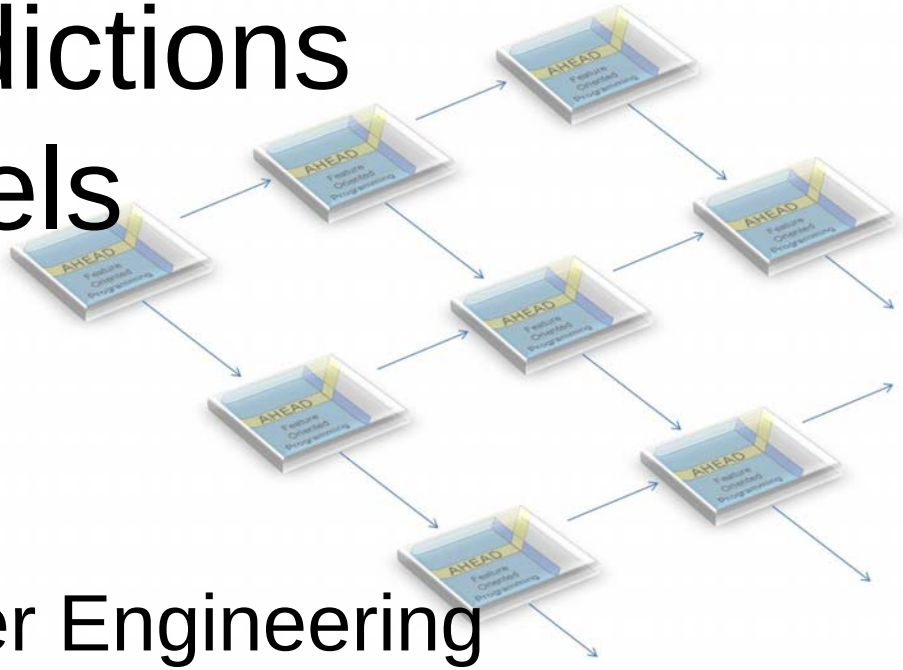
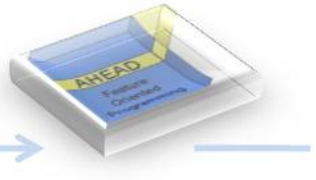
Finding Contradictions in Feature Models



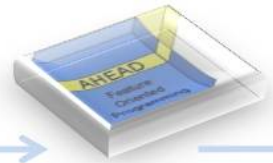
Adithya Hemakumar
Electrical and Computer Engineering
University of Texas at Austin

Introduction and Background

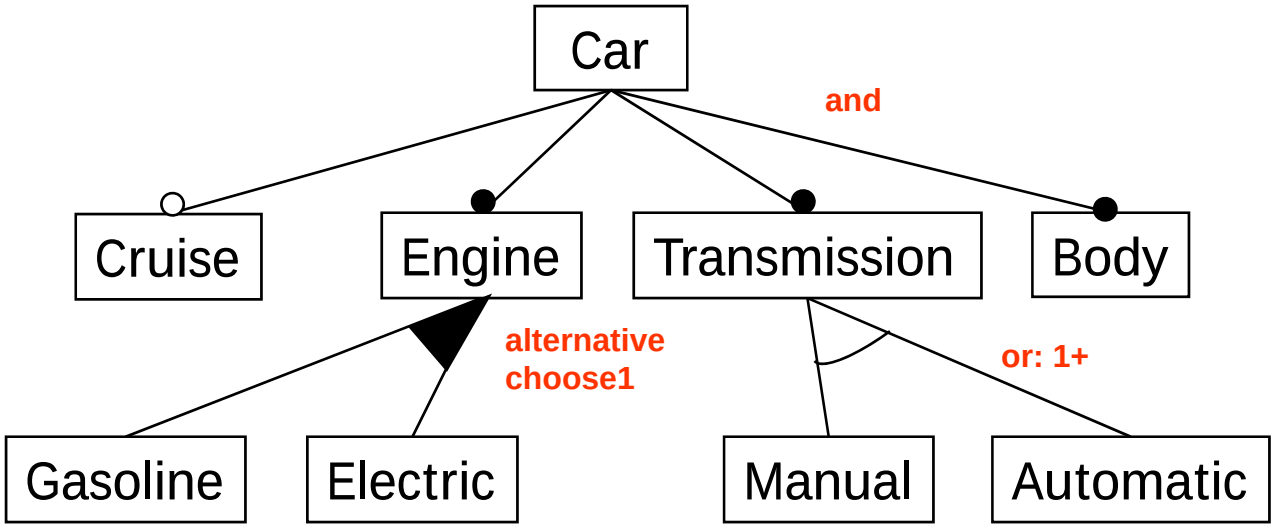


- **Feature Model (FM)** is a common way to express the products of a **Software Product Line (SPL)**
  - no 2 programs in an SPL have the same set of features
- FM can be formally defined as a:
  - context free grammar (CFG) with constraints
  - propositional formula

Example



feature diagram



CFG with constraints

Car : [Cruise] (Engine)+ Transmission Body ;

Engine : Gasoline | Electric ;

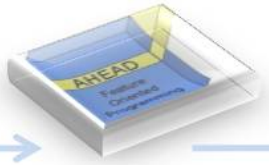
Transmission : Manual | Automatic ;

Cruise ➔ Automatic

grammar

constraint

FMs May Have Contradictions



- All products of M have features C and D, and optionally A and B

  M: [A] [B] C D ;

- Now add (nonsensical) constraints

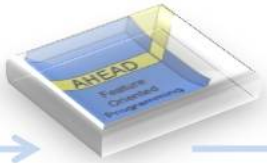
  A implies B;
  B implies not A;

- Contradiction arises when A is selected;
  B is selected by enforcing 1st constraint;
  A is deselected by enforcing the 2nd constraint

- Selecting a feature implies its deselection is an error and a contradiction in the model

- Example is trivial, but as models grow larger in size, very hard to detect contradictions by inspection – these errors do arise!

Contradictions and Dead Features



- A **dead feature** [Benevides et al] is a feature that is present in no product

  M: [A] [B] C D ;

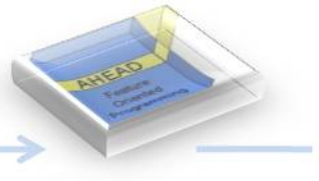
  A implies B;
  B implies not A;

- A is a dead feature
  - it appears in no product of M
  - no conditions for ‘deadness’
  - contradictions reveal dead features

- Finding such features is not too difficult

- Scary part – there are conditionally dead features
  - features that are dead only when specific conditions are satisfied
  - **zombies** “not quite dead”…

Zombies



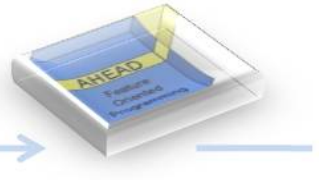
- Consider model G – each product has D, with optional features A, B, C

```
G: [A]  [B]  C   D ;          // grammar

(          A) implies B;      // constraints
B implies not A
```

- Note G is a generalization of M
  - all products of M are products of G
  - A is NOT a dead feature in G
  - G has a product with A – ex. {A, D}
- If feature C is selected, G reduces to model M
  - contradiction is exposed when C is selected, and then A
  - A is ‘dead’ whenever feature C is selected – A is a zombie
- We want to find such contradictions (zombies)

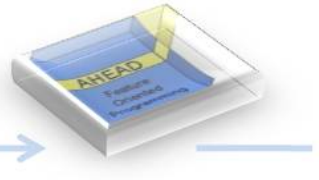
Interesting Connection



- A model is **void** if it has no products [Benavides]
  - conceptually not difficult to verify
  - model is **satisfiable** if it has 1 or more products

- When a user selects a feature in a model, a **submodel** is created
  - grammar and constraints are simplified based on this selection
    ex: G → M when feature C is selected
  - model is **contradiction free** if every possible submodel (created by selecting 1+ features) is satisfiable or non-void
  - equivalently, there are no dead features in a model and every one of its submodels

This Paper is About



- Finding contradictions in feature models

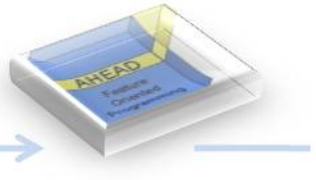
  = finding conditionally dead features in models

  = finding unsatisfiable submodels in models

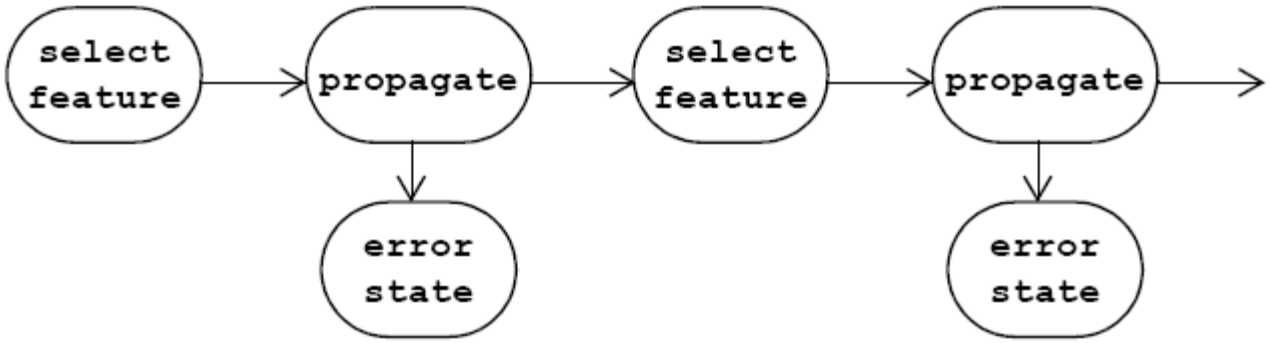
- In following slides, present brief overview of results

- Approach: use **Boolean Constraint Propagation (BCP)**
  - classical AI algorithm for logic truth maintenance
  - contradictions show up as pair of inference chains that lead to contradictory conclusions

Solution #1: Model Checking

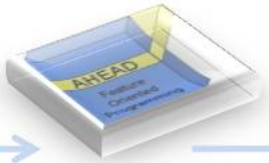


- Can express problem of contradiction freedom as a state reachability problem
  - encode BCP algorithm execution as a state machine



  - upon each feature selection, propagate constraints (thereby producing a submodel), and check for errors
  - if an error state can be reached, the model has a contradiction

SPIN Results



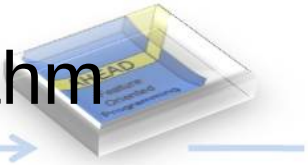
- Models

Model Name	# of Features	# of CNF Clauses	Brief Description of the Model's SPL
BerkeleyDB	55	185	BerkeleyDB [18]
Folutest	13	66	a notepad application
Freeman	3	17	a scalar vector graphics application
GG4-model	15	140	an elaborated graph product line
GPL	17	188	graph product line [20]
Notepad	20	155	a notepad application
SVGMap	19	52	a SVG map application
TightVNC	21	83	desktop sharing application
Violet	64	341	image processing application
apl	12	47	error-injected model
long	12	17	error-injected model

- Results

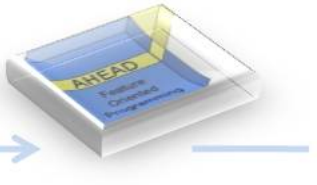
Model Name	Exhaustive (secs)
BerkeleyDB	∞
Folutest	3.5
Freeman	1.2
GG4-model	8.6
GPL	∞
Notepad	∞
SVGMapApp	∞
TightVNC	∞
Violet	∞
apl	4.8
long	1.6

Solution #2: Incremental Consistency Algorithm



- We recognized that in a selection of k+1 features, the order in which the first k features were selected does not matter
- Reduces search space form O(n!) to $O(n2^{n-1})$
- Encoding this space reduction in Promela problematic
  - programs became complicated
  - did not reduce likelihood of exhausting memory
  - would be easier to write our own tool …
- **Incremental Consistency Algorithm (ICA)**
  - verify that model is contradiction free for 1, 2, 3, … n selections
  - leverage computations of previous results
  - see paper for details

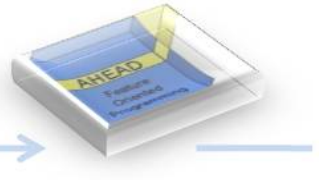
Results



- Could find more answers
- With backtracking optimizations (described in paper) over an order of magnitude faster than SPIN
- Still could not verify contradiction freedom for some models

Model Name	Spin Exhaustive (secs)	ICA Optimized (secs)
BerkeleyDB	∞	∞
Folutest	3.5	0.6
Freeman	1.2	0.01
GG4-model	8.6	1.8
GPL	∞	10.3
Notepad	∞	118.5
SVGMapApp	∞	10.8
TightVNC	∞	28.9
Violet	∞	∞
apl	4.8	0.02
long	1.6	0.02

Results

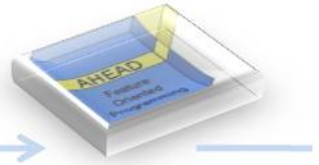


- When we couldn't prove contradiction freedom, we reported "coverage"
  - guarantees of conflict freedom for selecting k features

k-consistent	BerkeleyDB (in hours)	Violet (in hours)
1	0.00	0.00
2	0.01	0.01
3	0.04	0.20
4	0.20	1.80
5	0.86	11.16
6	3.22	–
7	11.52	–

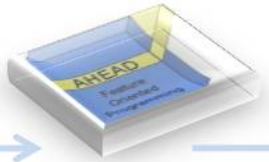
  - for large models, we hardly covered any of the search space
  - ICA is effective for models of 20-25 features
  - ineffective for larger models

In Interim… Use Run-Time Solution

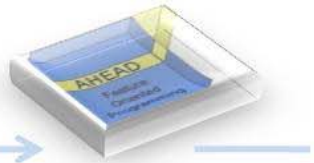


- Create feature modeling tools with both
  - BDDs and BCP algorithm
  - BCP algorithm runs in the background
  - under normal circumstances, constraint propagation output of BDD and BCP should be identical
  - when outputs are not identical, that’s when a contradiction (or conditionally dead) feature has been found
  - silently report the error back to designers for model repair

Conclusions and Future Work



- Finding errors (contradictions) in feature models is important
  - we discover them now by accident
  - would like to have a static, efficient analysis to find errors
  - static solution works for small feature models (size 20-25 features); larger models preclude analysis
- Seems to be very hard
  - we suspect that there is a simple, fast analysis
  - but we tried variations, with little luck
  - we encourage others to have a look



The End