑧** | In an *OR* constraint, if the *variation-point* feature is *selected*, at least one of the *variant* features should be *selected*, in order not to violate this constraint. |
| **⑨** | In an *XOR* constraint, if the *variation-point* feature is *selected*, exactly one of the *variant* features should be *selected*, in order not to violate this constraint. |
| **⑩** | For a general constraint among features, its state equals to the value of its formula by substitute each feature in this formula with this feature’s *binding-state* value. In any vaid configuration of a PFM, the state of a constraint should not be false, i.e. the constraint should be satisfiable. |

FM Example

Fig. 3. An illustrative feature model of the *mobile-phone service* software product line, and a valid configuration of it. In this configuration, all appeared features are selected, and all other features are removed.

Fig. 3 shows an illustrative example of PFMs, in a software product line called *mobile-phone service*, and a valid configuration of it.

It should be pointed out that, in the metamodel of PFMs (see Fig. 2), we make a unified definition of both PFMs and their valid configurations: any valid configuration of a PFM is also a PFM itself. The realtion between a PFM and a valid configuration of it is *speclization*. That is, a valid configuration inherits all semantic information from the PFM, and adds further constrants into the PFM (i.e. constraining some features’ binding-states from *to be decided* to *removed* or *selected*).

## 2.2 Transform PFMs into Propositional Formulas

TABLE 2 shows the 9 rules to transform elements in a PFM into the corresponding propositional formulas. For the sake of simplicity, in the target formulas, we refer to a feature’s binding-state just as its name (this simplification will be held in the following of this paper, when no confusion arises). Since most of these rules (excluding the last two obvious rules) are generated from the OCL defined in the metamodel (see Fig. 2), we will not give explanations to them individually. In addition, we assume the constraints other than *requires*, *excludes*, *OR*, *XOR* are CNF formulas by default, and therefore not need to be transformed anymore.

TABLE 2  
Transformation Rules

| Rule Name | Original  OCL  Constraints | | Source  Elements | Target  Propositional Formula |
| --- | --- | --- | --- | --- |
| Root-Feature | ② | | rule-root feature | r |
| Optional-Refines | ③ | | rule-optional refines | ¬b∨a |
| Mandatory-Refines | ③ ④ | | rule-mandatory refines | (¬b∨a)∧(¬a∨b) |
| Requires | ⑤ | | rule-requires | ¬a∨b |
| Excludes | ⑥ | | **rule-excludes** | ¬a∨¬b |
| OR-Structure | ⑧ ③ | | **rule-or** | (¬p∨c1∨c2⋅⋅⋅∨cn) ∧  rule-or-formula |
| XOR-Structure | ⑨ ③ | | **rule-xor** | (¬p∨c1∨c2⋅⋅⋅∨cn) ∧  rule-xor-formula ∧  rule-or-formula |
| Selected Feature |  | | **rule-selected features** | f |
| Removed Feature |  | | rule-removed features | ¬f |
|  | | | | |
| Notation | | Meaning | | |
| notation-feature | | A feature, which has the name *f* and can either be mandatory or optional (that is, we do not care whether it is mandatory or optional). | | |
| A feature’s name in a propositional formula | | The feature’s binding-state (for the sake of simplicity). | | |

A common characteristic of these rules is that the target formula is either a single propositional clause (i.e. a disjunction of propositional variables) or a conjunction of a set of propositional clauses. In both of the two cases, the target formula is a formula in CNF.

In the following, we use *CST* to denote the conjunction of all the propositional formulas derived from a PFM (since each of these formulas is in CNF, the conjunction of them also is a formula in CNF), and *Ci* an individual clause in the conjunction. Therefore, the following relation holds:

*CST* = *C1*∧*C2*∧⋅⋅⋅*Ci*⋅⋅⋅∧*Cn*.

According to the PFM metamodel in Fig. 2 (in particular, the 10th OCL constraint), in any valid PFM, the formula *CST* should be satisfiable.

## 2.3 False-Optional Features

A *false-optional* feature is an optional feature (i.e. a feature that can either be selected or removed when its parent feature can be and is selected) *in syntax*, but actually a dead feature (i.e. a feature that can only be removed when its parent can be and is selected) or a mandatory feature (i.e. a feature that can only be selected when its parent can be and is selected) *in semantics*. In contrast to false-optional features, a *true-optional* feature is an optional feature both in syntax and in semantics. Formal definitions of false-optional and true-optional features are listed as follows.

**Definition 1.** (false-optional/true-optional features)

*Given a propostional feature model PFM, the formula CST tramsformed from PFM, an optional feature c in PFM, and the parent feature p of c, c is a false-optional feature, iff at lest one of the following two properties is unsatisfied; c is a true-optional feature, iff both of the following two properties are satisfied.*

1. (*CST*∧*p is satisfiable*)→(*CST*∧*p*∧*c is satisfiable*)*.*
2. (*CST*∧*p is satisfiable*)→(*CST*∧*p*∧¬*c is satisfiable*)*.*

TABLE 3 gives some illustrative examples of false-/true-optional features, which involves three simple PFMs (see the column heads), and three possible configurations of them (see the row heads). The commonality of the three PFMs is their contained features and the refinment relation between features, and the difference is the constraints they have. The content of a cell shows whether a configuration is valid for a PFM. In all of the three PFMs, feature *a* is true-optional, because that *a* can either be selected or removed when its parent *r* is selected. Similarly, feature *b* in *PFM 1* is also true-optional. However, feature *b* in *PFM 2* is false-optional because it can not be removed when its parent is selected, and feature *b* in *PFM 3* is false-optional because it can not be selected when its parent is selected.

TABLE 3  
Examples of False-/True-Optional Features

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| PFM  Conf. | | | 1 | 2 | 3 |
| fo-PFM1 | fo-PFM2 | fo-PFM3 |
| 1 | fo-conf 1 | | valid | valid | valid |
| 2 | fo-conf 2 | | valid | *invalid* | valid |
| 3 | fo-conf 3 | | valid | valid | *invalid* |
|  | | | | | |
| Notation | | Meaning | | | |
| notation-selected feature | | A selected feature (i.e. its *binding-state* is *true*) | | | |
| notation-removed feature | | A removed feature (i.e. its *binding-state* is *false*) | | | |

False-optional features are a kind of deficiency of a PFM, and the existing of false-optional features usually indicates the existing of semantic errors in a PFM. If false-optional features are not detected explicitly, the indicated errors will not be uncovered and then be transmitted implicitly into the subsequent core-asset and product developing activities in SPL engineering. Therefore, before a PFM is used to develop core-asset or configure products in a SPL, we should ensure that there are no false-optional features in the PFM.

In theory, to detect false-optional features in a PFM, we can simply traverse each optional feature in the PFM, and check whether an optional feature is false-optional according to Definition 1 and by invoking a *SAT-Solver* (i.e. a software program that accepts a CNF propostional formula, and outputs a Boolean value denoting whether this propostional formula is satisfiable or not).

# 3 Two Techniques to Optimize the Detection of False-Optional Features

In this section, we first introduce the two difficulities in efficient detection of false-optional features, and two corressponding optimizing goals. After that, we present two techniques to optimize the detection of false-optional features, respectively. The two techniques are *atomic sets* and *reasoning-based detection*. The former technique optimizes the detection by decreasing the number of variables in the SAT problems that reflect whether a feature is false-optional or not. The latter technique optimizes the detection by integrating a reasoning-based approach into the detection and by removing those reasoning-decidable features from a PFM.

## 3.1 Two Diffuculities and Two Optimizing Goals.

To detect false-optional features *efficiently*, there are two basic difficulities:

1. *Efficient feature-level checking*: The difficulty in checking whether a single optional feature is false-optional *efficiently*. According to Definition 1, such a checking can be transformed into three SAT problems related to the formula *CST*. With the scale increasing of *CST*, in the worst case, the time to resolving these SAT problems increases exponentially.
2. *Efficient model-level detecting*: The difficulty in detecting all false-optional features in a PFM *efficiently*. When a PFM contains a large number of optionals features, it is time-consuming to detect all flase-optional features by using such a simple traversal-based approach as described in the last paragraph of Section 2.3.

To deal with the two difficulties above, in our research, we set two optimizing goals:

1. The *scale-decrease* goal, which deals with the difficulty of *efficient feature-level checking*, and aims to decrease the scale of the formula *CST* of a PFM as small as possible.
2. The *invoking-times-decrease* goal, which deals with the difficulty of *efficient model-level detecting*, and aims to detect all false-optional features in a PFM by invoking a SAT-Solver as few times as possible.

## 3.2 Technique 1: Atomic Sets

An *atomic set* is a set of features, which can be safely treated as a whole in the detection of false optional features. The idea of *atomic sets* comes from the following observation:

**Lemma 1.** *Given a propositional feature model PFM, the formula CST, a feature p in PFM, and a mandatory child feature c of p, then the two features p and c mutually require each other, i.e. CST* → *(*¬*p*∨*c)*∧*(*¬*c*∨*p)*.

**Proof.** From the *mandatory-refines* rule in TABLE 2. 

The idea of *atomic sets* is that since a parent feature and any of its mandatory child features mutually requires each other, we can safely treat the two features as a whole, from the viewpoint of feature model configuration (that is, if you select/remove one of them, then the other feature should also be selected/removed). Such a kind of treatments between two features has the possibility of being composed together. For examples, if a feature *p* has two mandatory child features *c1* and *c2*, then the three features can be treated as a whole; if *p* is mandatory and has its parent feature *a*, then *a* can be added into the whole; if *ci* has one or more mandatory child features, these child features can also be added into the whole. Based on the above analysis, we get the concept of *atomic sets*.

**Definition 2.** (atomic sets)

*Given a propostional feature model PFM, and a feature set AS in PFM, AS is an atomic set, iff it satisfies the following two properties:*

1. *For any two features a and b in AS (if has), they satisfiy either of the following two properties:*
2. *There is a mandatory refinement path between a and b.*
3. *There are a feature c in AS and two mandatory refinement paths between a and c, and between b and c, respectively.*
4. *For any feature c not in AS, there is no mandatory refinement path between c and any feature in AS.*

*[Here:*

***A******mandatory refinement path*** *is a sequence of features that satisfies the following two properties:*

* *For any two adjacent features <p, c> in the sequence, p is the parent of c.*
* *Excluding the first feature, all the other features in the sequence are mandatory features (the first feature can either be mandatory or optional).*

*A* ***mandatory refinement path between two features*** *means that the two features appear in the mandatory refinement path.]*

The left part in Fig. 4 a. shows a PFM (the constraints are omitted since they are irrelevant to the identification of atomic sets) and the five atomic sets in this PFM.

Generally, the atomic sets in a PFM hold a set of obvious properties (listed in Lemma 2).

**Lemma 2.** *A PFM holds the following properties about atomic sets*:

1. *For any two features a and b in an atomic set, the two features mutually require each other, i.e. CST* → *(*¬*a*∨*b)*∧*(*¬*b*∨*a).*
2. *Features in an atomic set form a tree by the refinement realtion between these features. (For simplicity, given a atomic set AS, we use AS.root to denote the root feature of the tree)*
3. *Excluding the root atomic set, all the root features of the other atomic sets in a PFM are optional fatures; the root feature of the root atomic set is the root feature of a PFM, and therefore a mandatory feature.*
4. *All the atomic sets in a FPM form a partition of features in the PFM – that is, each feature in a PFM belongs exactly to one atomic set. (For simplicity, given a feature f, we use f.AS to denote the atomic set to that f belongs)*
5. *The number of atomic sets in a PFM is equal to the number of optional features in the PFM plus 1 (the 1 represents the root atomic set).*

*[Here:*

***The root atomic set*** *in a PFM is the atomic set that contains the root feature of the PFM.*

***The root feature of an atomic set*** *is the root of the tree formed by features in this atomic set and the refinement relation between them.]*

**Proof.** From the metamodel of PFMs and the definition of atomic sets (details are omitted). 

Example - atomic sets

Fig. 4. An illustrative example of atomic-sets based reduction of PFMs. The left part shows a PFM (constraints are omitted) and the 5 atomic sets in this PFM. The right part shows the reduced PFM, in which, each atomic set in the original PFM is denoted by the root feature of this atomic set. In this example, the number of features is reduced from 14 to 5.

The concept of atomic sets can be used to transform a PFM into a new PFM with few features (Fig. 4 gives an illustrative example of such a transformation), while all the false-optional features in the original feature model are preserved in the reduced PFM. The formal definition of the reduced PFM and the *false-optional-preserving* property are given in Definition 3 and Theorem 1, respectively.

**Definition 3.** (atomic-set-reduced PFMs)

*Given two propostional feature models PFMO and PFMR, PFMR is an atomic-set-reduced PFM of PFMO, iff the two feature models satisfy the following properties:*

1. *Excluding the root feature, all the other features in PFMR are optional features.*
2. *There is a one-to-one mapping* ***FMap*** *between atomic sets in PFMO and features in PFMR.*
3. *There is a one-to-one mapping* ***RMap*** *between the refinement realtion between features in different atomic sets in PFMO and the refinement relation in PFMR, and in this one-to-one mapping, every element ((pO, cO), (pR, cR)) satisfies*

*(pO.AS, pR)∈FMap ∧ (cO.AS, cR)∈FMap.*

1. *Suppose CSTO and CSTR are the constraints in PFMO and PFMR, respectively, and suppose CSTM is the constraints derived by replacing each feature fO in CSTO with the feature fR in FPMR that satisfies (fO.AS, fR)∈ FMap, then:*

*CSTM ⇔ CSTR.*

**Theorem 1.** *Given a propositional feature models PFMO, an atomic-set-reduced propositional feature model PFMR of PFMO, and the FMap between them (see Definition 3.B),*

*∀ (AS, f) ∈ FMap ⋅*

*(AS.root is false-optional)↔ (f is false-optional).*

**Proof.** *AS.root* is false-optional

⇔ *CSTO*∧*AS.root.parent* is satisfiable ∧

(*CSTO*∧*AS.root.parent*∧*AS.root* is unsatisfiable ∨

*CSTO*∧*AS.root.parent*∧¬*AS.root* is unsatisfiable)

⇔ *CSTM*∧*f.parent* is satisfiable ∧

(*CSTM*∧*f.parent*∧*f* is unsatisfiable ∨

*CSTM*∧*f.parent*∧¬*f* is unsatisfiable)

⇔ *CSTR*∧*f.parent* is satisfiable ∧

(*CSTR*∧*f.parent*∧*f* is unsatisfiable ∨

*CSTR*∧*f.parent*∧¬*f* is unsatisfiable)

⇔ *f* is false-optional.

(Here, *CSTM* is *the constraints derived by replacing each feature fO in CSTO with the feature fR in FPMR that satisfies (fO.AS, fR)∈ FMap.*) 

**Theorem 2.** *Given a propositional feature models PFMO, and an atomic-set-reduced propositional feature model PFMR, the number of features in PFMR is equal to the numbe of optional features in PFMO plus 1, i.e.*

*#features(PFMR) = #optional-features(PFMO) + 1*

**Proof.** From Lemma 2.C, Lemma 2.E, and Definition 3. 

Since an atomic-set-reduced PFM holds the *false-optional-preserving* property but contains fewer features than the original PFM, in the detection of false-optional features, we only need to detect the reduced PFM. To attain this purpose, we need a practical method to construct an atomic-set-reduced PFM of a PFM. Such a method is provided in Algorithm 1.

**Algorithm 1.** (constructing atomic-set-reduced PFMs)

**Input:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *pfm* | : | A PFM, which has the following 4 attributes: | | |
| *root* | : | the root feature of this PFM. |
| *fset* | : | the set that contains all the features in this PFM. |
| *refine* | : | the refinement realtion between features. |
| *cset* | : | the set that contains all the clauses in *CST*. |

**Output:**

|  |
| --- |
| An atomic-set-reduced PFM of the input.(The output also has the 4 attributes as in the input) |

1. **PFM construct\_AS\_redecuded\_PFM(PFM *pfm*) {**
2. PFM *result*;
3. *result*.*fset* = *result*.*refine* = {};
4. Function<Feature→Feature> ffunc = {};
5. *result*.*fset*.*add*(*pfm*.*root*);
6. *result*.*root* = *pfm*.*root*;
7. *ffunc*.*add*((*root*, *root*));
8. *construct*(*root*, *root*, *pfm*, *result*);
9. Set *cset* = *pfm*.*cset*.*clone*();
10. for each *e* ∈ *cset* {
11. replace each feature *f* in *e* with ffunc(*f*)；
12. if (e is true) *cset*.*remove*(*e*);
13. }
14. *result*.*cset* = *cset*;
15. return result;
16. **}**
17. **void construct(Feature *as\_root*, Feature *f*,**

**PFM *original*, PFM *reduced*, Set** ffunc**) {**

1. Set *children* = *f*.(*original*.*refine*);
2. for each *e* ∈ *children*{
3. if(*e*.*optional*) {
4. *reduced*.*fset*.*add*(e);
5. *reduced*.*refine*.*add*((*as\_root*, *e*));
6. *ffunc*.*add*((*e*, *e*));
7. *construct*(*e*, *e*, *original*, *reduced, ffunc*);
8. }else{
9. *ffunc*.*add*(*e*, *as\_root*);
10. construct(*as\_root*, *e*, *original*, *reduced, ffunc*);
11. }
12. }
13. **}**

\*\*\*Please give some explanations and complex analysis of this algorithm. \*\*\*

**Theorem 3.** *Given a propositional feature models PFMO, and the atomic-set-reduced propositional feature model PFMR constructed by Algorithm 1, the number of clauses in CSTO and the number of clauses in CSTR holds the following relation:*

*#clauses(CSTO) − #clauses(CSTR)*

*≥ 2⋅(#features(PFMO)− #features(PFMR)).*

**Proof.** Given a tree with *n* node, there are *n−1* edges in this tree. Therefore, the number of refinement instances between features in *PFMR* will be *(#features(PFMO)− #features(PFMR))* smaller than that in *PFMO*. In Algorithm 1, the refinement relation of *PFMR* is constructed by only preserving those refinement instances between features in different atomic sets. That is, all the refinement instances between features in the same atomic set are removed. Based on the *Mandatory-Refines* rule in TABLE 2, a refinement instance between two features in an atomic set, will be transformed into two clauses in *CSTO*. Based on line 11 and 12 in Algorithm 1, any clause transformed either from a refinement realtion between two features in an atomic set or from a constraint in which all involved features belong to the same atomic set, will be removed in the construction of *CSTR*. And in Algorithm 1, the *CSTR* is only constructed by first transforming each clause in *CSTO* into a new clause and then either adding this new clause into *CSTR* or not, and no other clauses that are not transformed from *CSTO* is added into *CSTR*. Therefore, this theorem is proved. 

## 3.3 Technique 2: Reasoning-Based Detection

By using the *atomic sets* technique (i.e. the Algorithm 1), we can get a *atomic-set-reduced* PFM that exactly preserves all the *false-optional* features in the original PFM, but only has a smaller scale than the original one (i.e., a smaller number of features and a smaller number of constraints). And then, we can of cause traverse each optional feature in this atomic-set-reduced PFM, and check whether it is false-optional by invoking a *SAT-Solver*. However, we find that, the efficiency of such a traversal-based detection can be improved further, by integrating a reasoning-based approach into the detection (that is, some features’ *false-optional* values can be reasoned from that of other features, and therefore unnecessary to invoke a *SAT-Solver* anymore). We call such an efficiency-improving technique the *reasoning-based detection*.

In the following, we first distinguish four kinds of optional features in Definition 4, and then introduce two preliminary concepts in In the following, we use *the false-optional state* of an optional feature to denote whether this feature is *unselectable false-optional*, *unremovable false-optional*, *trival true-optional*, or *nontrivial true-optional*.

**Definition 5** and Definition 6, respectively. After that, we present the *reasoning-based detection* technique in Theorem 4, and … .

**Definition 4.** (unseletable/unremovable false-optional features; trivial/nontrivial true-optional features)

*Given a propositional feature model PFM, the constraint CST, and an optional feature f in PFM:*

1. *f is a unselectable false-optional feature, iff*

* *f is a false-optional feature*, *and*
* *CST*∧*f.parent*∧*f is unsatisfiable.*

1. *f is a unremovable false-optional feature, iff*

* *f is a false-optional feature, and*
* *CST*∧*f.parent*∧¬*f is unsatisfiable.*

1. *f is a trivial true-optional feature, iff*

* *f is a true-optional feature, and*
* *CST*∧*t.parent is unsatisfiable.*

1. *t is nontrivial true-optional feature, iff*

* *f is a true-optional feature, and*
* *CST*∧*t.parent is satisfiable.*

Definition 4 distinguishes between two kinds of false-optional features, and between two kinds of true-optional features.

**Lemma 3.** *The four concepts in Definition 4 form a partition to all the optional features in a PFM—that is, any optional feature in a PFM conforms to exactly one of the four concepts in Definition 4*.

**Proof.** According to Definition 1, the two concepts of *false-optional* features and *true-optional* features form a partition to all the optional features in a PFM. Then, we only need to prove that: 1. the two concepts of *unselectable* and *unremovable* false-optional features form a partition to all the *false-optional* features in a *PFM*; 2. the two concepts of *trivial* and *nontrivial* true-optional features form a partition to all the *true-optional* features in a *PFM*.

***Proof of Point 1***:

Given a *false-optional* feature *f*, suppose *f* is both *unselectable* and *unremovable false-optional*. According to the definitions of *unselectable* and *unremovable false-optional* features, *CST*∧*f.parent is satisfiable*, *CST*∧*f.parent*∧*f is unsatisfiable*, and *CST*∧*f.parent*∧¬*f is unsatisfiable*. Suppose *M* is a model *CST*∧*f.parent*, then we can know that, in *M*, *f.parent* is true, and *f* is either *true* or *false*. If *f* is *true*, then *CST*∧*f.parent*∧*f is satisfiable*, and thus f is not *unselectable* *false-optional*; otherwise, if *f* is *false*, then *CST*∧*f.parent*∧¬*f is satisfiable*, and thus *f* is not *unremovable* *false-optional*. Therefore, the supposition is *false*, and thus Point 1 is proved.

***Proof of Point 2***:

Given a *true-optional* feature *f*, suppose *f* is both *trivial* and *nontrivial* true-optional. According to the definition of *trivial true-optional* features, *CST*∧*f.parent is unsatisfiable*; and accoding to the definition of *nontrivial true-optional* features, *CST*∧*f.parent is satisfiable*. However, *CST*∧*f.parent* can not be both *satisfiable* and *unsatisfiable*. Therefore, the supposition is *false*, and thus Point 1 is proved. 

In the following, we use *the false-optional state* of an optional feature to denote whether this feature is *unselectable false-optional*, *unremovable false-optional*, *trival true-optional*, or *nontrivial true-optional*.

**Definition 5.** (refinement-imposed constraints)

*Given an atomic-set-reduced propositional feature model PFMR, and the constraint CSTR, for any clause C in CSTR, C is a refinment-imposed constraint iff it satisfied the following properties:*

1. *There are exactly two propositional variables in C (i.e. e is a constraint between two features).*
2. *Suppose the two features in C are a and b, then C conforms to one of the two forms: ¬a∨b, or ¬b∨a.*
3. *Suppose C conforms to the form ¬a∨b, then b is the parent of a.*

A refinement-imposed constraint is a constraint transformed from a refinement instance between two features. For convenience, we call a constraint that is not refinement-imposed an *explicitly-add* constraint. Based on Definition 5, the constraint *CST* of a PFM can be divided into two parts: *CST-RI* (the conjunction of all refinement-imposed constraints in *CST*), and *CST-EA* (the conjunction of all explicitly-added constraints in *CST*). Therefore, the following relation holds:

*CST* = *CST-RI*∧*CST-EA*.

**Definition 6.** (polluted/unpolluted features)

*Given an atomic-set-reduced propositional feature model PFMR, the constraint CSTR, and a feature f in PFMR, f is an unpolluted feature, iff every clause in CST containing f is a refinement-imposed constraint. Otherwise, f is a polluted feature.*

An unpolluted feature is a feature that is only involved in refinement-imposed constraints. An interesting point is that the root feature of an *atomic-set-reduced* PFM is always a polluted feature, since it is involved in the constraint generated by the *root-feature* rule in TABLE 2, a constraint that is not refinement-imposed.

**Theorem 4.** *Given an atomic-set-reduced propositional feature models PFMR, and an unpolluted feature f in PFMR, then the false-optional state of f can be decided using the rules in TABLE 4.*

**Proof.**

***Proof of rule 1*:** Since the nearest polluted acenstor *a* is a *trivial true-optional* or *unseletable false-optional* feature, we can deduce that *CSTR*∧*a* is unsatisfiable, and furthermore for any descendant *d* of *a*, *CSTR*∧*d* is unsatisfiable.

Since *f* is a descendant of *a*, then *f.parent* is either *a* or *a*’s descendant, we can deduce that *CSTR*∧*f.parent* is unsatisfiable. Therefore, *f* is a *trivial true-optional* feature.

***Proof of rule 2*:** Since *a* is either the *root* feature, or a *nontrivial true-optional* or *unremovable false-optional* feature, we can deduce that *CSTR*∧*a* is satisfiable.

Since the *Des* set is empty, then all descendants of *a* are unpolluted features, that is, all these descendants are only involved in refinement-imposed constraints. According to whether a clause involves *a*’s descendants, *CSTR* can be dividied into two parts:

*P1* =def *the conjunction of all the constraints in CSTR*

*involving a’s descendants*;

*P2* =def *the conjuction of all the other clauses in CSTR*.

It is obvious that *a* is the only feature that appears both in *P1* and *P2*. And therefore, if both *P1*∧*a* and *P2*∧*a* are satisfiable, then *CSTR*∧*a* is satisfiable.

Since *CSTR*∧*a* is satisfiable, we can know that both *P1*∧*a* and *P2*∧*a* are satisfiable. Since *f* is a descendant of *a*, then *P1*∧*a*∧*f.parent*∧*f* is satisfible (by selecting *a*, *f*, and any feature between *a* and *f*, and removing all other descendants of *a*); and, *P1*∧*a*∧*f.parent*∧¬*f* is satisfible (by selecting *a*, *f.parent*, and any feature between *a* and *f.parent*, and removing all other descendants of *a*).

Since *a* is the only common feature to *P1* and *P2*, and *P2*∧*a* is satisfiable, we can deduce that both *CSTR*∧*a*∧*f.parent*∧*f* and *CSTR*∧*a*∧*f.parent*∧¬*f* are satisfiable. Therefore, *f* is a *nontrivial true-optional* feature.

***Proof of rule 3*:** Suppose *d* is a *trivial true-optional* feature in *Des*, then we can know that *d.parent* is an unpolluted feature. Accroding to the definition of unpolluted features, *d.parent* is involved exactly in two constraints: ¬*d*∨*d.parent*, and ¬*d.parent*∨ *d.parent.parent*. Since *d* is *trival true-optional*, we know that *CSTR*∧*d.parent* is unsatisfiable.

TABLE 4  
Reasoning Rules about Unpolluted Features



*Given an atomic-set-reduced propositional feature models PFMR, and an unpolluted feature f in PFMR:*

* ***The nearest polluted ancestor of f*** *is a feature that satisfies the following three properties:*
  1. *It is a polluted feature;*
  2. *It is an ancestor of f;*
  3. *It is a descendant of all the other polluted ancestors of f (if has).*
* ***The nearest polluted descendants*** *of f is a set of features that satisfies the following four properties:*
  1. *Each feature in it is a polluted feature;*
  2. *Each feature in it is a descendant of f;*
  3. *Given a feature d in it, all the features between f and d are unpolluted features.*
  4. *If d is a polluted descendant feature of f, and all the features between f and d are unpolluted features, then d is a feature in it.*

Then, we can deduce that *CSTR*∧*d.parent.parent* is also unstisfiable. Otherwise, suppose itis stisfiable. According to whether a clause involves *d.parent*, *CSTR* can be divided into two parts:

*P1* =def (¬*d*∨*d.parent*)∧( ¬*d.parent*∨*d.parent.parent*);

*P2* =def *the conjuction of all the other clauses in CSTR*.

Then, we can know that both *P1*∧*d.parent.parent* and *P2*∧*d.parent.parent* are satisfiable. And the following two sets are two valid models of *P1*∧*d.parent.parent*:

*M1* = {*d.parent.parent*=*true*, *d.parent*=*true*, *d*=*true*},

*M2* = {*d.parent.parent*=*true*, *d.parent*=*true*, *d*=*false*}.

Suppose *M3* is a model of *P2*∧*d.parent.parent*, then in *M3*, *d* is either *true* or *false*. If *d* is *true*, and since *d.parent* is not involved in *P2*, then we can deduce that *M1*∪*M3* is a valid model of *CSTR*∧*d.parent*; if *d* is *false*, similarly, we can deduce that *M2*∪*M3* is a valid model of *CSTR*∧*d.parent*. Thus, we can deduce that *CSTR*∧*d.parent* is satisfiable, which conflicts with the assumption that *d* is *trivial true-optional*. Therefore, *CSTR*∧*d.parent.parent* is unstisfiable.

Similarly, we can deduce that for each feature *e* between *d.parent.parent* and *a* (including *a*), *CSTR*∧*e* is unstisfiable; in partular, *CSTR*∧*a* is unstisfiable, which conflicts with the precondition that *a* is either the *root* feature, or a *nontrivial true-optional* or *unremovable false-optional* feature. Therefore, when the above condition about *a* is true, it is impossible for *Des* to contain a *trivial true-optional* feature.

***Proof of rule 4*:** Suppose *d* is an *unremovable false-optional* feature in *Des*, we can know that *d.parent* is an unpolluted feature. Accroding to the definition of unpolluted features, *d.parent* is involved exactly in two constraints: ¬*d*∨*d.parent*, and ¬*d.parent*∨*d.parent.parent*. Then, *CSTR* can be divided into two parts:

*P1* =def (¬*d*∨*d.parent*)∧( ¬*d.parent*∨*d.parent.parent*);

*P2* =def *the conjuction of all the other clauses in CSTR*.

Since *d* is *unremovable false-optional*, then we can know that *P1*∧*P2*∧*d.parent*∧¬*d* is unsatisfiable. Since *P1* has a model *M* = {*d.parent.parent*=*true*, *d.parent*=*true*, *d*=*false*}, then, we can know that *P2*∧*d.parent.parent*∧*d.parent*∧¬*d* is unsatisfiable. Since *d.parent* does not appear in *P2*, then *P2*∧*d.parent.parent*∧¬*d* is unsatisfiable. And therefore, *CSTR*∧*d.parent.parent*∧¬*d* is unsatisfiable.

Then we can deduce *CSTR*∧*d.parent.parent*∧¬*d.parent* is unsatisfiable; otherwise, suppose it is satisfiable, and since ¬*d*∨*d.parent* is a clause in *CSTR*, then we can know *CSTR*∧*d.parent.parent*∧¬*d.parent*∧¬*d* is satisfiable, and thus *CSTR*∧*d.parent.parent*∧¬*d* is satisfiable, which conflicts with the fact that *CSTR*∧*d.parent.parent*∧¬*d* is unsatisfiable.Therefore, *CSTR*∧*d.parent.parent*∧¬*d.parent* is unsatisfiable.

Since *d* is *unremovable false-optional*, then, we can know that *CSTR*∧*d.parent* is satisfiable, and furthermore *CSTR*∧*d.parent.parent*∧*d.parent* is satisfiable. And since *CSTR*∧*d.parent.parent*∧¬*d.parent* is unsatisfiable, we can deduce that *d.parent* is an *unremovable flase-optional* feature.

Similarly, we can prove that each feature between *d* and *a* (not including *a*) is *unremovable flase-optional*, and therefore, *f* is *unremovable flase-optional*.

***Proof of rule 5*:** According to the definition of rule 5, any feature in *Des* is either *unselectable false-optional* or *nontrivial true-optional*, and at most one feature in *Des* is *nontrivial true-optional*.

It is obvious that any feature that is both a dscendant of *a* and an ancestor of any feature in Des is an unpolluted feature. For simplity, we call such a feature is a feature between *a* and *Des*. According to whether a clause involes features between *a* and *Des*, *CSTR* can be divided into two parts:

*P1* =def *the conjunction of all the constraints in CSTR*

*involving features between a and Des*;

*P2* =def *the conjuction of all the other clauses in CSTR*.

It is obvious to observe that the common features in *P1* and *P2* are exactly *a* and the features in *Des*.

Then we can deduce *CSTR* ∧*a* is satisfiable. The deduction process is as follows. If there is a *nontrivial true-optional* feature *d* in *Des*. Then we can know *CSTR* ∧*a*∧¬*d* is satisfiable. Since other features in *Des* all are *unselectable false-optional*, we can deduce that *CSTR*∧*a* is satisfiable; otherwise, there is at least one *unselectable false-optional* feature *e* in *Des* that is selectable (i.e. *CSTR*∧*a*∧¬*d*∧*e* is stisfiable), which is a conflicting result. Similarly, if there is no *nontrivial true-optional* feature *d* in *Des*, we can also decuce that *CSTR*∧*a* is satisfiable.

Therefore, we can know that both *P1*∧*a* and *P2*∧*a* are satisfiable. Then, for any feature *x* between *a* and *Des*, we can know that *P1*∧*a*∧*x.parent*∧*x* is satisfiable (by selecting *a*, *x*, and any feature between *a* and *x*, and removing all other features in *P1*); and *P1*∧*a*∧*x.parent*∧¬*x* is satisfiable (by selecting *a*, *x.parent*, and any feature between *a* and *x.parent*, and removing all other features in *P1*).

Since only *a* and features in *Des* are common features in *P1* and *P2*, and *P2*∧*a* are satisfiable, we can deduce that, for any feature *x* between *a* and *Des*, both *CSTR*∧ *x.parent*∧*x* and *CSTR*∧ *x.parent*∧¬*x* are satisfiable, and thus *x* is *nontrivial true-optional*. Obviously, *f* is a feature between *a* and *Des*, therefore *f* is also *nontrivial true-optional*.

***Proof of rule 6*:** If *CSTR*∧*a* is satisfiable, similarly to the proof of rule 5, we can prove that *f* is *nontrivial true-optional*.

If *CSTR*∧*a* is unsatisfiable, since *CSTR*∧*a* is satisfiable, we can know that for any valid model *M* of *CSTR*∧*a*, there is at least one feature *e* in *Des* that is selected (i.e. *e*=*true* ∈ *M*). Since *f* is between *a* and *e*, and when *e* is selected *f* must also be selected, we can deduce that any valid model *M* of *CSTR*∧*a*, *f* is must be selected. That is, *CSTR*∧*a*∧*f* is satisfiable and *CSTR*∧*a*∧¬*f* is unsatisfiable. Since *CSTR*∧*a*∧*f* is satisfiable, we can deduce that both *CSTR*∧*a*∧*f.parent* and *CSTR*∧*f.parent*∧*f* are satisfable. Since *CSTR*∧*a*∧¬*f* is unsatisfiable and *CSTR*∧*a*∧*f.parent* is satisfiable, we can deduce *CSTR*∧*f.parent*∧¬*f* is unsatisfiable. Therefore, *f* is *unremovable false-optional*. 

The value of Theorem 4 is obvious: to detect all the *false-optioanl* features in an *atomic-set-reduced* PFM, we only need to check the *false-optional states* of those polluted features by invoking a *SAT-Solver*; the *false-optional states* of those unpolluted features can be reasoned from the *false-optional states* of polluted features, and therefore unnecessary to invoke a *SAT-Solver* anymore.

**Definition 7.** (unpolluted-feature-removed PFMs)

*Given a propostional feature model PFMO, an atomic-set-reduced propositional feature model PFMR of PFMO, and a propostional feature model PFMRR, PFMRR is an unpolluted-feature-removed PFM of PFMR, iff it satisfies the following properties:*

1. *Excluding the root feature, all the other features in PFMRR are optional feature.*
2. *There is a one-to-one mapping* ***FMap*** *between polluted features in PFMR and features in PFMRR.*
3. *For any two elements (aR, pRR) and (dR, cRR) in FMap,*

*pRR is the parent of cRR ⇔*

*there is a refinment path from aR to dR, and in this path, all the features between aR to dR are unpolluted features*

1. *Suppose CST-EAR and CST-EARR are the conjunctions of all the explicitly-added constraints in PFMR and in PFMRR, respectively, and CST-EAM is the constraints derived by replacing each feature fR in CST-EAR with the feature fRR in PFMRR that satisfies (fR, fRR)∈ FMap, then:*

*CST-EAM⇔ CST-EARR*

**Theorem 5.** *Given*

* *an atomic-set-reduced propositional feature model PFMR,*
* *an unpolluted-feature-removed propositional feature model PFMRR of PFMR,*
* *the FMap between PFMR and PFMRR (see* Definition 7*.B), and*
* *the constraints CSTR of PFMR, and the constraints CSTRR of PFMRR,*

*then,*

1. *For any model MR of CSTR, if we remove all the value-assignments of unpolluted features from MR and then replace each polluted feature fR in MR with its coressponding feature fRR in PFMRR (i.e. (fR, fRR)∈FMap), we get a set of value-assignments of all features in PFMRR (denoted as MRR), then MRR is a model of PFMRR.*
2. *For any model MRR of CSTRR, if we replace each features fRR in MRR with its corresponding feature fR in PFMR (i.e. (fR, fRR)∈FMap), we get a set of value-assignment of all polluted features in PFMRR (denoted as M’R), then there is a model MR of CSTR that satisfies*

*M’R ⊆ MR.*

**Proof.**

***Proof of A*:** [To be added].

***Proof of B*:** [To be added]

**Corollary 1.** Given

* *an atomic-set-reduced propositional feature model PFMR,*
* *an unpolluted-feature-removed propositional feature model PFMRR of PFMR,*
* *the FMap between PFMR and PFMRR (see* Definition 7*.B), and*
* *the constraints CSTR of PFMR, and the constraints CSTRR of PFMRR,*

*then,*

1. *∀ (fR, fRR)∈FMap⋅ fR and fRR have a same false-optional state.*
2. *For any feature fRR in PFMRR and the set Chd contaning all children of fRR, suppose fR is fRR’s corresponding feature in PFMR (i.e. (fR, fRR)∈FMap) and Des is Chd’s corresponding feature set in PFMR (i.e. Des is constructed by replace each feature in Chd with its coressponding feature in PFMR), then*

*(*

*⇔ .*

**Proof.**

***Proof of A*:** [To be added].

***Proof of B*:** [To be added]

Corollary 1.A shows that in order to know the false-optional state of a polluted feature in an atomic-set-reduced PFM, we only need to check the false-optional state of the corresponding feature in an unpolluted-feature-removed PFM (which contains fewer features and has a smaller scale of SAT problems than the atomic-set-reduced PFM). Corollary 1.B shows that the applying of rule 6 in TABLE 4 can be transformed into SAT problems related to an unpolluted-feature-removed PFM.

To take advantages of Corollary 1, we need a practical method to construct an unpolluted-feature-removed PFM of an atomic-set-reduced PFM. Such a method is provided in Algorithm 2.

**Algorithm 2.** (constructing unpolluted-feature-removed PFMs)

**Input:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *pfm* | : | A atomic-set-reduced PFM, which has the following 4 attributes: | | |
| *root* | : | the root feature of this PFM. |
| *fset* | : | the set that contains all the features in this PFM. |
| *refine* | : | the refinement realtion between features. |
| *cset* | : | the set that contains all the clauses in *CST*. |

**Output:**

|  |
| --- |
| An unpolluted-feature-removed PFM of the input.(The output also has the 4 attributes as in the input) |

# 4 Reasoning-Based Detection

IEEE Computer Society style is to not citations in individual brackets, followed by a comma, e.g. “[1], [5]” (as opposed to the more common “[1, 5]” form.) Citation ranges should be formatted as follows: [1], [2], [3], [4] (as opposed to [1]-[4], which is not IEEE Computer Society style). When citing a section in a book, please give the relevant page numbers [2]. In sentences, refer simply to the reference number, as in [3]. Do not use “Ref. [3]” or “reference [3]” At the beginning of a sentence use the author names instead of “Reference [3],” e.g., “Smith and Smith [3] show ... .” Please note that references will be formatted by IEEE Computer Society production staff in the same order provided by the author.

# 5 Equations

If you are using Word, use either the Microsoft Equation Editor or the MathType add-on (http://www.mathtype.com) for equations in your paper (Insert | Object | Create New | Microsoft Equation or MathType Equation). “Float over text” should not be selected.

Number equations consecutively with equation numbers in parentheses flush with the right margin, as in (1). First, use the equation editor to create the equation. Then, select the “Equation” markup style. Press the tab key and write the equation number in parentheses. To make your equations more compact, you may use the solidus ( / ), the exp function, or appropriate exponents. Use parentheses to avoid ambiguities in denominators. Punctuate equations when they are part of a sentence, as in

 (1)

Be sure that the symbols in your equation have been defined before the equation appears or immediately following. Italicize symbols (*T* might refer to temperature, but T is the unit tesla). Per IEEE Computer Society, please refer to “(1),” not “Eq. (1)” or “equation (1),” except at the beginning of a sentence: “Equation (1) shows ... .” Also see *The Handbook of Writing for the Mathematical Sciences*, 1993. Published by the Society for Industrial and Applied Mathematics, this handbook provides some helpful information about math typography and other stylistic matters. For further information about typesetting mathematical equations, please visit the IEEE Computer Society styel guide: http://computer.org/author/style/math-exp.

Please note that math equations might need to be reformatted from the original submission for page layout reasons. This includes the possibility that some in-line equations will be made display equations to create better flow in a paragraph. If display equations do not fit in the two-column format, they will also be reformatted. Authors are strongly encouraged to ensure that equations fit in the given column width.

# 6 Discussions

## 6.1 Feature Models and Configuration

Because IEEE Computer Society staff will do the final formatting of your paper, some figures may have to be moved from where they appeared in the original submission. Figures and tables should be sized as they are to appear in print. Figures or tables not correctly sized will be returned to the author for reformatting.

## 6.2 Feature Selecting VS. Feature Binding

Detailed information about the creation and submission of images for articles can be found at: <http://computer.org/author/transguide/>. We strongly encourage authors to carefully review the material posted here to avoid problems with incorrect files or poorly formatted graphics.

Place figure captions below the figures; place table titles above the tables. If your figure has two parts, include the labels “(a)” and “(b)” as part of the artwork. Please verify that the figures and tables you mention in the text actually exist. Figures and tables should be called out in the order they are to appear in the paper. For example, avoid referring to figure “8” in the first paragraph of the article unless figure 8 will again be referred to after the reference to figure 7. **Please do not include figure captions as part of the figure. Do not put captions in “text boxes” linked to the figures. Do not put borders around the outside of your figures.** Per IEEE Computer Society, please use the abbreviation “Fig.” even at the beginning of a sentence. Do not abbreviate “Table.” Tables are numbered numerically.

Figures may only appear in color for certain journals. Please verify with IEEE Computer Society that the journal you are submitting to does indeed accept color before submitting final materials. **Do not use color unless it is necessary for the proper interpretation of your figures.**

Figures (graphs, charts, drawing or tables) should be named fig1.eps, fig2.ps, etc. If your figure has multiple parts, please submit as a single figure. Please do not give them descriptive names. Author photograph files should be named after the author’s LAST name. Please avoid naming files with the author’s first name or an abbreviated version of either name to avoid confusion. If a graphic is to appear in print as black and white, it should be saved and submitted as a black and white file (grayscale or bitmap.) If a graphic is to appear in color, it should be submitted as an RGB color file.



Fig. 1. Magnetization as a function of applied field. Note that “Fig.” is abbreviated. There is a period after the figure number, followed by one space. It is good practice to briefly explain the significance of the figure in the caption.

Figure axis labels are often a source of confusion. Use words rather than symbols. As an example, write the quantity “Magnetization,” or “Magnetization *M*,” not just “*M*.” Put units in parentheses. Do not label axes only with units. As in Fig. 1, for example, write “Magnetization (A/m)” or “Magnetization (Am−1),” not just “A/m.” Do not label axes with a ratio of quantities and units. For example, write “Temperature (K),” not “Temperature/K.” Table 1 shows some examples of units of measure.

Multipliers can be especially confusing. Write “Magnetization (kA/m)” or “Magnetization (103 A/m).” Do not write “Magnetization (A/m) × 1,000” because the reader would not know whether the top axis label in Fig. 1 meant 16,000 A/m or 0.016 A/m. Figure labels should be legible, approximately 8 to 12 point type. When creating your graphics, especially in complex graphs and charts, please ensure that line weights are thick enough that when reproduced at print size, they will still be legible. We suggest at least 1 point.

## 6.3 Footnotes

Number footnotes separately in superscripts (Insert | Footnote)[[1]](#footnote-1). Place the actual footnote at the bottom of the column in which it is cited; do not put footnotes in the reference list (endnotes). Use letters for table footnotes (see Table 1). Please do not include footnotes in the abstract and avoid using a footnote in the first column of the article. This will cause it to appear of the affiliation box, making the layout look confusing.

TABLE 1  
Units for Magentic Properties



Statements that serve as captions for the entire table do not need footnote letters.

aGaussian units are the same as cgs emu for magnetostatics; Mx = maxwell, G = gauss, Oe = oersted; Wb = weber, V = volt, s = second, T = tesla, m = meter, A = ampere, J = joule, kg = kilogram, H = henry.

## 6.4 Lists

The IEEE Computer Society style is to create displayed lists if the number of items in the list is longer than three. For example, within the text lists would appear 1) using a number, 2) followed by a close parenthesis. However, longer lists will be formatted so that:

1. Items will be set outside of the paragraphs.
2. Items will be punctuated as sentences where it is appropriate.
3. Items will be numbered, followed by a period.

## 6.5 Theorems and Proofs

Theorems and related structures, such as axioms corollaries, and lemmas, are formatted using a hanging indent paragraph. They begin with a title and are followed by the text, in italics.

**Theorem 1.** *Theorems, corollaries, lemmas, and related structures follow this format. They do not need to be numbered, but are generally numbered sequentially.*

Proofs are formatted using the same hanging indent format. However, they are not italicized.

**Proof.** The same format should be used for structures such as remarks, examples, and solutions (though these would not have a Q.E.D. box at the end as a proof does). 

# 7 End Sections

## Appendices

Appendixes, if needed, appear before the acknowledgment. In the event multiple appendices are required, they will be labeled “Appendix A,” “Appendix B, “ etc. If an article does not meet submission length requirements, authors are strongly encouraged to make their appendices supplemental material.

IEEE Computer Society Transactions accepts supplemental materials for review with regular paper submissions. These materials may be published on our Digital Library with the electronic version of the paper and are available for free to Digital Library visitors. Please see our guidelines below for file specifications and information. Any submitted materials that do not follow these specifications will not be accepted. All materials must follow US copyright guidelines and may not include material previously copyrighted by another author, organization or company. More information can be found at http://computer.org/author/transguide/  
SuppMat.htm.

## 7.2 Acknowledgments

The preferred spelling of the word “acknowledgment” in American English is without an “e” after the “g.” Use the singular heading even if you have many acknowledgments. Avoid expressions such as “One of us (S.B.A.) would like to thank ... .” Instead, write “F. A. Author thanks ... .” Sponsor and financial support acknowledgments are included in the acknowledgment section. For example: This work was supported in part by the US Department of Commerce under Grant BS123456 (sponsor and financial support acknowledgment goes here). Researchers that contributed information or assistance to the article should also be acknowledged in this section.

## 7.3 References

Unfortunately, the Computer Society document translator cannot handle automatic endnotes in Word; therefore, type the reference list at the end of the paper using the “References” style. See the IEEE Computer Society’s style for reference formatting at: http://computer.org/author/style/  
transref.htm. The order in which the references are submitted in the manuscript is the order they will appear in the final paper, i.e., references submitted nonalphabetized will remain that way.

Please note that the references at the end of this document are in the preferred referencing style. Within the text, use “et al.” when referencing a source with more than three authors. In the reference section, give all authors’ names; do not use “et al.” Do not place a space between an authors' initials. Papers that have not been published should be cited as “unpublished” [4]. Papers that have been submitted or accepted for publication should be cited as “submitted for publication” [5]. Please give affiliations and addresses for personal communications [6].

Capitalize all the words in a paper title. For papers published in translation journals, please give the English citation first, followed by the original foreign-language citation [7].

## 7.3 Additional Formatting and Style Resources

Additional information on formatting and style issues can be obtained in the IEEE Computer Society Style Guide, which is posted online at: http://computer.org/author/style/. Click on the appropriate topic under the Special Sections link.

# Conclusion

Although a conclusion may review the main points of the paper, do not replicate the abstract as the conclusion. A conclusion might elaborate on the importance of the work or suggest applications and extensions. Authors are strongly encouraged not to call out multiple figures or tables in the conclusion—these should be referenced in the body of the paper.

**Acknowledgment**

The authors wish to thank A, B, C. This work was supported in part by a grant from XYZ.

**References**

1. 1
2. 2
3. 3
4. 4
5. J.S. Bridle, “Probabilistic Interpretation of Feedforward Classification Network Outputs, with Relationships to Statistical Pattern Recognition,” *Neurocomputing—Algorithms, Architectures and Applications,* F. Fogelman-Soulie and J. Herault, eds., NATO ASI Series F68, Berlin: Springer-Verlag, pp. 227-236, 1989. (Book style with paper title and editor)
6. W.-K. Chen, *Linear Networks and Systems.* Belmont, Calif.: Wadsworth, pp. 123-135, 1993. (Book style)
7. H. Poor, “A Hypertext History of Multiuser Dimensions,” *MUD History,* http://www.ccs.neu.edu/home/pb/mud-history.html. 1986. (URL link \*include year)
8. K. Elissa, “An Overview of Decision Theory,"unpublished. (Unplublished manuscript)
9. R. Nicole, "The Last Word on Decision Theory," *J. Computer Vision,* submitted for publication. (Pending publication)
10. C. J. Kaufman, Rocky Mountain Research Laboratories, Boulder, Colo., personal communication, 1992. (Personal communication)
11. D.S. Coming and O.G. Staadt, "Velocity-Aligned Discrete Oriented Polytopes for Dynamic Collision Detection," *IEEE Trans. Visualization and Computer Graphics*, vol. 14,  no. 1,  pp. 1-12,  Jan/Feb  2008, doi:10.1109/TVCG.2007.70405. (IEEE Transactions )
12. S.P. Bingulac, “On the Compatibility of Adaptive Controllers,” *Proc. Fourth Ann. Allerton Conf. Circuits and Systems Theory*, pp. 8-16, 1994. (Conference proceedings)
13. H. Goto, Y. Hasegawa, and M. Tanaka, “Efficient Scheduling Focusing on the Duality of MPL Representation,” *Proc. IEEE Symp. Computational Intelligence in Scheduling* *(SCIS ’07)*, pp. 57-64, Apr. 2007, doi:10.1109/SCIS.2007.367670. (Conference proceedings)
14. J. Williams, “Narrow-Band Analyzer,” PhD dissertation, Dept. of Electrical Eng., Harvard Univ., Cambridge, Mass., 1993. (Thesis or dissertation)
15. E.E. Reber, R.L. Michell, and C.J. Carter, “Oxygen Absorption in the Earth’s Atmosphere,” Technical Report TR-0200 (420-46)-3, Aerospace Corp., Los Angeles, Calif., Nov. 1988. (Technical report with report number)
16. L. Hubert and P. Arabie, “Comparing Partitions,” *J. Classification,* vol. 2, no. 4, pp. 193-218, Apr. 1985. (Journal or magazine citation)
17. R.J. Vidmar, “On the Use of Atmospheric Plasmas as Electromagnetic Reflectors,” *IEEE Trans. Plasma Science*, vol. 21, no. 3, pp. 876-880, available at http://www.halcyon.com/pub/journals/21ps03-vidmar, Aug. 1992. (URL for Transaction, journal, or magzine)
18. J.M.P. Martinez, R.B. Llavori, M.J.A. Cabo, and T.B. Pedersen, "Integrating Data Warehouses with Web Data: A Survey," *IEEE Trans. Knowledge and Data Eng.*, preprint, 21 Dec. 2007, doi:10.1109/TKDE.2007.190746.(PrePrint)

**First A. Author** Biographies should be limited to one paragraph consisting of the following: sequentially ordered list of degrees, including years achieved; sequentially ordered places of employ concluding with current employment; association with any official journals or conferences; major professional and/or academic achievements, i.e., best paper awards, research grants, etc.; any publication information (number of papers and titles of books published); current research interests; association with any professional associations.

**Second B. Author Jr.** biography appears here. Degrees achieved followed by current employment are listed, plus any major academic achievements.

**Third C. Author** is a member of the IEEE and the IEEE Computer Society.

1. It is recommended that footnotes be avoided (except for the unnumbered footnote with the receipt date on the first page). Instead, try to integrate the footnote information into the text. [↑](#footnote-ref-1)