Learning Product Automata

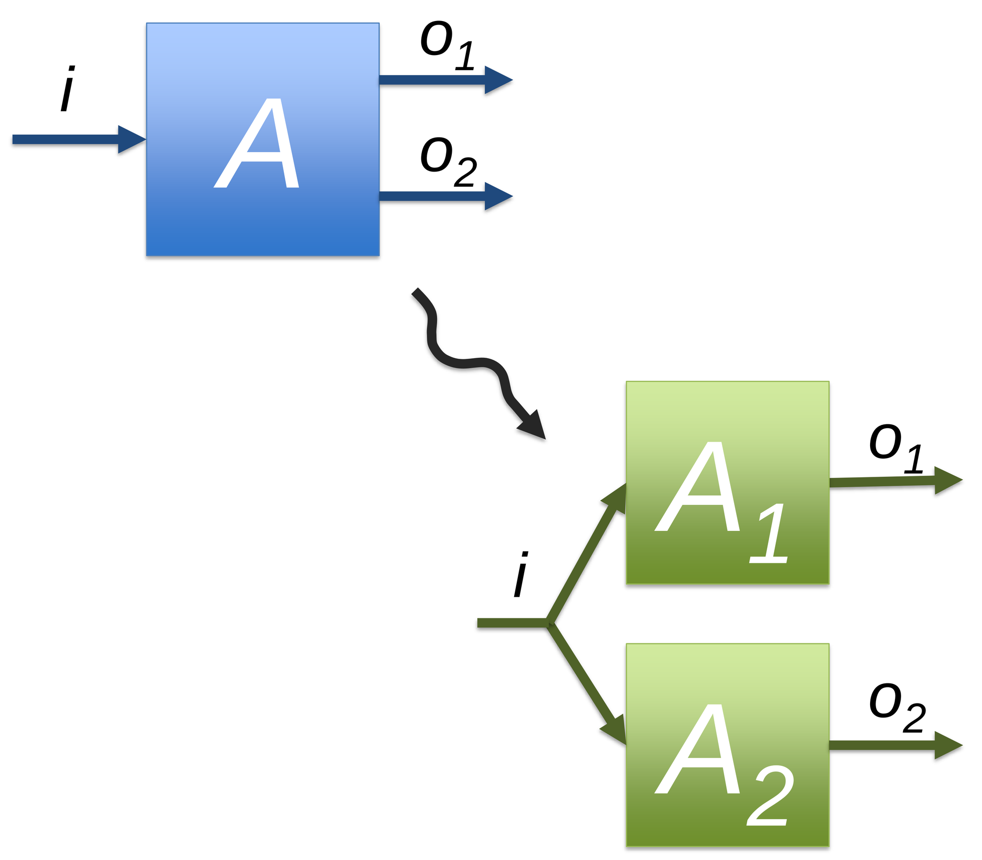
Joshua Moerman   moerman@science.ru.nl

The problem

In *model learning* one big problem is *scalability*. Big systems need many queries in order to be learnt. So we have to look for *structure* which we can exploit. In this paper we look at *parallel composition*.

Decomposition and learning

Mealy machines can be decomposed if they have multiple observables. Having smaller subcomponents is beneficial for many algorithms (divide & conquer).



The teacher answers *output queries* and *equivalence queries* for the machine *A*. The learning algorithm will infer the components A_1 and A_2 simultaneously.

Two approaches:
- Direct L* extension, or
- Reducing to existing learning algorithms (below)

Advantages vs. learning *A*
- More efficient to learn smaller automata.

Advantages vs. learning A_1 and A_2 independently:
- Sharing queries and counterexamples.

Convergence

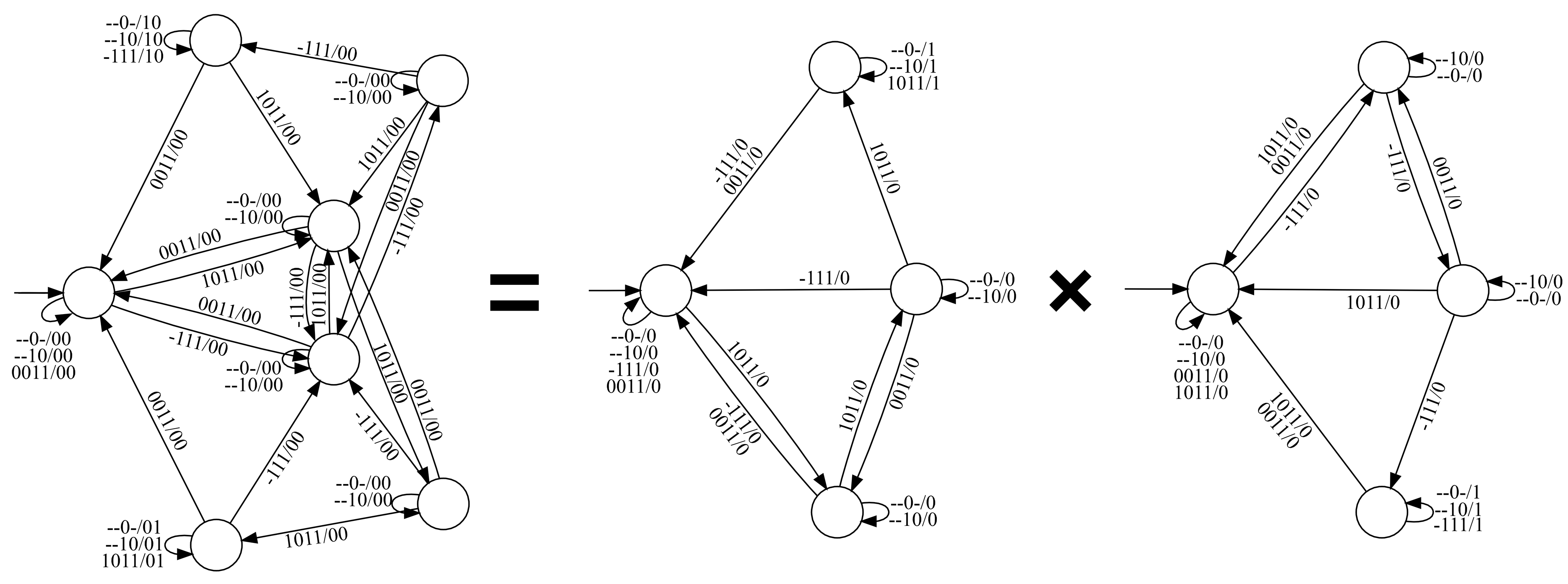
Number of states increases monotonically *per component*. Hence convergence is guaranteed.

Nevertheless, number of states for the intermediate hypotheses may actually exceed that of the target!

Algorithm

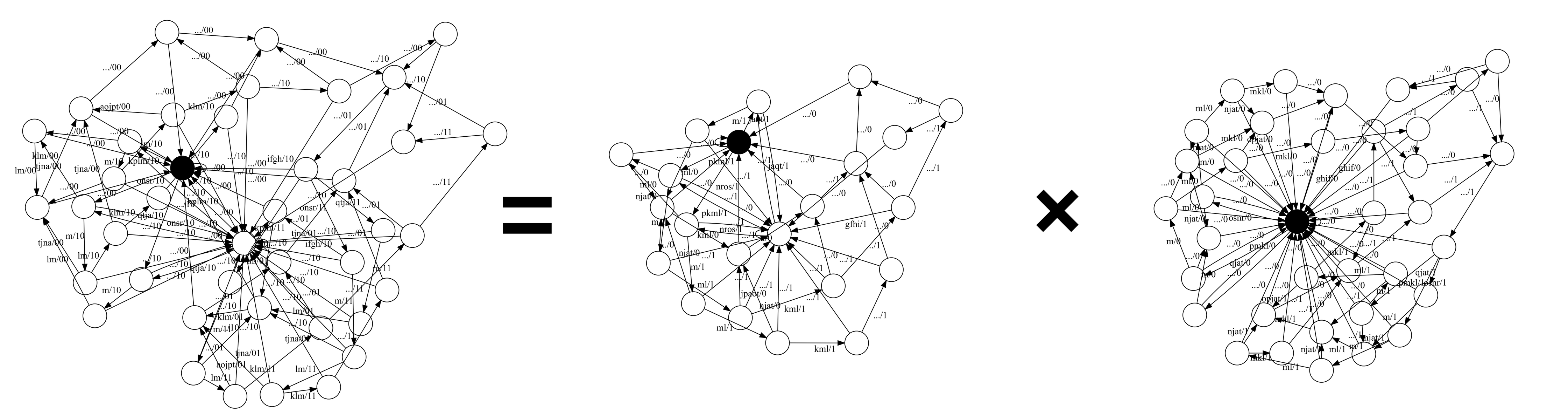
1: Initialise two learners L_1 and L_2
2: **repeat**
3: **while** L_i queries $MQ(w)$ **do**
4: forward $MQ(w)$ to the teacher and get output o
5: return $\pi_i o$ to L_i
{at this point both learners constructed a hypothesis}
6: Let H_i be the hypothesis of L_i
7: Construct $H = H_1 \times H_2$
8: **if** $EQ(H)$ returns a counterexample w **then**
9: **if** $[\![H_1]\!](w) \neq \pi_1[\![M]\!](w)$ **then**
10: return w to L_1
11: **if** $[\![H_2]\!](w) \neq \pi_2[\![M]\!](w)$ **then**
12: return w to L_2
13: **until** $EQ(H) = \texttt{YES}$
14: return $\texttt{YES}$ to both learners
15: **return** H

Example: bbara



7 states = 4 states × 4 states

Example: keyb



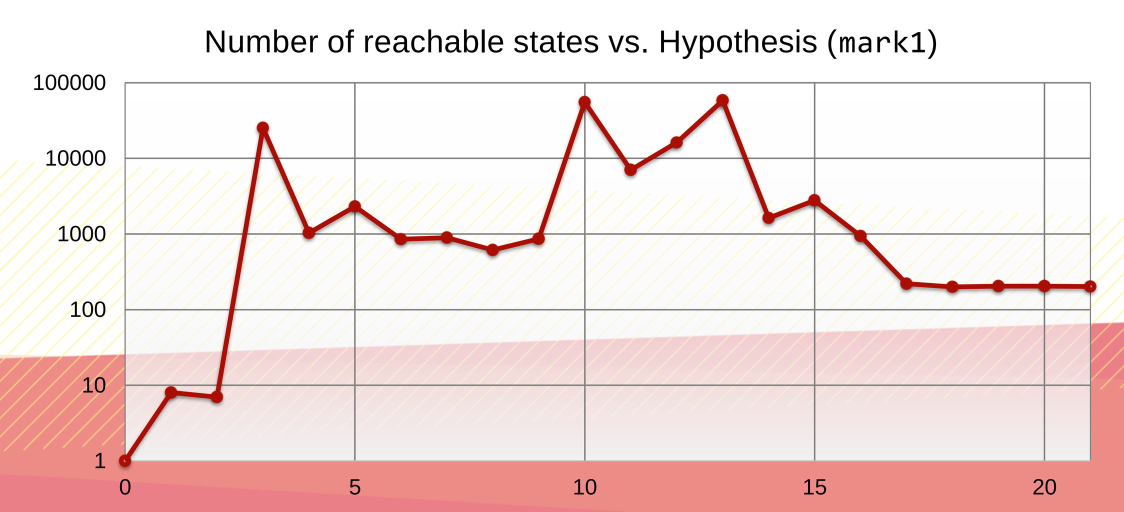
41 states = 22 states × 36 states

Results

The product learner is implemented in LearnLib. We compare it to the efficient TTT algorithm. We measure:
- *EQs*: number of equivalence queries posed
- *MQs*: number of membership queries posed
- *Actions*: total number of actions performed on the system (this includes testing for equivalence)

On all examples the product learner is *more efficient* in terms of actions.

				Product Learner			TTT Learner		
	Machine	States	Components	EQs	MQs	Actions	EQs	MQs	Actions
Artificial example	M_4	64	4	8	**456**	**3 025**	**6**	1 058	13 824
	M_5	160	5	**6**	**869**	**7 665**	17	2 723	34 657
	M_6	384	6	**11**	**1 383**	**12 870**	25	6 250	90 370
	M_7	896	7	**11**	**2 087**	**24 156**	52	14 627	226 114
	M_8	2048	8	**13**	**3 289**	**41 732**	160	34 024	651 678
Real circuit specifications	`bbara`	7	2	3	**167**	**1 049**	3	216	1 535
	`mark1`	202	8	**22**	**13 027**	**117 735**	67	15 192	252 874
	`keyb`	41	2	25	12 464	**153 809**	**24**	**6024**	265 805
	`ex3`	28	2	24	1 133	**9 042**	**18**	**878**	91 494



In this example, the number of states in the product hypothesis actually grew beyond 202 states during the learning process. Nonetheless, the product learner was more efficient than TTT.

Paper: https://arxiv.org/abs/1705.02850

Code: https://gitlab.science.ru.nl/moerman/learning-product-automata

Radboud University