Learning from Difference: An Automated Approach for Learning Family Models from Software Product Lines

Carlos Diego N. Damasceno
damascenodiego@usp.br
University of Sao Paulo, BR and
University of Leicester, UK

Mohammad Reza Mousavi
mm789@leicester.ac.uk
University of Leicester
Leicester,UK

Adenilso Simao
adenilso@icmc.usp.br
University of Sao Paulo (USP)
São Carlos, SP, BR

September 6, 2019

Research objective

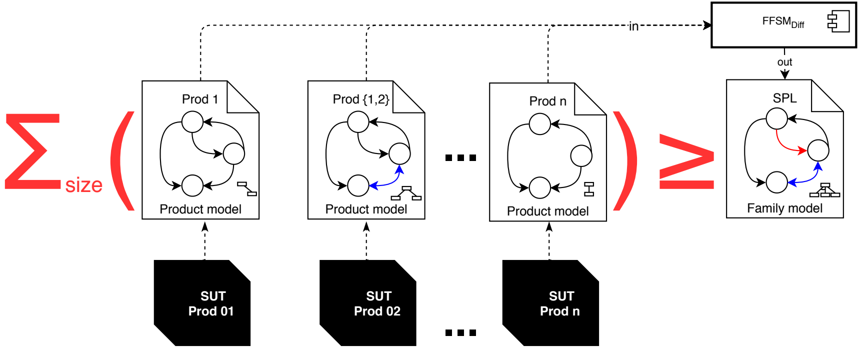

Figure: Introduce an approach for learning succinct family models from products specifications

Featured Finite State Machines

Arcade Game Maker

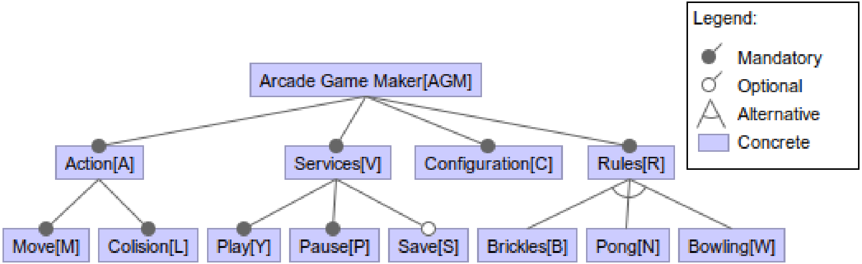

Figure: The Arcade Game Maker SPL

Featured Finite State Machine (FFSM)

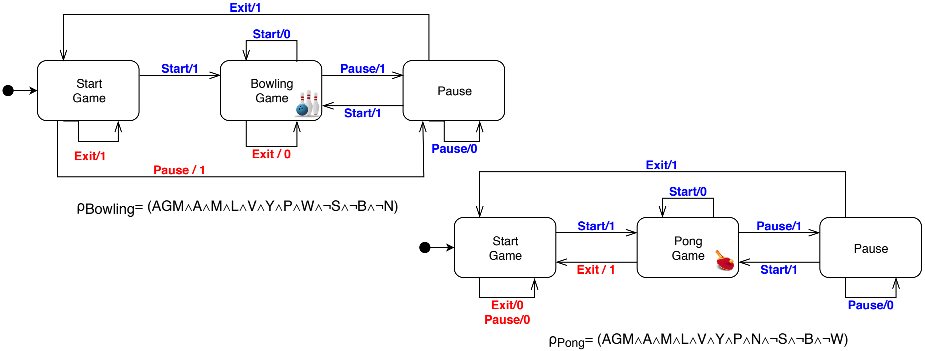

Figure: An FFSM unifies multiple finite state machines of a product-line into a single model where states and transitions are annotated with feature constraints (i.e., **conditional states** and **conditional transitions**) [1]

[1]Fragal et al. (2017)

Featured Finite State Machine (FFSM)

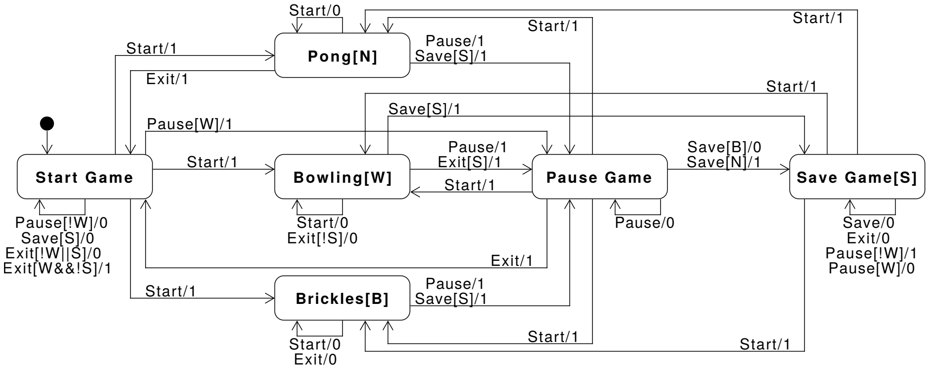

Figure: An FFSM unifies multiple finite state machines of a product-line into a single model where states and transitions are annotated with feature constraints (i.e., **conditional states** and **conditional transitions**) [1]

[1] Fragal et al. (2017)

Featured Finite State Machine (FFSM)

- Family-based analysis (e.g., model-based testing [2] and model checking [3])
- **Cost** as a function of the **number of features** and **amount of feature sharing**
- **Redundant analysis** are **avoided/minimised**

- **Creation and maintenance of family models** is challenging
- **Outdated family models** may arise as product instances evolve

[2] Fragal et al. (2017)
[3] ter Beek et al. (2017)

The $FFSM_{Diff}$ algorithm

The $FFSM_{Diff}$ algorithm

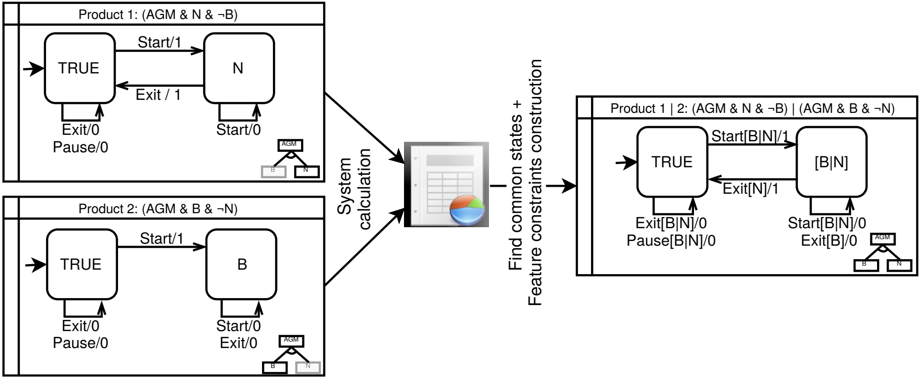

Figure: An automated technique to learn **fresh FFSM** and **include new FSMs into existing FFSMs** by **comparing products models** and **incorporating variability** to express product-specific behaviors with feature constraints

The $FFSM_{Diff}$ algorithm – Global similarity score

$$S^G_{Succ}(a,b) = \frac{1}{2}\frac{\sum_{(c,d,i,o)\,\in\,Succ_{a,b}}(1 + k \times S^G_{Succ}(c,d))}{|\sum_r^{out}(a) - \sum_u^{out}(b)| + |\sum_r^{out}(b) - \sum_u^{out}(a)| + |Succ_{a,b}|}$$

Figure: Global similarity score [4]

Global similarity score (Outgoing and incoming transitions)

- Pairwise similarity based on surrounding matching transitions and connected state pairs.
- Attenuation ratio k gives precedence to the closest state pairs.
- Matching transitions and distinct transitions.

[4] Walkinshaw and Bogdanov (2013)

The $FFSM_{Diff}$ algorithm – Global similarity score

$$S^G_{Succ}(a,b) = \frac{1}{2}\frac{\sum_{(c,d,i,o)\,\in\,Succ_{a,b}}(1 + k \times S^G_{Succ}(c,d))}{|\sum_r^{out}(a) - \sum_u^{out}(b)| + |\sum_r^{out}(b) - \sum_u^{out}(a)| + |Succ_{a,b}|}$$

Figure: Global similarity score [4]

Global similarity score (Outgoing and incoming transitions)

- Pairwise similarity based on surrounding matching transitions and connected state pairs.
- Attenuation ratio k gives precedence to the closest state pairs.
- Matching transitions and distinct transitions.

[4] Walkinshaw and Bogdanov (2013)

The $FFSM_{Diff}$ algorithm – Global similarity score

$$S^G_{Succ}(a,b) = \frac{1}{2}\frac{\sum_{(c,d,i,o)\,\in\,Succ_{a,b}}(1 + k \times S^G_{Succ}(c,d))}{|\sum_r^{out}(a) - \sum_u^{out}(b)| + |\sum_r^{out}(b) - \sum_u^{out}(a)| + |Succ_{a,b}|}$$

Figure: Global similarity score [4]

Global similarity score (Outgoing and incoming transitions)

- Pairwise similarity based on surrounding matching transitions and connected state pairs.
- Attenuation ratio k gives precedence to the closest state pairs.
- Matching transitions and distinct transitions.

[4]Walkinshaw and Bogdanov (2013)

The $FFSM_{Diff}$ algorithm – Global similarity score

$$S^G_{Succ}(a,b) = \frac{1}{2}\frac{\sum_{(c,d,i,o)\,\in\, Succ_{a,b}}(1 + k \times S^G_{Succ}(c,d))}{|\sum_r^{out}(a) - \sum_u^{out}(b)| + |\sum_r^{out}(b) - \sum_u^{out}(a)| + |Succ_{a,b}|}$$

Figure: Global similarity score [4]

Global similarity score (Outgoing and incoming transitions)

- Pairwise similarity based on surrounding matching transitions and connected state pairs.
- Attenuation ratio k gives precedence to the closest state pairs.
- Matching transitions and distinct transitions.

[4]Walkinshaw and Bogdanov (2013)

The $FFSM_{Diff}$ algorithm – Global similarity score

$$S^G_{Succ}(a,b) = \frac{1}{2}\frac{\sum_{(c,d,i,o)\,\in\,Succ_{a,b}}(1 + k \times S^G_{Succ}(c,d))}{|\sum_r^{out}(a) - \sum_u^{out}(b)| + |\sum_r^{out}(b) - \sum_u^{out}(a)| + |Succ_{a,b}|}$$

Figure: Global similarity score [4]

Global similarity score (Outgoing and incoming transitions)

- Pairwise similarity based on surrounding matching transitions and connected state pairs.
- Attenuation ratio k gives precedence to the closest state pairs.
- Matching transitions and distinct transitions.

[4] Walkinshaw and Bogdanov (2013)

The $FFSM_{Diff}$ algorithm – Similarity score (Running example)

$$S^G_{Succ}(St, St) = \frac{1}{2} \times \frac{1 + k \times S^G_{Succ}(Bo, Po)}{2 + 2 + 1} = 0.12$$

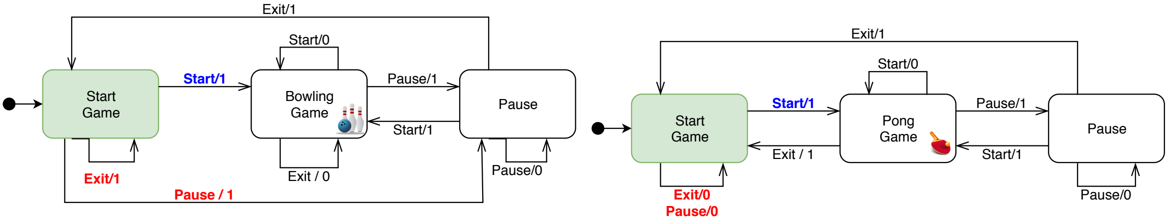

Figure: Two examples of product FSMs and their similarity scores

The $FFSM_{Diff}$ algorithm – Similarity score (Running example)

$$S^G_{Succ}(Pa, Pa) = \frac{1}{2} \times \frac{3 + k \times [S^G_{Succ}(St, St) + S^G_{Succ}(Bo, Po) + S^G_{Succ}(Pa, Pa)]}{0 + 0 + 3} = 0.58$$

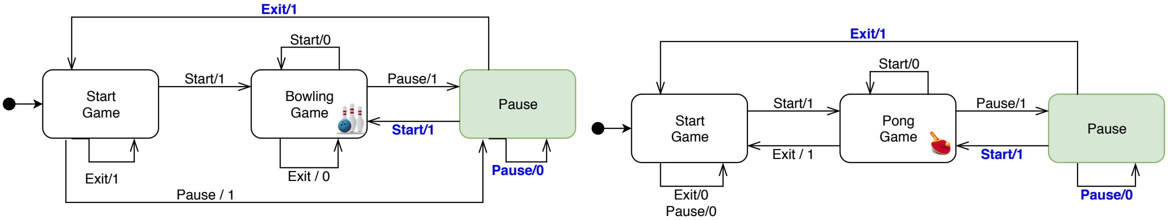

Figure: Two examples of product FSMs and their similarity scores

The $FFSM_{Diff}$ algorithm – State similarity (Running example)

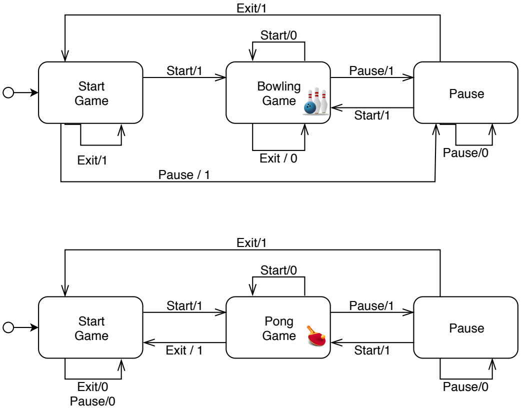

Figure: Two examples of product FSMs

$$\mathrm{pair(St,\ St)} = 0.12$$
$$pair(St, Po) = 0.29$$
$$pair(St, Pa) = 0.28$$
$$pair(Bo, St) = 0.11$$
$$pair(Bo, Po) = 0.31$$
$$pair(Bo, Pa) = 0$$
$$pair(Pa, St) = 0.29$$
$$pair(Pa, Po) = 0.11$$
$$pair(Pa, Pa) = 0.58$$

Figure: Pairwise state similarity

The $FFSM_{Diff}$ algorithm – State similarity (Running example)

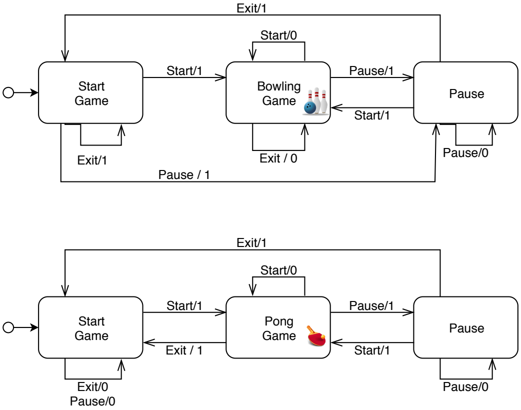

Figure: Two examples of product FSMs

$\text{pair(St, St)} = 0.12$

$pair(St, Po) = 0.29$

$pair(St, Pa) = 0.28$

$pair(Bo, St) = 0.11$

$pair(Bo, Po) = 0.31$

$pair(Bo, Pa) = 0$

$pair(Pa, St) = 0.29$

$pair(Pa, Po) = 0.11$

$pair(Pa, Pa) = 0.58$

Figure: Pairwise state similarity

The $FFSM_{Diff}$ algorithm – State similarity (Running example)

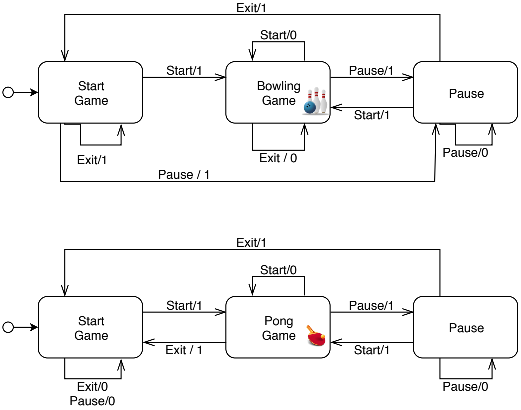

Figure: Two examples of product FSMs

$$pair(St, St) = 0.12$$
$$pair(St, Po) = 0.29$$
$$pair(St, Pa) = 0.28$$
$$pair(Bo, St) = 0.11$$
$$pair(Bo, Po) = 0.31$$
$$pair(Bo, Pa) = 0$$
$$pair(Pa, St) = 0.29$$
$$pair(Pa, Po) = 0.11$$
$$pair(Pa, Pa) = 0.58$$

Figure: Pairwise state similarity

The $FFSM_{Diff}$ algorithm – State similarity (Running example)

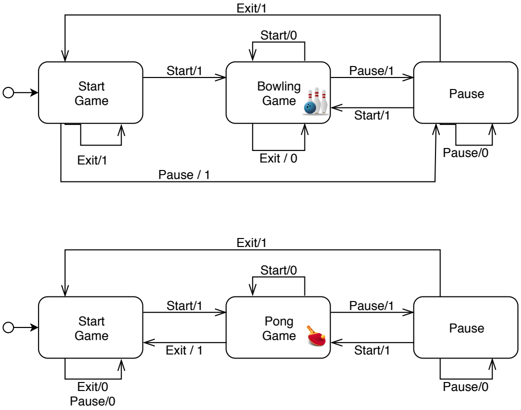

Figure: Two examples of product FSMs

$pair(St, St) = 0.12$

$pair(St, Po) = 0.29$

$pair(St, Pa) = 0.28$

$pair(Bo, St) = 0.11$

$pair(Bo, Po) = 0.31$

$pair(Bo, Pa) = 0$

$pair(Pa, St) = 0.29$

$pair(Pa, Po) = 0.11$

$pair(Pa, Pa) = 0.58$

Figure: Pairwise state similarity

The $FFSM_{Diff}$ algorithm – Incorporating variability

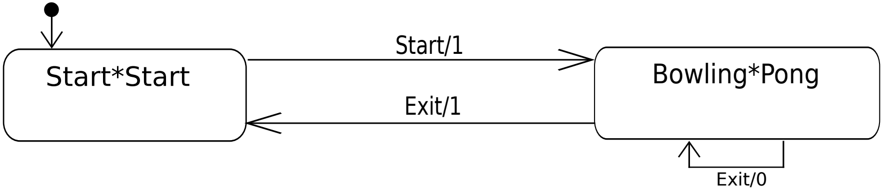

Figure: Fragment of the FFSM learnt from two products of the AGM SPL.

Product configuration – Example

$\rho_{Bowling} = (AGM \wedge A \wedge M \wedge L \wedge V \wedge Y \wedge P \wedge W \wedge \neg S \wedge \neg B \wedge \neg N)$
$\rho_{Pong} = (AGM \wedge A \wedge M \wedge L \wedge V \wedge Y \wedge P \wedge N \wedge \neg S \wedge \neg B \wedge \neg W)$

The $FFSM_{Diff}$ algorithm – Incorporating variability

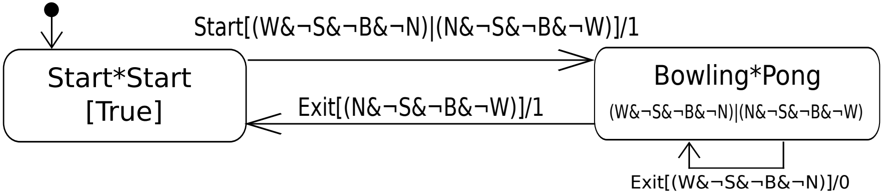

Figure: Fragment of the FFSM learnt from two products of the AGM SPL.

Simplified configuration – Example

$\rho_{Bowling} = (W \wedge \neg S \wedge \neg B \wedge \neg N)$
$\rho_{Pong} = (N \wedge \neg S \wedge \neg B \wedge \neg W)$

Empirical Evaluation

Empirical evaluation – Research Questions (RQ)

(RQ1) Is our automated technique effective in learning succinct family models compared to the total size of product analyzed?

(RQ2) Is the size of learnt family models influenced by the amount of feature reuse?

(RQ3) Is our automated technique effective in learning succinct family models compared to hand-crafted family models?

Empirical evaluation – Research Questions (RQ)

(RQ1) Is our automated technique effective in learning succinct family models compared to the total size of product analyzed?

- **Size of learnt FFSM $\leq$ Size of products pairs**

(RQ2) Is the size of learnt family models influenced by the amount of feature reuse?

(RQ3) Is our automated technique effective in learning succinct family models compared to hand-crafted family models?

Empirical evaluation – Research Questions (RQ)

(RQ1) Is our automated technique effective in learning succinct family models compared to the total size of product analyzed?

(RQ2) Is the size of learnt family models influenced by the amount of feature reuse?

- **Size of learnt FFSM vs. Feature reuse**

(RQ3) Is our automated technique effective in learning succinct family models compared to hand-crafted family models?

Empirical evaluation – Research Questions (RQ)

(RQ1) Is our automated technique effective in learning succinct family models compared to the total size of product analyzed?

(RQ2) Is the size of learnt family models influenced by the amount of feature reuse?

(RQ3) Is our automated technique effective in learning succinct family models compared to hand-crafted family models?

- Size of FFSMs learnt from the **whole family** vs. hand-crafted FFSM
- Size of FFSMs learnt from **products pairs** vs. hand-crafted FFSM

Empirical evaluation – Subject systems [5]

SPL		Number of		Sum total of states in	
ID	Name	Features	Valid config.	FFSM	All products
AGM	Arcade Game Maker	13	6	6	21
WS	Wiper System	8	8	13	56
VM	Vending Machine	9	20	14	207
Total number of products: 34 products					

Table: Description of the SPLs under learning

[5] Fragal et al. (2017); Classen (2010)

Analysis of Results **(FFSM learnt vs. Size of product pairs)**

RQ1) Is the *$FFSM_{Diff}$* algorithm effective in **learning succinct family models** compared to the **total size of the products**?

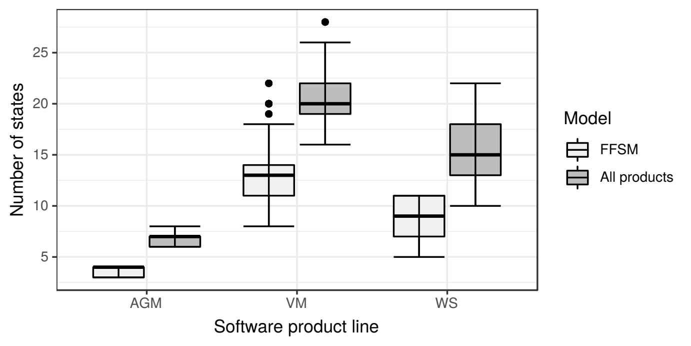

Figure: We found **statistically significant difference ($p < 0.01$)** and **large effect sizes** between the size of learnt FFSMs and total size of products pairs

Analysis of Results **(FFSM learnt vs. Size of product pairs)**

RQ1) Is the *$FFSM_{Diff}$* algorithm effective in **learning succinct family models** compared to the **total size of the products**?

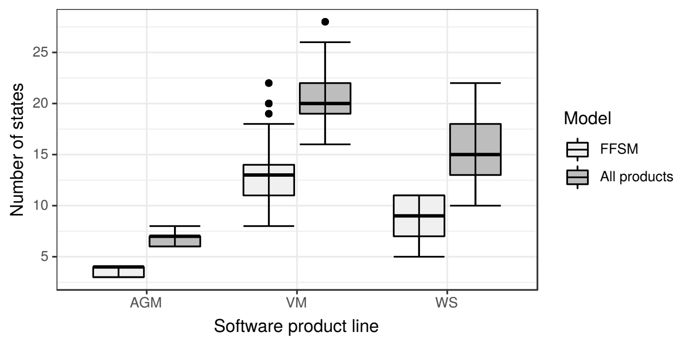

Figure: The size of learnt FFSMs is **at most equal** to the total size of products pairs analyzed

Analysis of Results **(FFSM learnt vs. Feature reuse)**

RQ2) Is the size of learnt family models influenced by the amount of feature reuse?

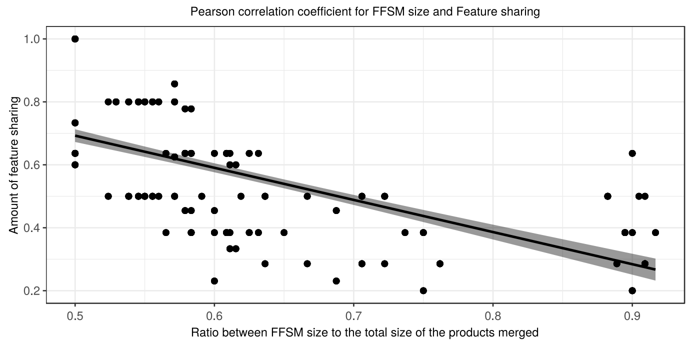

Figure: We found a **strong negative correlation** between FFSM size and amount of feature reuse

Analysis of Results **(FFSM learnt vs. Feature reuse)**

RQ2) Is the size of learnt family models influenced by the amount of feature reuse?

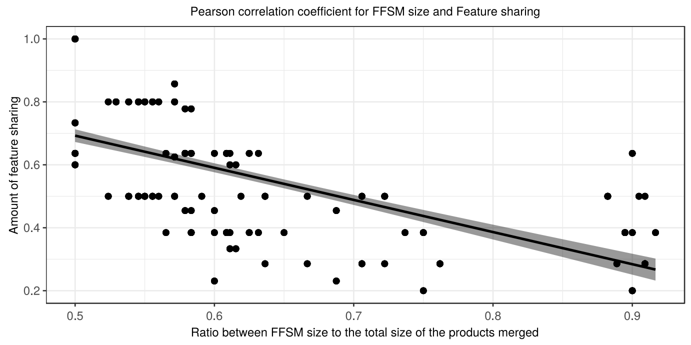

Figure: FFSMs learnt from products implementing a **similar set of features** tend to be **more succinct**

Analysis of Results **(FFSMs learnt from the whole family)**

RQ3) Is the $FFSM_{Diff}$ algorithm effective in **learning succinct family models** compared to **hand-crafted family models**?

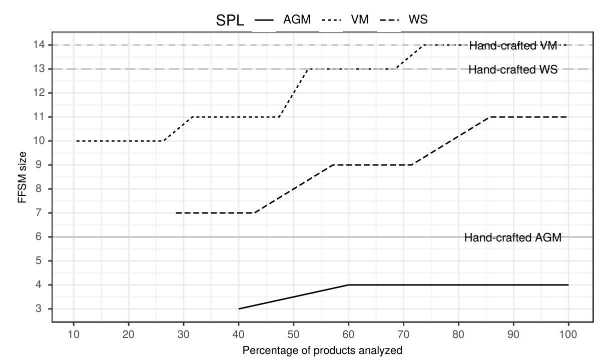

Figure: Two FFSMs were learnt with fewer states and one with the same original size

Analysis of Results **(FFSMs learnt from products pairs)**

RQ3) Is the $FFSM_{Diff}$ algorithm effective in **learning succinct family models** compared to **hand-crafted family models**?

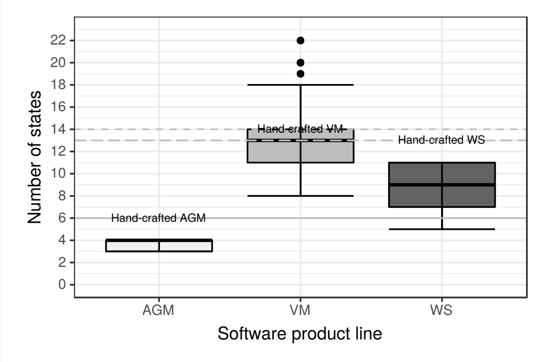

Figure: We found **significant differences** ($p < 0.01$) and **large effect sizes** for two SPLs

Analysis of Results **(FFSMs learnt from the AGM SPL)**

RQ3) Is the *$FFSM_{Diff}$* algorithm effective in **learning succinct family models** compared to **hand-crafted family models**?

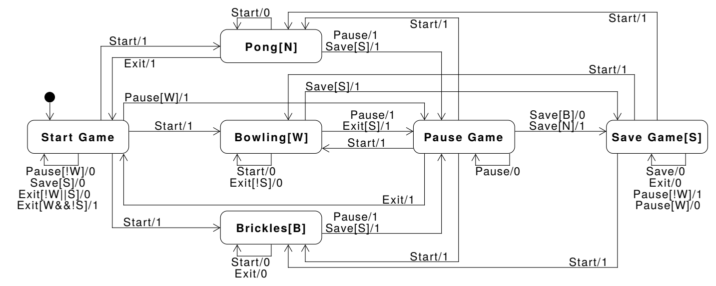

Figure: The AGM FFSM with separated states

RQ3) Is the $FFSM_{Diff}$ algorithm effective in **learning succinct family models** compared to **hand-crafted family models**?

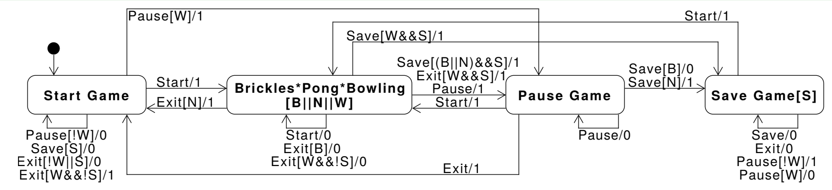

Figure: Alternative FFSM learnt from the AGM SPL presented fewer states [a]

[a]Fragal (2017)

Final remarks

Final remarks

- The creation and maintenance of family models is chllenging
- $FFSM_{Diff}$ is an effective algorithm
  - Learning fresh FFSMs from products pairs
  - Including an FSMs into an existing FFSM model
  - Especially if there is high feature reuse
- Our work complements reverse engineering techniques and SPL analysis

Final remarks

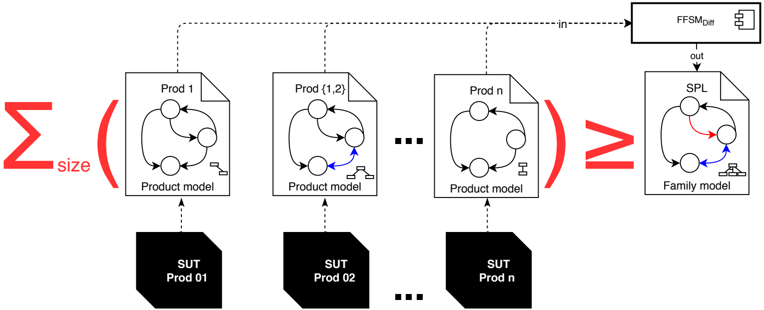

Figure: The $FFSM_{Diff}$ algorithm is available online at `https://damascenodiego.github.io/learningFFSM/`

Future work

Configuration simplification techniques

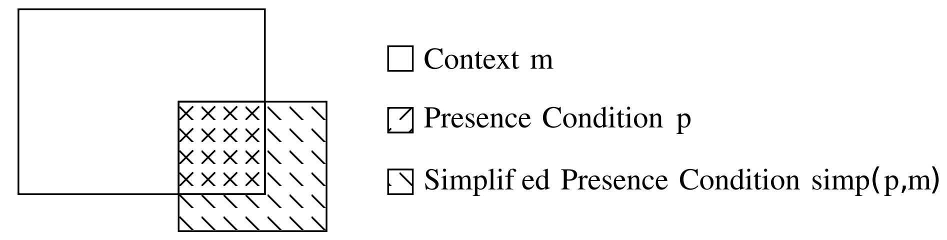

Figure: Configuration simplification techniques [a]

[a] von Rhein et al. (2015)

Future work

Configuration selection/prioritization

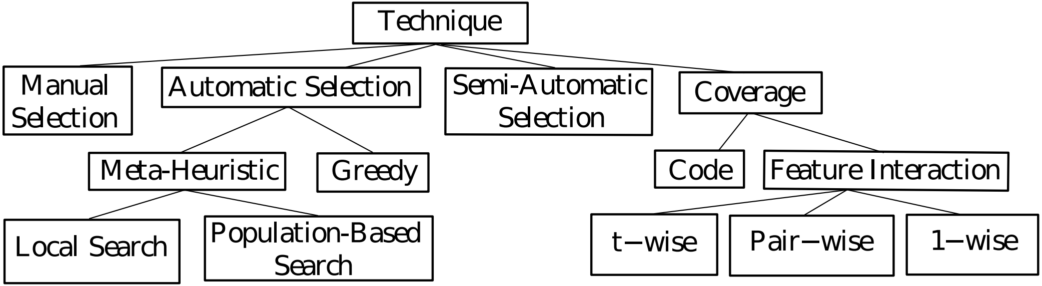

Figure: Techniques for configuration selection [a]

[a] Varshosaz et al. (2018)

Future work

Active learning of family models

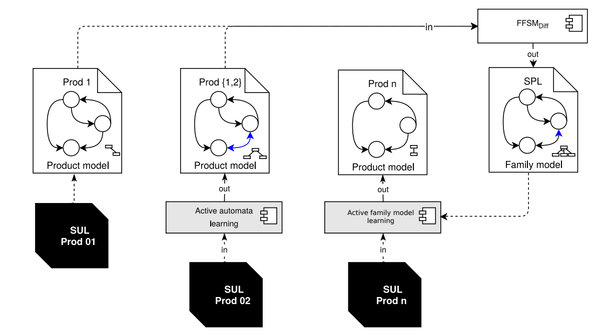

Figure: Active learning of FFSMs [a]

[a] Vaandrager (2017)

Future work

Learning hierarchical FFSMs

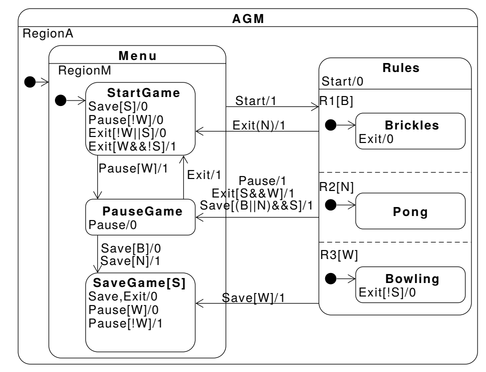

Figure: Hierarchical FFSM [a]

[a] Fragal et al. (2019)

Thank you!

https://damascenodiego.github.io/learningFFSM/

References I

Classen, A. (2010). Modelling with fts: a collection of illustrative examples.

Fragal, V. H. (2017). *Automatic generation of configurable test-suites for software product lines*. PhD thesis, Universidade de São Paulo. [Online] `http://www.teses.usp.br/teses/disponiveis/55/55134/tde-10012019-085746/`.

Fragal, V. H., Simao, A., and Mousavi, M. (2019). Hierarchical featured state machines. *Science of Computer Programming*, 171:67–88.

Fragal, V. H., Simao, A., and Mousavi, M. R. (2017). *Validated Test Models for Software Product Lines: Featured Finite State Machines*, pages 210–227. Springer International Publishing, Cham.

ter Beek, M. H., de Vink, E. P., and Willemse, T. A. C. (2017). *Family-Based Model Checking with mCRL2*, pages 387–405. Springer, Berlin, Heidelberg.

Vaandrager, F. (2017). Model learning. *Communications of the ACM*, 60(2):86–95.

Varshosaz, M., Al-Hajjaji, M., Thüm, T., Runge, T., Mousavi, M. R., and Schaefer, I. (2018). A classification of product sampling for software product lines. In *Proceedings of the 22Nd International Systems and Software Product Line Conference - Volume 1*, SPLC '18, pages 1–13, New York, NY, USA. ACM.

References II

von Rhein, A., Grebhahn, A., Apel, S., Siegmund, N., Beyer, D., and Berger, T. (2015). Presence-condition simplification in highly configurable systems. In *Proceedings of the 37th International Conference on Software Engineering - Volume 1*, ICSE '15, pages 178–188, Piscataway, NJ, USA. IEEE.

Walkinshaw, N. and Bogdanov, K. (2013). Automated comparison of state-based software models in terms of their language and structure. *ACM Transactions on Software Engineering and Methodology*, 22(2):1–37.

Questions?

https://damascenodiego.github.io/learningFFSM/