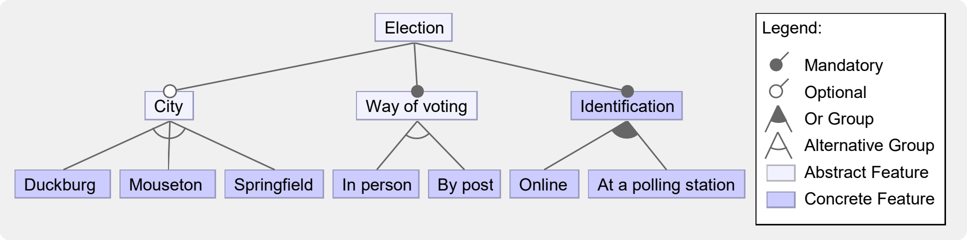
Incremental Construction of Modal Implication Graphs for Evolving Feature Models

FOSD 2022, Wien | S. Krieter, **Rahel Arens**, M. Nieke, C. Sundermann, T. Heß, T. Thüm, C. Seidl | March 30, 2022

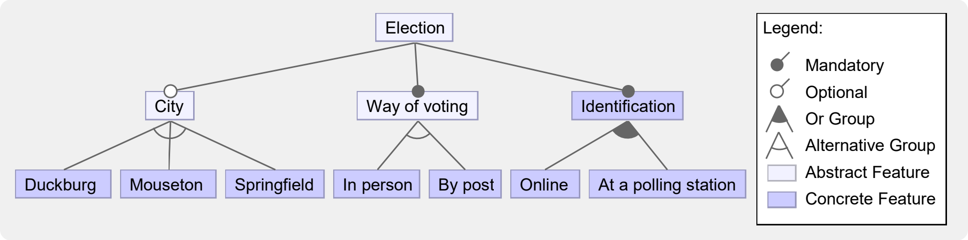
Software Engineering
Programming Languages

universität ulm

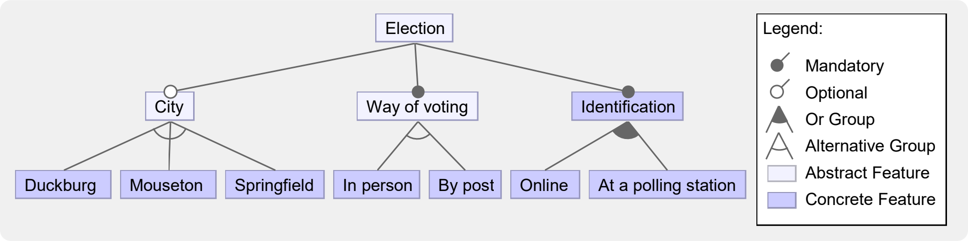
Feature Models and Configurations

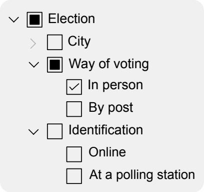


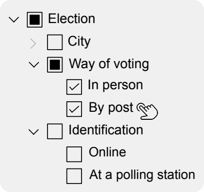
Feature Models and Configurations



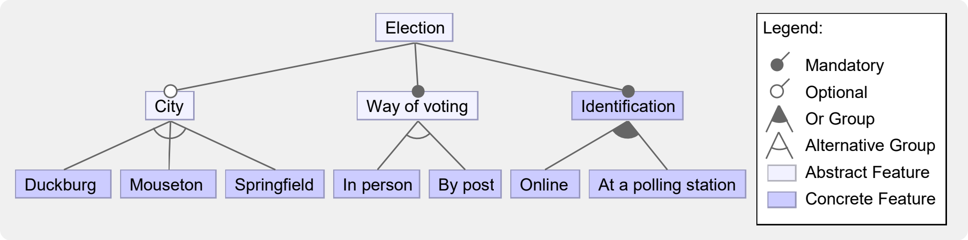
Feature Models and Configurations

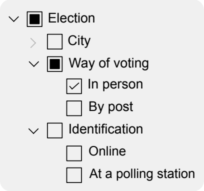


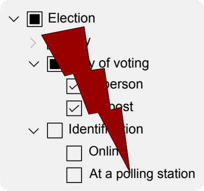




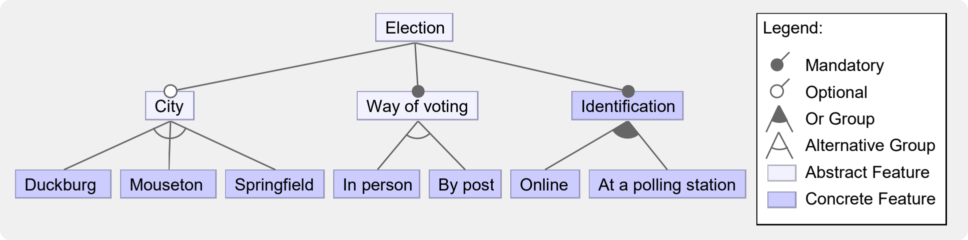
Feature Models and Configurations

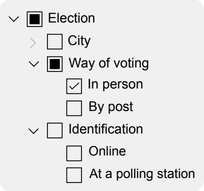


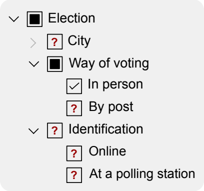




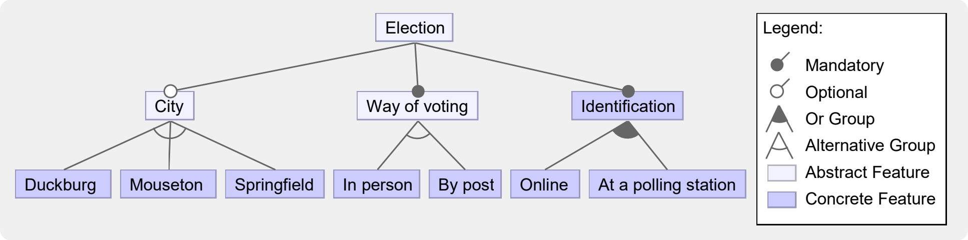
Feature Models and Configurations

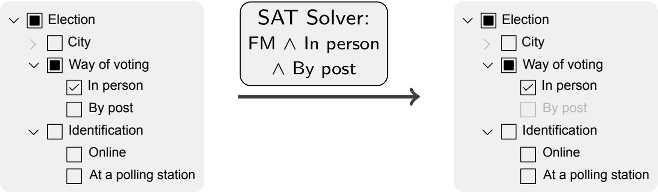




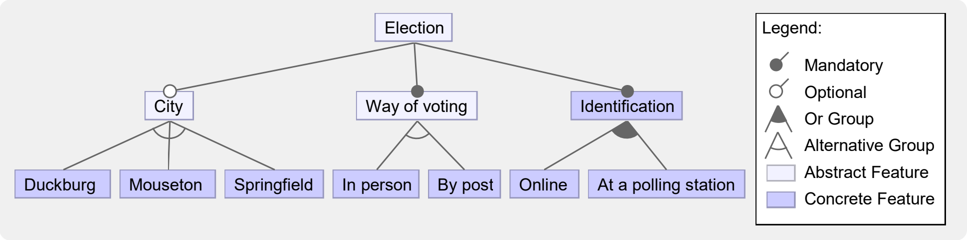


Feature Models and Configurations

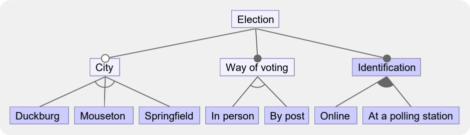




Feature Models and Configurations



Modal Implication Graph Background

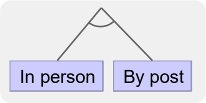


Extended Implication Graph

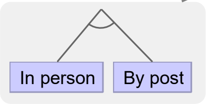
Represents the dependencies of the features

Each vertex represents a single literal

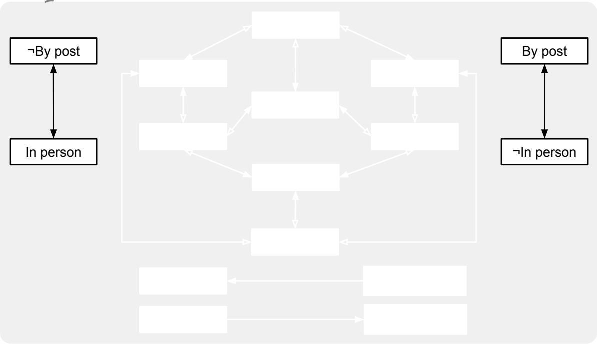
Modal Implication Graph Background



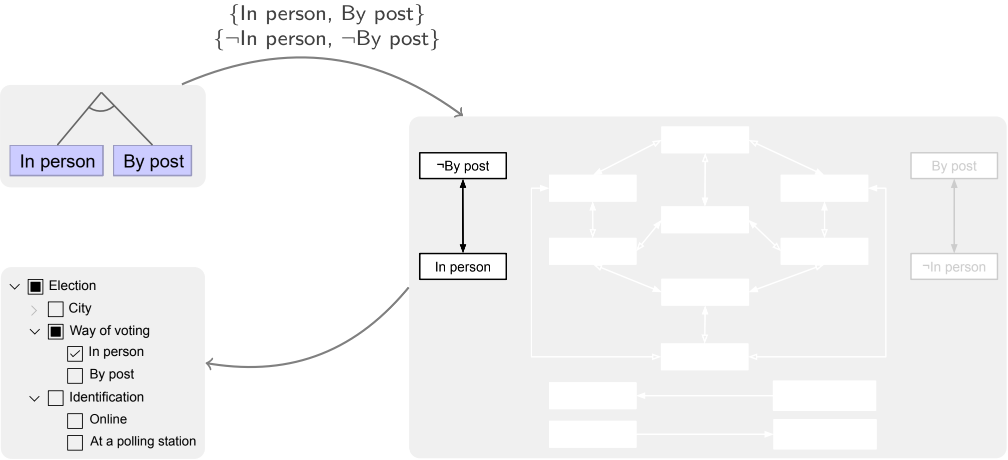
Modal Implication Graph Background



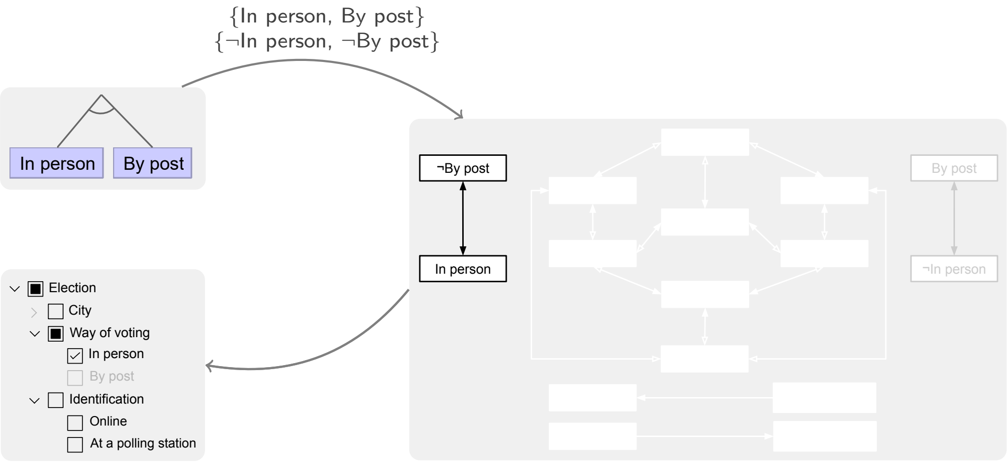
{In person, By post}
{¬In person, ¬By post}



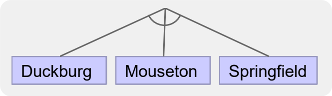
Modal Implication Graph Background



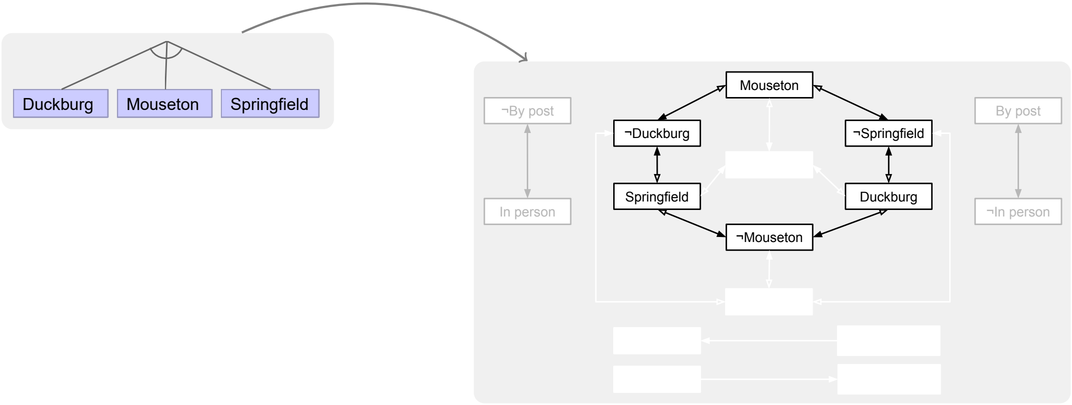
Modal Implication Graph Background



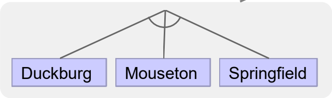
Modal Implication Graph Background



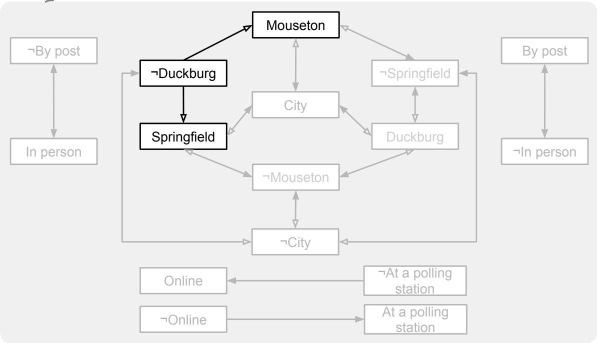
Modal Implication Graph Background



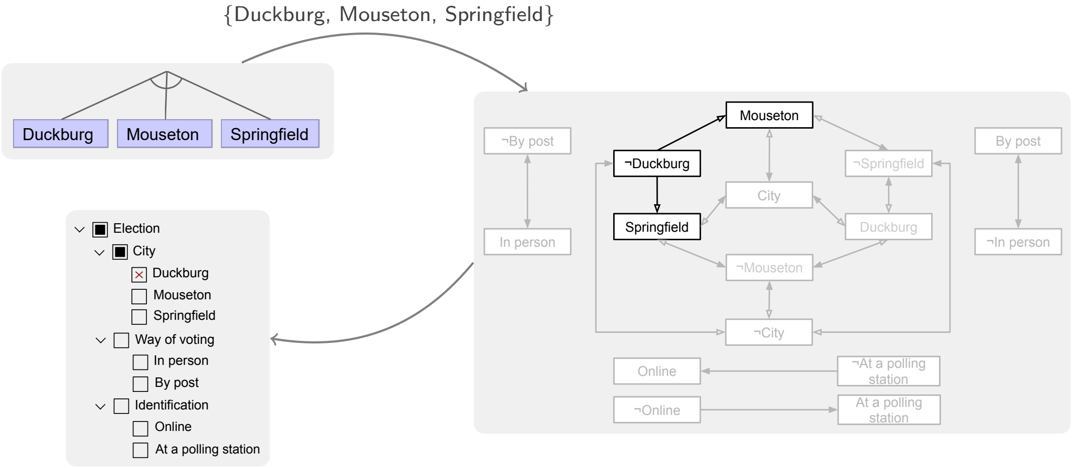
Modal Implication Graph Background



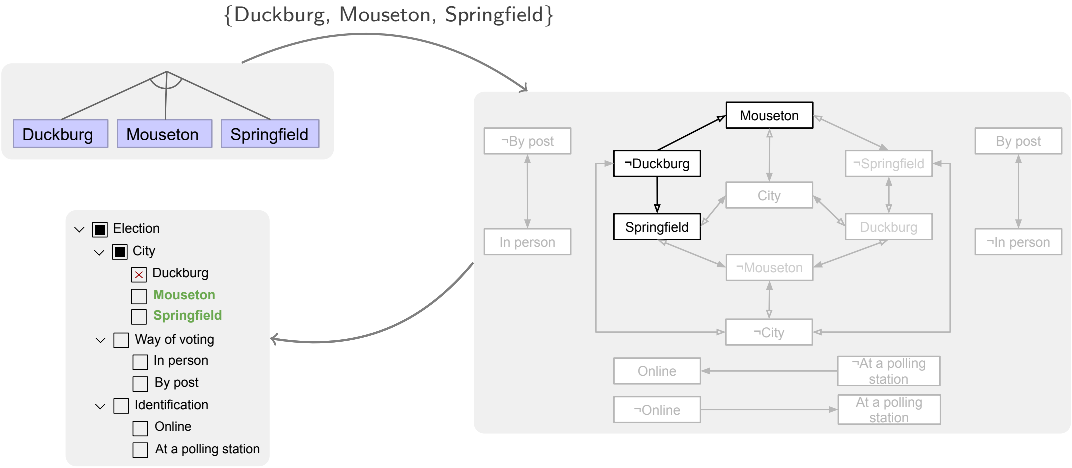
{Duckburg, Mouseton, Springfield}



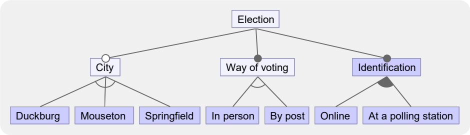
Modal Implication Graph Background



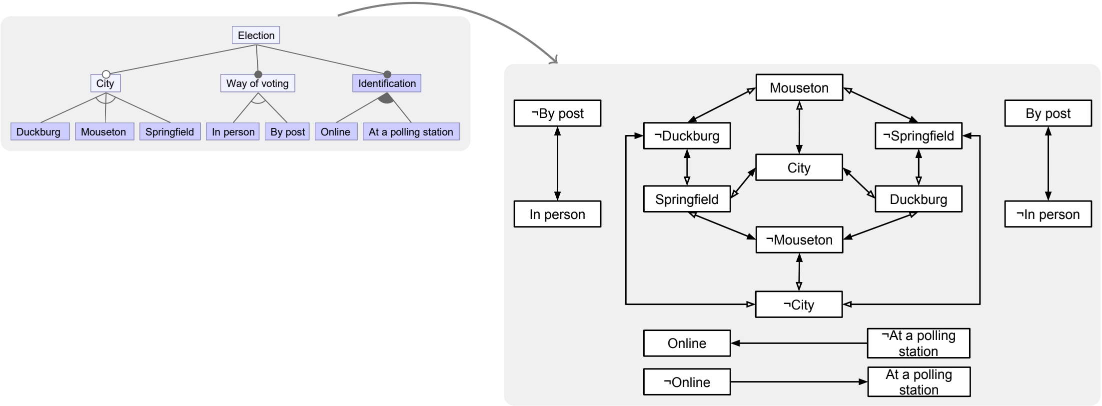
Modal Implication Graph Background



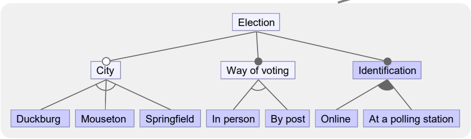
Modal Implication Graph Background

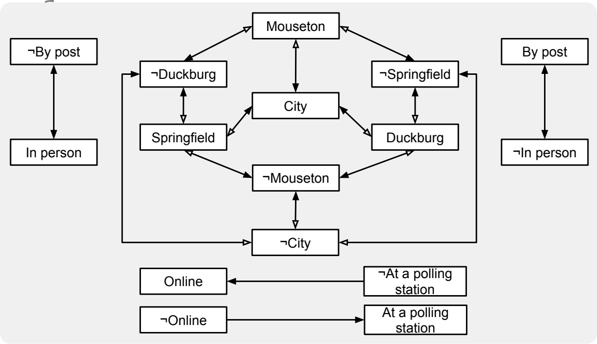


Modal Implication Graph Background



Modal Implication Graph Background





Unit clause:
Core or dead feature

Clauses with two literals:
Strong edges

Clauses with three or more literals:
Weak edges

2018 ACM/IEEE 40th International Conference on Software Engineering

Propagating Configuration Decisions with Modal Implication Graphs

Sebastian Krieter
University of Magdeburg, Germany
Harz University of Applied Sciences
Wernigerode, Germany
sebastian.krieter@ovgu.de

Thomas Thüm
TU Braunschweig, Germany
t.thuem@tu-braunschweig.de

Sandro Schulze
Reimar Schröter
Gunter Saake
University of Magdeburg, Germany
sandro.schulze@ovgu.de
reimar.schroeter@ovgu.de
saake@iti.cs.uni-magdeburg.de

ABSTRACT

Highly-configurable systems encompass thousands of interdependent configuration options, which require a non-trivial configuration process. Decision propagation enables a backtracking-free configuration process by computing values implied by user decisions. However, employing decision propagation for large-scale systems is a time-consuming task and, thus, can be a bottleneck in interactive configuration processes and analyses alike. We propose modal implication graphs to improve the performance of decision propagation by precomputing intermediate values used in the process. Our evaluation results show a significant improvement over state-of-the-art algorithms for 120 real-world systems.

CCS CONCEPTS

• **Software and its engineering → Software product lines**;

KEYWORDS

Software product line, Configuration, Decision Propagation

ACM Reference Format:
Sebastian Krieter, Thomas Thüm, Sandro Schulze, Reimar Schröter, and Gunter Saake. 2018. Propagating Configuration Decisions with Modal Implication Graphs. In *ICSE '18: ICSE '18: 40th International Conference on Software Engineering , May 27-June 3, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3180155.3180159

1 INTRODUCTION

Highly-configurable systems consist of thousands of configuration options also known as *features* [16, 18, 81]. This enormous and even growing amount of variability poses challenges for established algorithms used to analyze configurable systems [12, 89]. In particular, the variability analysis of large-scale systems, including their configuration, is still challenging as these tasks are computationally complex problems.

The features of a configurable system are typically connected by interdependencies that result from interactions within the system [64]. Examples of these dependencies are features that require another feature's functionality and features that are mutually exclusive [49]. In order to configure a working system variant, all dependencies of a configurable system must be considered. Thus, every decision a user makes in a configuration (i.e., selecting a feature) can imply to the inclusion or exclusion of other features.

During the configuration process it is often critical for users to immediately know the consequences of their decisions to avoid unwanted effects later on. For example, some users aim to configure a server system with a certain operating system and traffic monitoring. However, their chosen monitoring application is incompatible with their operating system. If they are unaware of such dependencies, their configured system variant is invalid. As real systems may contain thousands of interdependent configuration options, finding contradictions within a configuration manually is not feasible.

Decision propagation guarantees that users are informed about all consequences of their decisions at any point during the configuration process. Decision propagation determines the features that are implied or excluded by user decisions [42, 43, 59]. In an interactive configuration process, decision propagation prevents users from making contradictory decisions and reduces the amount of decisions a user has to make. By employing decision propagation in our example, users, who chose a particular monitoring application or operating system, can immediately notice the respective dependency and adjust their configuration accordingly (e.g., by choosing an alternative monitoring application).

Decision propagation is a computationally expensive task. In general, decision propagation is NP-hard as it involves finding valid assignments for interdependent boolean variables, also known as the boolean satisfiability problem (SAT), which is NP-complete [25]. With FeatureIDE, we have implemented decision propagation ten years ago and did not face scalability problems while using smaller feature models. However, when our industry partner used FeatureIDE with systems having more than 18,000 features, propagation of a single decision took over 20 seconds on average, summing up to 13 hours to create one configuration without even considering the time required to reason about decisions and to interact with the tool. While modern decision-propagation techniques can reduce this time to a feasible level for human interaction, decision propagation is still a bottleneck within automated configuration processes such as t-wise sampling [2].

ICSE '18, May 27-June 3, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5638-1/18/05…$15.00
https://doi.org/10.1145/3180155.3180159

2018 ACM/IEEE 40th International Conference on Software Engineering

Propagating Configuration Decisions with Modal Implication Graphs

Sebastian Krieter
University of Magdeburg, Germany
Harz University of Applied Sciences
Wernigerode, Germany
sebastian.krieter@ovgu.de

Thomas Thüm
TU Braunschweig, Germany
t.thuem@tu-braunschweig.de

Sandro Schulze
Reimar Schröter
Gunter Saake
University of Magdeburg, Germany
sandro.schulze@ovgu.de
reimar.schroeter@ovgu.de
saake@iti.cs.uni-magdeburg.de

ABSTRACT

Highly-configurable systems encompass thousands of interdependent configuration options, which require a non-trivial configuration process. Decision propagation enables a backtracking-free configuration process by computing values implied by user decisions. However, employing decision propagation for large-scale systems is a time-consuming task and, thus, can be a bottleneck in interactive configuration processes and analyses alike. We propose modal implication graphs to improve the performance of decision propagation by precomputing intermediate values used in the process. Our evaluation results show a significant improvement over state-of-the-art algorithms for 120 real-world systems.

CCS CONCEPTS

• **Software and its engineering → Software product lines**;

KEYWORDS

Software product line, Configuration, Decision Propagation

ACM Reference Format:
Sebastian Krieter, Thomas Thüm, Sandro Schulze, Reimar Schröter, and Gunter Saake. 2018. Propagating Configuration Decisions with Modal Implication Graphs. In *ICSE '18: ICSE '18: 40th International Conference on Software Engineering , May 27-June 3, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3180155.3180159

1 INTRODUCTION

Highly-configurable systems consist of thousands of configuration options also known as *features* [16, 18, 81]. This enormous and even growing amount of variability poses challenges for established algorithms used to analyze configurable systems [12, 89]. In particular, the variability analysis of large-scale systems, including their configuration, is still challenging as these tasks are computationally complex problems.

The features of a configurable system are typically connected by interdependencies that result from interactions within the system [64]. Examples of these dependencies are features that require another feature's functionality and features that are mutually exclusive [49]. In order to configure a working system variant, all dependencies of a configurable system must be considered. Thus, every decision a user makes in a configuration (i.e., selecting a feature) can imply to the inclusion or exclusion of other features.

During the configuration process it is often critical for users to immediately know the consequences of their decisions to avoid unwanted effects later on. For example, some users aim to configure a server system with a certain operating system and traffic monitoring. However, their chosen monitoring application is incompatible with their operating system. If they are unaware of such dependencies, their configured system variant is invalid. As real systems may contain thousands of interdependent configuration options, finding contradictions within a configuration manually is not feasible.

Decision propagation guarantees that users are informed about all consequences of their decisions at any point during the configuration process. Decision propagation determines the features that are implied or excluded by user decisions [42, 43, 59]. In an interactive configuration process, decision propagation prevents users from making contradictory decisions and reduces the amount of decisions a user has to make. By employing decision propagation in our example, users, who chose a particular monitoring application or operating system, can immediately notice the respective dependency and adjust their configuration accordingly (e.g., by choosing an alternative monitoring application).

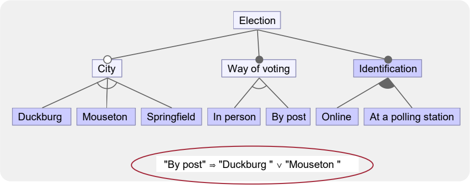
Decision propagation is a computationally expensive task. In general, decision propagation is NP-hard as it involves finding valid assignments for interdependent boolean variables, also known as the boolean satisfiability problem (SAT), which is NP-complete [25]. With FeatureIDE, we have implemented decision propagation ten years ago and did not face scalability problems while using smaller feature models. However, when our industry partner used FeatureIDE with systems having more than 18,000 features, propagation of a single decision took over 20 seconds on average, summing up to 13 hours to create one configuration without even considering the time required to reason about decisions and to interact with the tool. While modern decision-propagation techniques can reduce this time to a feasible level for human interaction, decision propagation is still a bottleneck within automated configuration processes such as t-wise sampling [2].

ICSE '18, May 27-June 3, 2018, Gothenburg, Sweden

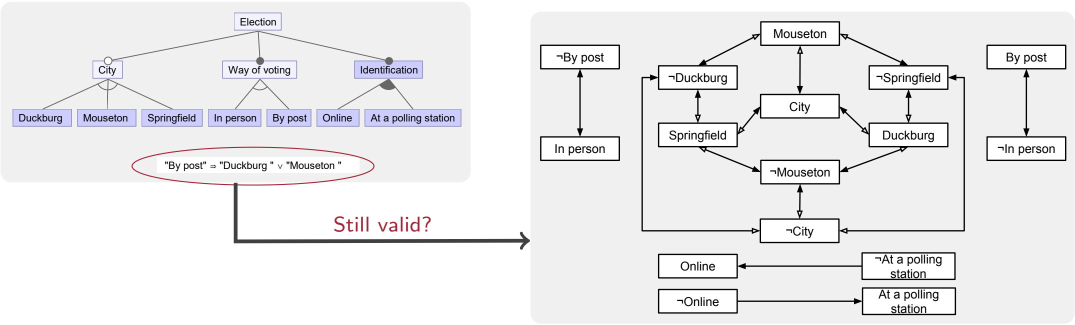
ACM ISBN 978-1-4503-5638-1/18/05…$15.00
https://doi.org/10.1145/3180155.3180159

Multiple times faster for decision propagation

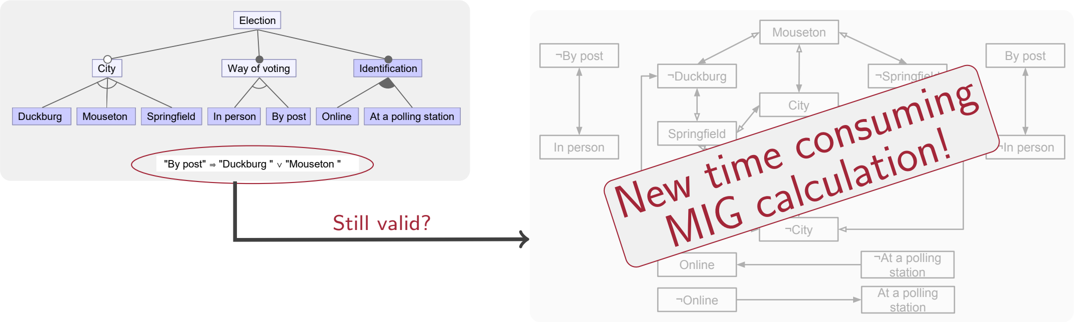
Incremental Modal Implication Graph Motivation



Incremental Modal Implication Graph Motivation



Incremental Modal Implication Graph Motivation



Technische Universität Braunschweig

Bachelor's Thesis

Incremental Construction of Modal Implication Graphs for Feature-Model Evolution

Author:

Rahel Arens

January 08, 2021

Advisors:

Prof. Dr.-Ing. Ina Schaefer

Michael Nieke, M.Sc.

Institute of Software Engineering and Automotive Informatics
TU Braunschweig

Prof. Dr.-Ing. Thomas Thüm

Institute of Software Engineering and Programming Languages
Ulm University

Institut für Softwaretechnik
und Fahrzeuginformatik

ISF

Technische Universität Braunschweig

Bachelor's Thesis

Incremental Construction of Modal Implication Graphs for Feature-Model Evolution

Author:

Rahel Arens

January 08, 2021

Advisors:

Prof. Dr.-Ing. Ina Schaefer

Michael Nieke, M.Sc.

Institute of Software Engineering and Automotive Informatics
TU Braunschweig

Prof. Dr.-Ing. Thomas Thüm

Institute of Software Engineering and Programming Languages
Ulm University

ISF
Institut für Softwaretechnik
und Fahrzeuginformatik

Incremental Construction of Modal Implication Graphs for Evolving Feature Models

Sebastian Krieter
Harz University of Applied Sciences
Wernigerode, Germany

Rahel Arens*
TU Braunschweig
Brunswick, Germany

Michael Nieke†
ITU Copenhagen
Copenhagen, Denmark

Chico Sundermann
Tobias Heß
University of Ulm
Ulm, Germany

Thomas Thüm
University of Ulm
Ulm, Germany

Christoph Seidl
ITU Copenhagen
Copenhagen, Denmark

ABSTRACT

A feature model represents a set of variants as configurable features and dependencies between them. During variant configuration, (de)selection of a feature may entail that other features must or cannot be selected. A Modal Implication Graph (MIG) enables efficient decision propagation to perform automatic (de)selection of subsequent features. In addition, it facilitates other configuration-related activities such as t-wise sampling. Evolution of a feature model may change its configuration logic, thereby invalidating an existing MIG and forcing a full recomputation. However, repeated recomputation of a MIG is expensive, and thus hampers the overall usefulness of MIGs for frequently evolving feature models. In this paper, we devise a method to incrementally compute updated MIGs after feature model evolution. We identify expensive steps in the MIG construction algorithm, enable them for incremental computation, and measure performance compared to a full rebuild of a complete MIG within the evolution histories of four real-world feature models. Results show that our incremental method can increase the speed of MIG construction by orders of magnitude, depending on the given scenario and extent of evolutionary changes.

CCS CONCEPTS

• **Software and its engineering → Software product lines**.

KEYWORDS

Configurable System, Software Product Line, Evolution

ACM Reference Format:
Sebastian Krieter, Rahel Arens, Michael Nieke, Chico Sundermann, Tobias Heß, Thomas Thüm, and Christoph Seidl. 2021. Incremental Construction of Modal Implication Graphs for Evolving Feature Models . In *25th ACM International Systems and Software Product Line Conference - Volume A (SPLC '21), September 6–11, 2021, Leicester, United Kingdom*. ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3461001.3471148

*Also with University of Ulm.
†Also with TU Braunschweig.

SPLC '21, September 6–11, 2021, Leicester, United Kingdom

ACM ISBN 978-1-4503-8469-8/21/09…$15.00
https://doi.org/10.1145/3461001.3471148

1 INTRODUCTION

A *Software Product Line (SPL)* captures a family of closely related software *variants* [2, 9, 11, 32]. On a conceptual level, a variant is defined via a *configuration* comprised of configuration options, i.e., selected or deselected *features* [2, 9, 11, 32]. A *feature model* captures features of an SPL and their relations, such as implications and exclusions, as *constraints* [2, 9, 11, 32]. Real-world feature models commonly grow large, resulting in a massive number of features and complex constraints [6, 7, 34]. As a consequence, defining a valid configuration is challenging for engineers, because they have to obey all constraints when (de)selecting features. Related applications, such as configuration sampling [10, 20, 23, 36], suffer from similar challenges to derive and reason on (many) valid configurations. To support the configuration process, a *Modal Implication Graph (MIG)* facilitates efficient *decision propagation* by directly modeling the impact of feature (de)selections and automatically (de)selecting subsequent features [20, 21]. While a generated MIG can be reused to support an unlimited number of configuration processes, it has to be specifically tailored to encode the configuration logic of a particular feature model, which entails significant computational cost. Feature model evolution may change a feature model or its constraints and, subsequently, invalidate an existing MIG that then represents outdated configuration logic. For feature models that frequently evolve, it is costly to perform a full rebuild of a complete MIG after each evolution step [21]. Thus, in the light of frequent feature model evolution, reaping the benefits of a MIG for interactive configurations or sampling configurations is, currently, severely hampered if not outright infeasible.

In this paper, we present a method to incrementally create a MIG for an evolving feature model. After feature-model evolution, we reuse information from a previously built MIG and compute the impact of the feature-model changes on the MIG. To this end, we identify which steps of the original MIG creation algorithm are the most costly and can benefit from incremental computation. We define an overall incremental build process for MIGs consisting of steps from the original build process as well as incremental steps that reuse information from a previous MIG. Thus, we enable efficient creation of MIGs in presence of feature-model evolution. Our method makes the benefits of MIGs accessible even for frequently changing feature models. We evaluate our method by comparing

Incremental Modal Implication Graph Solution

1. Detect CNF changes

Incremental Modal Implication Graph Solution

1. Detect CNF changes

2. Update core and dead features

Incremental Modal Implication Graph Solution

1. Detect CNF changes
2. Update core and dead features
3. Detect redundant clauses

Incremental Modal Implication Graph Solution

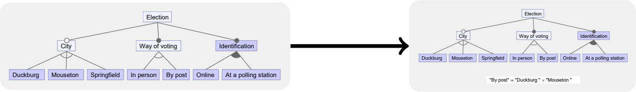
1. Detect CNF changes
2. Update core and dead features
3. Detect redundant clauses
4. Update MIG

Incremental Modal Implication Graph Solution

1. Detect CNF changes
2. Update core and dead features
3. Detect redundant clauses
4. Update MIG
5. Calculate implicit strong edges

Incremental Modal Implication Graph Solution

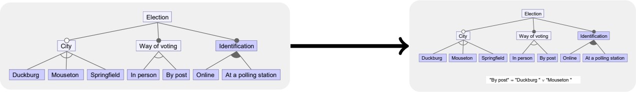
1. Detect CNF changes
2. Update core and dead features
3. Detect redundant clauses
4. Update MIG
5. Calculate implicit strong edges



Incremental Modal Implication Graph Solution

1. Detect CNF changes
2. Update core and dead features
3. Detect redundant clauses
4. Update MIG
5. Calculate implicit strong edges

Added clauses: $\{\neg$ By post, Duckburg, Mouseton$\}$
Removed clauses: $\emptyset$



Incremental Modal Implication Graph Solution

1. Detect CNF changes

2. Update core and dead features

Calculate from scratch

3. Detect redundant clauses

4. Update MIG

5. Calculate implicit strong edges

Incremental Modal Implication Graph Solution

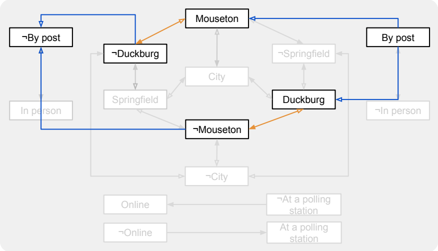
1. Detect CNF changes
2. Update core and dead features
3. Detect redundant clauses
4. Update MIG
5. Calculate implicit strong edges

Evaluated different approaches:
Incremental vs Complete

Incremental Modal Implication Graph Solution

1. Detect CNF changes
2. Update core and dead features
3. Detect redundant clauses
4. Update MIG
5. Calculate implicit strong edges

Added clauses: $\{\neg$ By post, Duckburg, Mouseton$\}$
Removed clauses: $\emptyset$



Incremental Modal Implication Graph Solution

1. Detect CNF changes

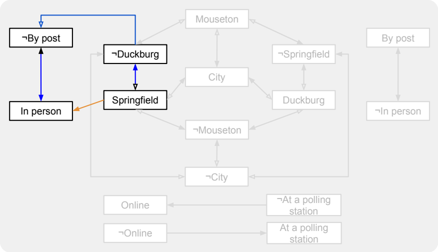
2. Update core and dead features

3. Detect redundant clauses

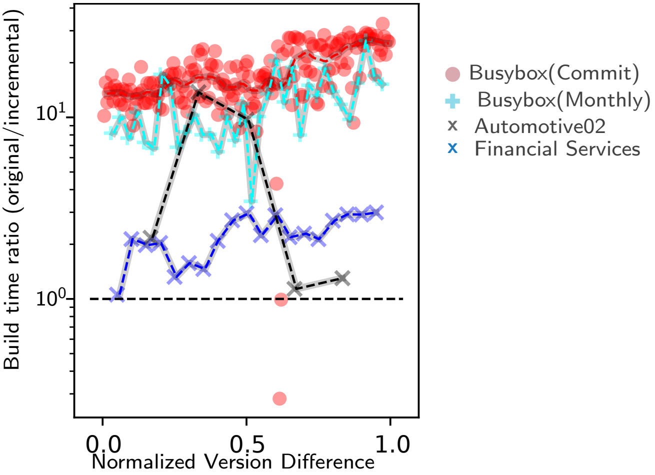
4. Update MIG

5. Calculate implicit strong edges

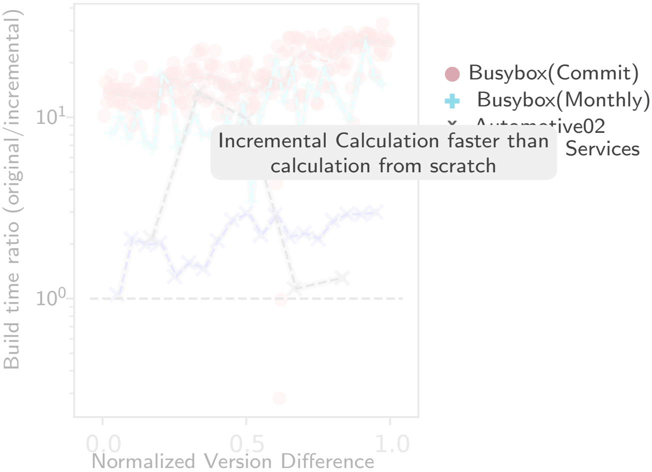
Added clauses: $\{\neg$ By post, Duckburg, Mouseton$\}$
Removed clauses: $\emptyset$



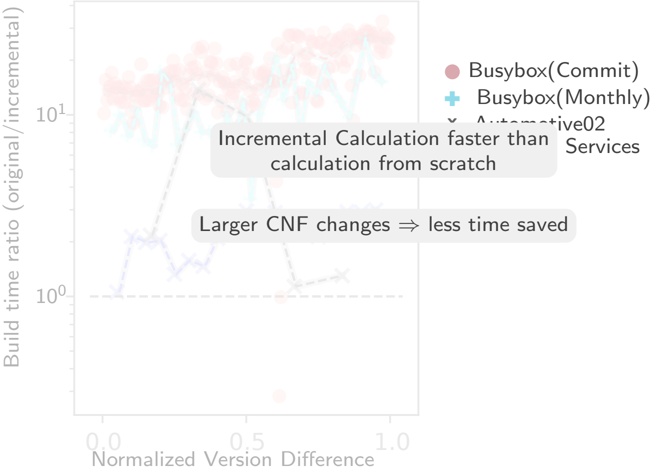
Incremental Modal Implication Graph Evaluation



Incremental Modal Implication Graph Evaluation



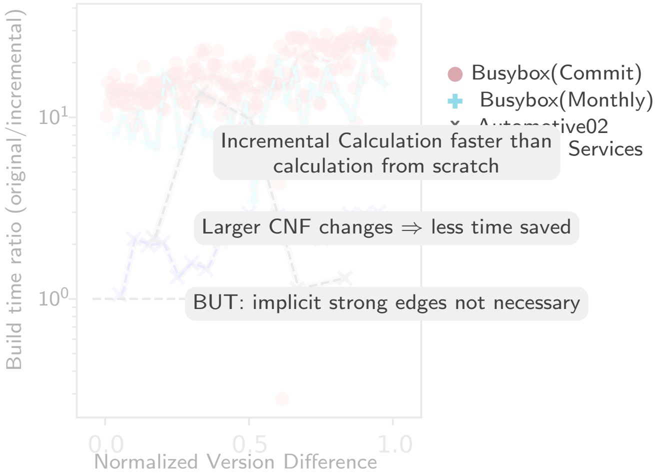
Incremental Modal Implication Graph Evaluation



Incremental Calculation faster than calculation from scratch

Larger CNF changes ⇒ less time saved

Incremental Modal Implication Graph Evaluation

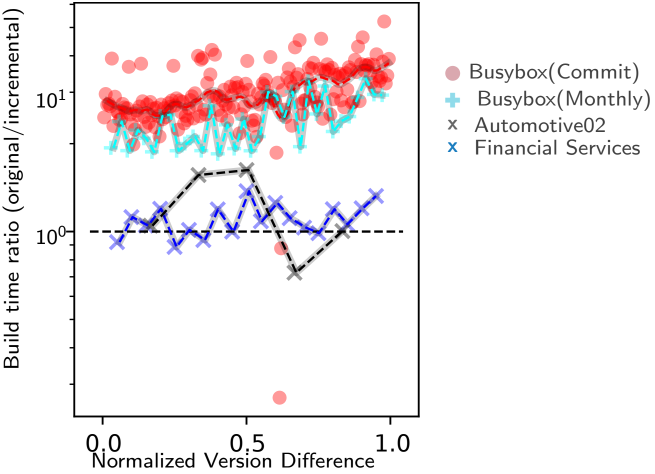


Incremental Calculation faster than calculation from scratch

Larger CNF changes ⇒ less time saved

BUT: implicit strong edges not necessary

Incremental Modal Implication Graph Evaluation

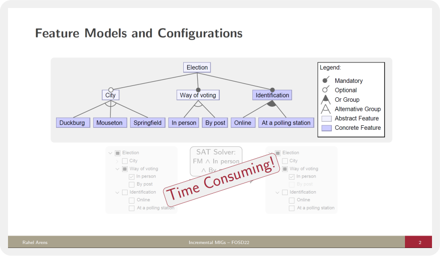


Future Work

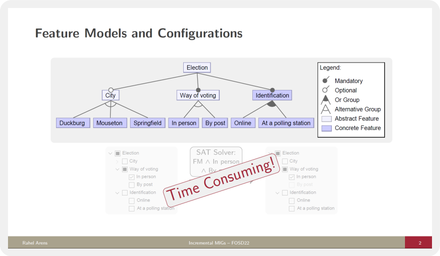
Break-even point
Incremental vs from scratch calculation

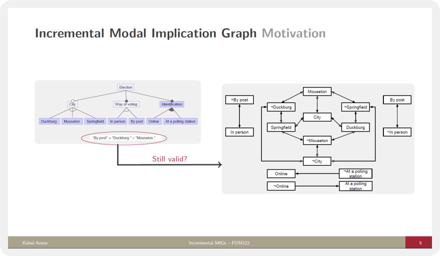
Further SAT-based analyses

Summary

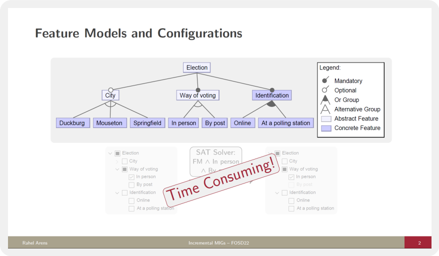

Feature Models and Configurations
Election
City
Way of voting
Identification
Duckburg
Mouseton
Springfield
In person
By post
Online
At a polling station
Legend:
Mandatory
Optional
Or Group
Alternative Group
Abstract Feature
Concrete Feature
SAT Solver:
FM ∧ In person
Time Consuming!
Rahel Arens
Incremental MIGs – FOSD22
2

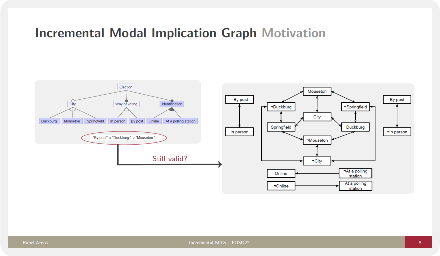
Summary

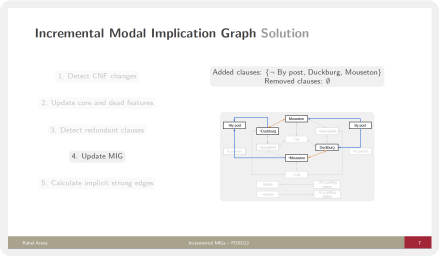

Feature Models and Configurations
Election
City
Way of voting
Identification
Duckburg
Mouseton
Springfield
In person
By post
Online
At a polling station
Legend:
Mandatory
Optional
Or Group
Alternative Group
Abstract Feature
Concrete Feature
SAT Solver:
FM ∧ In person
Time Consuming!
Rahel Arens
Incremental MIGs – FOSD22
2


Incremental Modal Implication Graph Motivation
Still valid?
Mouseton
¬By post
In person
By post
¬In person
¬Duckburg
¬Springfield
City
Springfield
Duckburg
¬Mouseton
¬City
Online
¬Online
¬At a polling station
At a polling station
Rahel Arens
Incremental MIGs – FOSD22
5

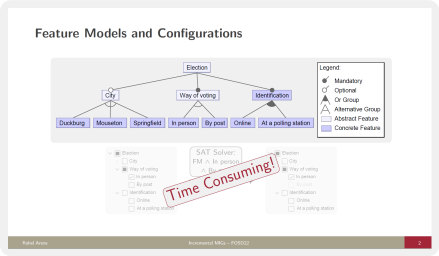
Summary

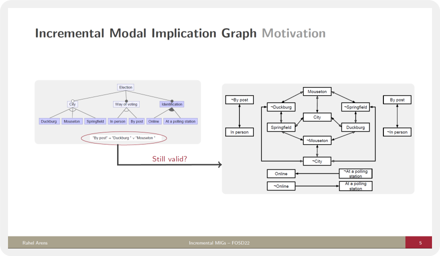

Feature Models and Configurations
Election
City
Way of voting
Identification
Duckburg
Mouseton
Springfield
In person
By post
Online
At a polling station
Legend:
Mandatory
Optional
Or Group
Alternative Group
Abstract Feature
Concrete Feature
SAT Solver:
Time Consuming!
Rahel Arens
Incremental MIGs – FOSD22
2

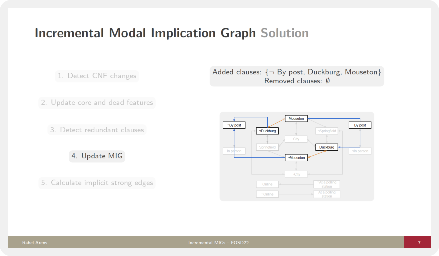

Incremental Modal Implication Graph Motivation
Still valid?
Mouseton
¬By post
¬Duckburg
¬Springfield
By post
City
In person
Springfield
Duckburg
¬In person
¬Mouseton
¬City
Online
¬At a polling station
¬Online
At a polling station
Rahel Arens
Incremental MIGs – FOSD22
5

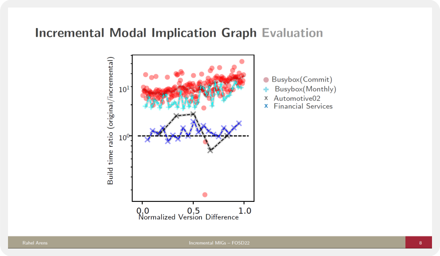

Incremental Modal Implication Graph Solution
1. Detect CNF changes
2. Update core and dead features
3. Detect redundant clauses
4. Update MIG
5. Calculate implicit strong edges
Added clauses: {¬ By post, Duckburg, Mouseton}
Removed clauses: ∅
Rahel Arens
Incremental MIGs – FOSD22
7

Summary


Feature Models and Configurations
Election
City
Way of voting
Identification
Duckburg
Mouseton
Springfield
In person
By post
Online
At a polling station
Legend:
Mandatory
Optional
Or Group
Alternative Group
Abstract Feature
Concrete Feature
SAT Solver:
Time Consuming!
Rahel Arens
Incremental MIGs – FOSD22
2


Incremental Modal Implication Graph Motivation
Still valid?
Mouseton
¬By post
By post
¬Duckburg
¬Springfield
City
In person
Springfield
Duckburg
¬In person
¬Mouseton
¬City
Online
¬At a polling station
¬Online
At a polling station
Rahel Arens
Incremental MIGs – FOSD22
5


Incremental Modal Implication Graph Solution
1. Detect CNF changes
2. Update core and dead features
3. Detect redundant clauses
4. Update MIG
5. Calculate implicit strong edges
Added clauses: {¬ By post, Duckburg, Mouseton}
Removed clauses: ∅
Rahel Arens
Incremental MIGs – FOSD22
7


Incremental Modal Implication Graph Evaluation
Build time ratio (original/incremental)
10^1
10^0
0.0
0.5
1.0
Normalized Version Difference
Busybox(Commit)
Busybox(Monthly)
Automotive02
Financial Services
Rahel Arens
Incremental MIGs – FOSD22
8

Feature Model	#Features	#Clauses	MIG Memory (Byte)	Offline time in s (∅)			∑ Online time in s for relative number of defined features (∅) 3%			10%			100%		
				ASAT	IMIG	CMIG	ASAT	IMIG	CMIG	ASAT	IMIG	CMIG	ASAT	IMIG	CMIG
FreeBSD 8.0.0	1,397	14,295	243,168	0.04	0.42	6.89	0.21	0.05	0.04	0.44	0.07	0.05	1.82	0.10	0.08
Automotive01	2,513	2,833	1,098,248	0.07	0.70	11.60	1.54	0.36	0.35	3.10	0.56	0.54	6.05	0.76	0.74
Linux 2.6.28.6	6,889	80,715	2,157,320	0.27	16.75	399.98	11.78	5.75	4.24	26.98	8.39	5.63	80.67	10.07	6.66
Automotive02	18,616	1,369	5,088,720	2.30	42.47	296.73	329.29	38.08	37.84	535.70	58.77	58.45	821.48	68.03	67.68
All models (∅)	–	–	–	0.03	0.62	6.99	2.98	0.38	0.36	4.96	0.58	0.55	8.10	0.68	0.64

System	Parameters	Build Time					Usage Time				
		Orig (s)	Inc (s)	Ratio			Orig (s)	Inc (s)	Ratio		
		∅	∅	*Min*	∅	*Max*	∅	∅	*Min*	∅	*Max*
Busybox (Commits)	1	0.002	0.003	0.478	0.823	1.690	0.034	0.034	0.771	0.997	1.052
	2	0.012	0.004	0.311	3.176	6.333	0.034	0.034	0.662	0.997	1.050
	3	0.012	0.006	0.275	2.416	5.266	0.034	0.034	0.959	1.002	1.070
	4	0.086	0.018	0.283	12.732	32.996	0.034	0.034	0.955	1.000	1.204
	5	0.086	0.004	0.301	21.094	47.502	0.034	0.034	0.948	1.001	1.191
Busybox (Monthly)	1	0.003	0.003	0.589	0.766	0.919	0.040	0.040	0.973	0.999	1.026
	2	0.013	0.004	1.998	2.981	4.226	0.040	0.040	0.972	0.999	1.031
	3	0.013	0.008	0.911	1.990	3.490	0.040	0.040	0.970	0.997	1.021
	4	0.102	0.025	1.296	8.614	26.702	0.041	0.041	0.966	1.014	1.058
	5	0.102	0.005	10.534	19.867	37.456	0.041	0.040	0.986	1.034	1.056
FinacialServices01	1	0.161	0.165	0.925	0.977	1.031	0.198	0.197	0.929	1.006	1.069
	2	0.345	0.182	1.336	1.913	2.348	0.195	0.197	0.934	0.988	1.030
	3	0.341	0.333	0.841	1.038	1.392	0.196	0.197	0.956	0.995	1.049
	4	18.809	13.218	0.969	1.604	2.992	0.195	0.193	0.963	1.009	1.060
	5	18.837	6.792	0.853	9.975	23.033	0.195	0.192	0.974	1.017	1.051
Automotive02	1	3.109	3.535	0.822	0.882	0.958	7.490	7.502	0.992	0.999	1.005
	2	8.583	3.873	1.672	2.120	3.395	7.451	7.499	0.985	0.994	0.999
	3	8.561	9.120	0.690	1.080	1.616	7.447	7.458	0.993	0.998	1.005
	4	2090.844	1547.998	1.069	4.244	13.733	7.353	7.365	0.992	0.998	1.005
	5	2095.159	4.270	110.037	424.784	1221.410	7.362	7.471	0.977	0.986	1.001
Linux	1	29.326	29.446	0.829	0.998	1.127	15.686	15.677	0.992	1.001	1.010
	2	2218.864	132.196	15.514	16.806	18.269	15.667	15.664	0.993	1.000	1.004
	3	2228.968	2845.951	0.748	0.783	0.815	15.671	15.665	0.993	1.000	1.008