Modular finite-state machine analysis

Simon Ware

Supervisor: Robi Malik

October 15, 2007

Abstract

Physical systems can be modelled as a system of finite state automata running in parallel. When a system is represented in this way it can be verified whether or not the given system conforms to given specifications by composing the synchronous product of the system[17]. However composing the synchronous product requires time and memory which is exponential in the number of automata in the system thus making it impractical for more complicated system. One of the methods for getting around this is the modular technique of controllability checking[3], but unfortunately this is also subject to in the worst case requiring time and memory which is exponential in the number of automata in the system. This project has developed new techniques for checking the controllability of systems which have even made it possible to check the controllability of a system which had previously been unsolved.

Contents

Chapter 1

Introduction

In industry it is frequently required to be able to verify that within a system such as a factory, certain states are unreachable (for example for the system of an elevator it might be useful to verify that it is impossible for the doors on the first floor to open when the elevator is currently on a different floor). These states can be referred to as "Bad States". Reasons for confirming that these "Bad States" are unreachable can range from loss of money to possible loss of life. Physical systems can be modelled as a set of Finite State Automata running in parallel. Once modelled in this form whether or not it is possible for the system to reach any "Bad State" can be checked by generating the synchronous product of the entire set of automata [17]. Unfortunately, this has the drawback that generating the synchronous product of a system takes time proportional to the number of reachable states in the composed system, which in turn grows exponentially with the number of automata in the system. For more complicated models this can be more states than could be realistically processed. Several methods to combat this have been developed. All of these methods seem to be superior for certain systems but inferior for others.

One of these is the modular method [3]. This method has shown great promise when verifying certain extremely large models. It is for this reason that we chose to investigate this method in this project. In this project we have managed to implement the modular method of controllability checking as well to design and implement several variants of this approach. In addition to this, we have designed a method of automatically simplifying models such that they may have orders of magnitude less reachable states. Yet it can be provably shown that there will exist at least one "Bad State" in this simplified model, if and only if there exists a "Bad State" in the original model. To do this we designed and implemented a method of transforming the original model into a new model which is equivalent in terms of the behaviour we are verifying [7], as well as a method for continuously using projection [17, 9] to simplify this model. Also if a "Bad State" is found in the simplified model we have also designed and implemented a method for translating the cause of a "Bad State" in the simplified model to its cause in the original model. This method has been

developed to be used in conjunction with the modular method. It has however shown promise when used independently of it. All of these checkers have been integrated into the WATERS toolkit and have been tested on a large set of models of varying sizes, some of them extremely large.

This thesis is be composed of the following chapters.

In Chapter 2 we discuss some of the underlying theory required to understand this thesis. In Chapter 3 we discuss the modular checker implemented in this paper and how it compares original implemented in Valid [3]. In Chapter 4 we discuss the a variant of the Modular Checker called the Parallel Checker and how well it works. In Chapter 5 we discuss another variant of the Modular Checker called the Culling Checker and how well it works. In Chapter 6 we discuss various algorithms for using projection in model checker. In Chapter 7 we discusses related work.

Chapter 2

Preliminaries

To make the contents of the rest of this report more readily understandable, it is useful to give definitions for some of the more important terms.

2.1 Languages

Languages are made up of an alphabet and words. The alphabet of a language is the set of symbols which can possibly occur within that language's words. A word is a sequence of symbols. A language contains a set of words.

When expressing a discrete event system as a language the alphabet of the language is the set of all events which can possibly occur in the system. The set of words of the language is the set of all possible sequences of events which could possibly occur in that system. A sequence of events can also be referred to as a trace through the system.

2.2 Automata

Figure 2.1 shows an example of an automaton. This example consists of circles called states and arrows between states called transitions. Each state has a label associated with it representing its state name. Likewise each transition has a label associated with it representing the event required to travel along that edge. Transitions can be written as $p \xrightarrow{\alpha} q$ where p is the source state, α is the event, and q is the target state. In addition, we can see that the state S0 has an arrow pointing to it from nowhere. This marks it as the initial state of the automaton.

Now that it has been explained what makes up an automaton it can be explained how an automaton can be used to represent a language. An automaton will accept a trace $\sigma_1 \dots \sigma_n$ if and only if there exists a sequence of transitions $p_1 \xrightarrow{\alpha_1} q_1, \dots, p_n \xrightarrow{\alpha_n} q_n$ through the automaton which fulfils the following requirements.

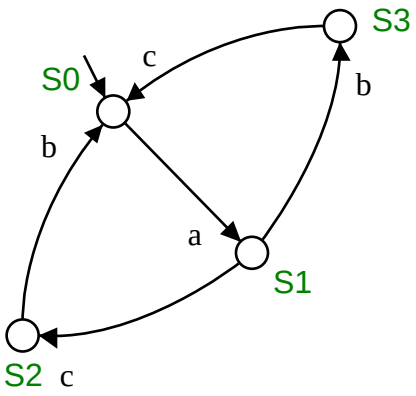

Figure 2.1: An example automaton

1. For each transition the event labelled on it matches the corresponding event in the trace.
$$\forall i \in 1..n; \sigma_i = \alpha_i$$

2. The source state of each transition matches the target state of the previous transition.
$$\forall i \in 1..n-1; q_i = p_{i+1}$$

3. In the case of the first transition, its source state must be in the set of initial states.

From this we can define the language $L(A)$ of the automaton A to be equal to the set all the event sequences which the automaton accepts. Here are two example traces for the automaton given in figure 2.1.

1. For the trace *abcacba* the automaton would have the possible sequence of transitions
$$S0 \xrightarrow{a} S1 \xrightarrow{b} S3 \xrightarrow{c} S0 \xrightarrow{a} S1 \xrightarrow{c} S2 \xrightarrow{b} S0 \xrightarrow{a} S1$$

2. Whereas for the sequence *abbcacba* would only have a trace up to
$$S0 \xrightarrow{a} S1 \xrightarrow{b} S3$$
and the automaton would reject the trace.

For the automaton given, none of the states have been marked as accepting states, because for the purpose of this report it is good enough to consider all states as accepting.

2.3 Running in Parallel

When modelling discrete event systems they are generally represented as a set of automata running in parallel. This is because designing just one automaton representing the entirety of a system is impractical due to the complexity of most system's.

When automata run in parallel they essentially all run at the same time. This means that the system can only receive those events which are allowed to occur in the current state of each automaton and that the state of each automaton is updated concurrently. It should also be noted that for every automaton, there is implicitly a selfloop for every event not contained in the alphabet of that automaton on every state in the automaton. Selfloops are transitions for which both the source and target state are the same.

If we have a set of automata running in parallel we can build an automaton which represents the language of the entire system running in parallel. This is called the synchronous product of the system.

Figure 2.2 is an example of a factory with two machines [17] called small factory. Both machines start in the Idle state (I) and can enter a Running state (R) by receiving a start event (s1 or s2). Then when they are Running they can either receive a finish event (f1 or f2) meaning that they have finished doing their job and return to the Idle state (I), or they can receive a break event (b1,b2) which causes the automaton to break down and enter the Broken state (B), in which they can be repaired(r1,r2) and return to the Idle state (I). In addition to this there is a buffer between the two machines. This buffer starts out as being Empty (E) but when Machine1 finishes (f1) it takes the product from Machine1 and becomes Full (F). Then when Full (F), if Machine2 starts (s1), it takes the product and works on it making the buffer Empty (E) again.

Figure 2.3 is the synchronous product of Figure 2.2. When looking at the state labels of this automaton we can see that they contain three names separated by commas. The significance of these three names is that the first name corresponds to the state of machine1 the second state corresponds to the state of the buffer and the third name corresponds to the state of machine2. Thus, if this automaton is in the state R,F,I then machine1 is Running (R), Machine2 is Idle (I) and the buffer is Full (F). From the diagram, it is possible to see that machine1 cannot finish when the buffer is Full (F) and machine2 cannot start when the buffer is Empty (E). Also when looking at figure 2.3, we can see that even a simple system such as small factory produces a rather complex and messy synchronous product.

2.4 Language Inclusion

Language inclusion can be used to check to see if the language of a system is more restrictive than the language of a given automaton. Such an automaton is called a property or requirement. We say that a set of automata A satisfies a requirement R if and only if every trace t which is in the language of A is also

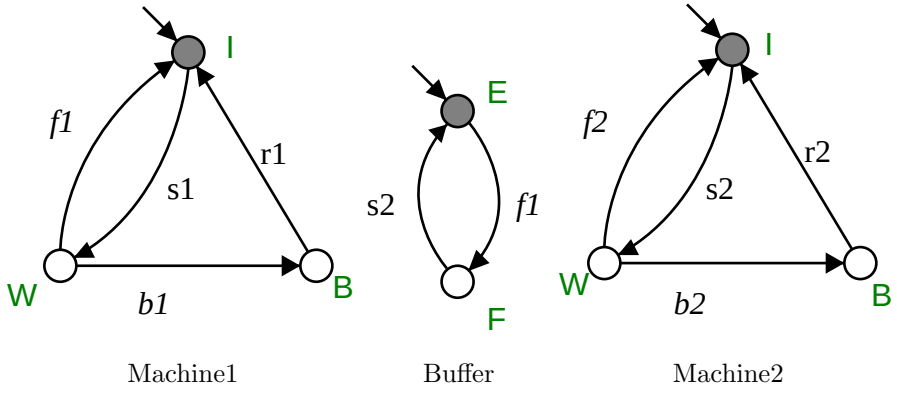

Figure 2.2: Small factory

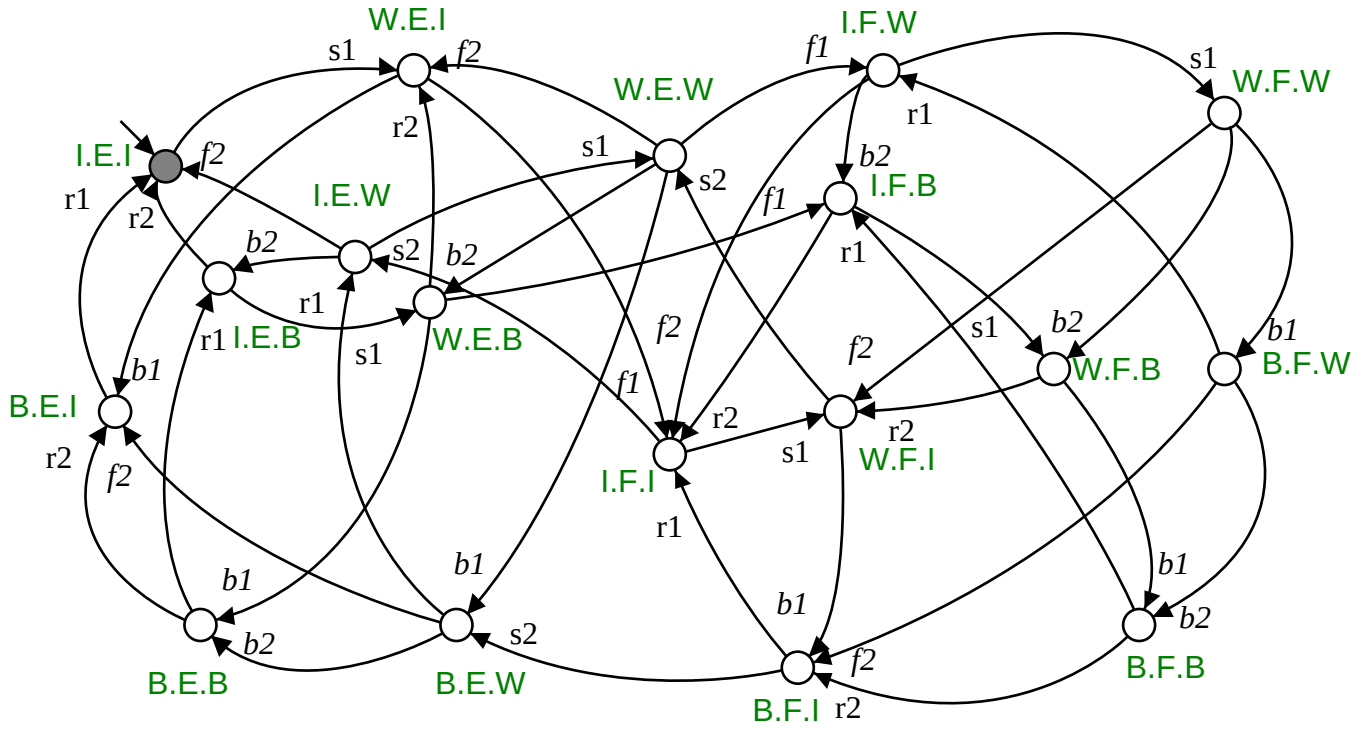

Figure 2.3: Synchronous product of small factory

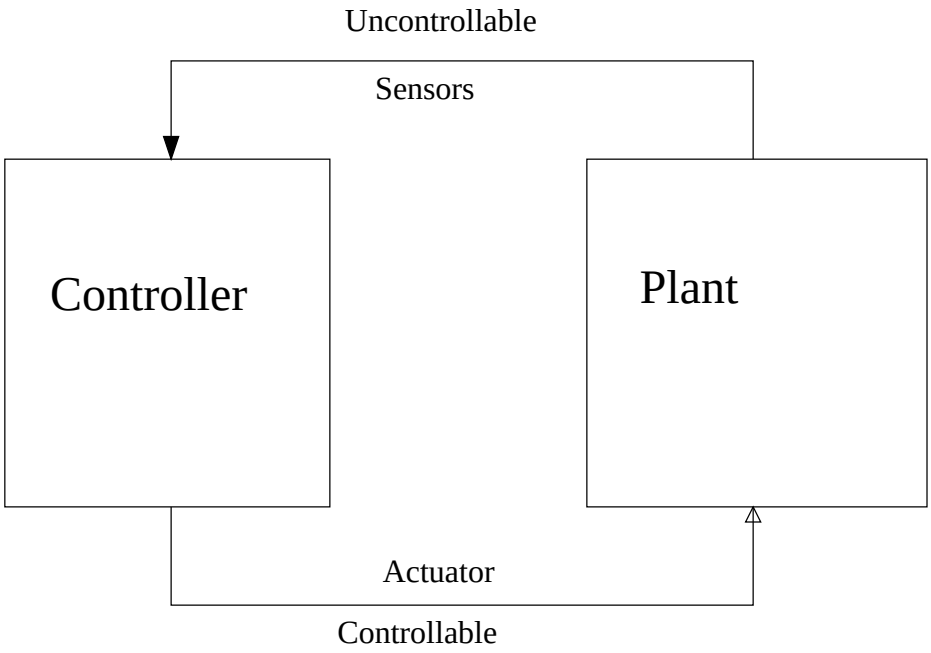

Figure 2.4: Figure of Controllability

in the language of R [1].

$$\forall t : t \in L(A) \rightarrow t \in L(R)$$

The standard algorithm for checking if a set of automata A satisfies the requirement R is to construct the synchronous product A and R and then to check to see that for all states in the synchronous product that whenever all the automata in A allows a given event to occur R will also allow that event to occur.

This is useful because it can often be easier to define aspects of high-level system behaviour in a property without having to talk about low-level aspects of the system, and then check to see that the actual system which has to deal with the low-level details still conforms to this.

2.5 Controllability Checking

For controllability a system can be divided up into a controller and a plant. The controller is capable of sending controllable events to the plant telling the plant what to do, whereas the plant is capable of sending uncontrollable events back to the controller telling the controller what has happened to it. This relationship is shown in figure 2.4.

Automata are defined as being either plants which are part of the Plant, or controllers which are part of the Controller.

The language of the Plant is the intersection of the languages of all the plants in the system and represents all the possible sequences of events the Plant could go through.

The language of the Controller is likewise the intersection of the languages of all the controllers in the system and represents all the sequences of events the Controller would allow the system to go through.

From this we define a Plant P as being controllable with respect to a Controller C, if and only if, there exists no trace t for which, if it had an uncontrollable event υ appended to it's end, it would be accepted by the language of the plant but not the controller.

$$\forall t, \upsilon : t\upsilon \in L(P) \wedge t \in L(C) \rightarrow t\upsilon \in L(C)$$

The standard algorithm for checking for controllability in a system is similar to that of checking language inclusion. We simply to construct the synchronous product of the system and check that in every state of the system that whenever all the plants in P allows a given uncontrollable event to occur all the automata in C also allow that uncontrollable event to occur [17]. This approach to controllability is called the **Monolithic** approach.

Now we can revisit the example automaton given in Figure 2.2 where we now state that machine1 and machine2 are both plants as they represents how the system behaves. Buffer however is a specification as it has no control over when machine1 and machine2 stop. We further go on to state that starting and repairing a machine are both controllable, whereas a machine finishing or breaking is uncontrollable. A machine finishing is said to be uncontrollable as once the process starts there is no way to delay the process finishing. The item produced in the process must be removed from the machine as soon as it finishes lest either the machine or the object be damaged. Now if we look at the synchronous product of this system given in Figure 2.3 we can see that machine1 and machine2 are not in fact controllable with respect to buffer, as in the state Running,Full,Idle, the buffer would not allow the uncontrollable event of machine1 finishing (f1) to occur whereas both machine1 and machine2 would.

The small factory example can be made controllable by replacing its buffer controller by the one given in Figure 2.5 as can be seen by the fact that there is no state in the synchronous product of this new system given in figure 2.6, where machine1 is never allowed to finish when the buffer would not allow it to.

It should also be noted that the Language Inclusion problem referred to in the previous section can be converted into a Controllability problem simply by stating that all specifications are plants, all properties are specifications, and all events are uncontrollable [1].

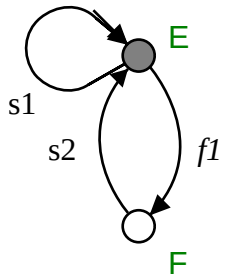

Figure 2.5: Modified buffer

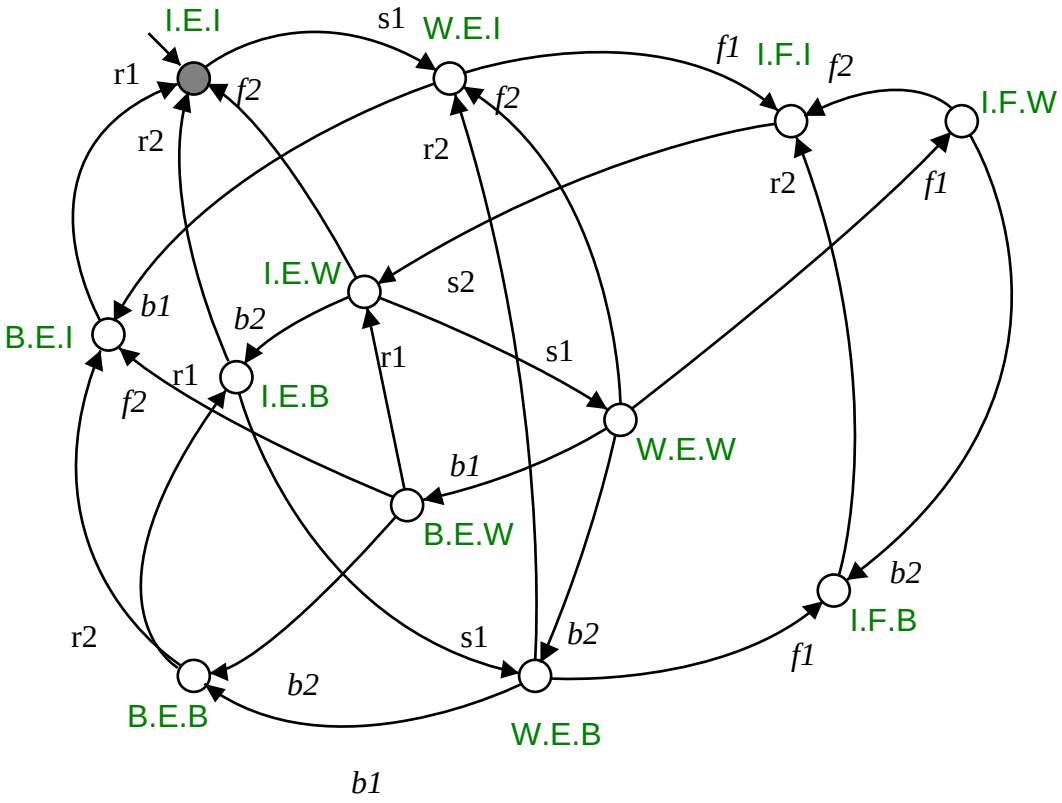

Figure 2.6: Synchronous product of Small factory with modified buffer

Chapter 3

Modular Checker

The monolithic method of checking controllability in a system has the major drawback that the construction of the synchronous product of a system takes time which is exponentially proportional to the number of automata in the system, this makes it unfeasible to use on larger systems.

The method of modular controllability checking [3] attempts to solve the problem of verifying controllability, in less time. It does this by exploiting the fact that the synchronous product of a system is the intersection of all languages in the system. Thus if a subset of the automata in the system do not contain a trace which is a counterexample for a given controller, then the entire system must not contain a counterexample for the given controller, as adding extra automata to the synchronous product will not add any new traces to the synchronous product. Therefore, it is possible in many instances to prove a specification without having to compose the entire system of automata.

3.1 Algorithm

The modular checker described in [3] was implemented in the WATERS framework. The algorithm is detailed in Figure 3.1. We pick a controller to prove controllable with respect to the plant. Then we use a controllability checker to find a counterexample for the controller. Then we pick a plant or controller which doesn't permit said counterexample, and add it to the composition with the original controller, then repeat the process by checking the controllability of the system with the extra automaton. This process course end under one of two conditions first, we may come to a situation where we find a counterexample which no plants or controllers in the system are capable of rejecting, in which case the counterexample must be a counterexample for the model as a whole, and as such the model is not controllable. Alternatively, we may come to a point where the controllability checker finds no counterexample for the composition, in which case we know that there is in fact no counterexample for the controllers currently in the composition [3], and we can now treat them as

C equals the set of Controller automata in the model and P the set of Plant automata, and S is the set of automata composed so far.

1. Set the set S as being empty.
2. If C is empty the model has been proven controllable; otherwise take an automaton from the set C and add it to S.
3. Check controllability of S using the monolithic method. Consider automata which are elements of C as controllers and automata which are elements of P as plants.
4. If no counterexample for S was found go to 7. Otherwise set t to be the counterexample found by the controllability check.
5. Set the set N to contain all automata in P and C which do not accept the counterexample t. Take into consideration for all automata in C that specifications in addition to not accepting t, must also not consider t as being a counterexample to their controller.
6. If N is empty, then the model has been proven not controllable and t represents a counterexample in the system. Otherwise, pick a subset of N to add to S, then go to 3.
7. For all elements of S, if they are also an element of C, remove them from C and add them to P. Then go to 1.

Figure 3.1: Modular Controllability checking algorithm

plants (as we have proven that under no circumstances can an uncontrollable event occur when the controller would not allow it). At this point we go on to prove any controllers which have yet to be proven in the same way as above. Once all controllers have been proven, we can say that the model is controllable.

If we look at the algorithm for modular controllability checking in Figure 3.1, it can be noticed that for both steps 1 and 6, it has not been adequately specified just exactly which automata to pick at each of these steps. This is because for both of these steps there is more than one way of choosing these automata, and which one is best can be different for any given model. Firstly for step 1 there are two ways of selecting which specification to prove. The first is to use the comparator described in Figure 3.2 to find the smallest specification in S and to use that one, The rationale is that it will most likely be easier to prove the smallest specifications first, and once proved they can be treated as plant automata, thus helping to prove all subsequent specifications. The second is to compare all the specifications in exactly the same manner as above, but instead of taking the smallest to take the largest, hoping that either largest is either more likely to have a counterexample, or that, when proven, it will be more helpful as a plant for the purpose of proving subsequent specifications.

Then for step 6, there are many heuristics which can be used to choose an automaton to add to the composition. The heuristics are as follows.

All add all automata in N into C.

EarlyNotAccept Add the automaton in N which rejected t the after the fewest number of steps through t.

LateNotAccept Add the automaton in N which rejected t the after the greatest number of steps through t.

MaxCommonEvents Add the automaton in N which has the maximum number of events in common with C.

MaxCommonUncontrollable Add the automaton in N which has the maximum number of Uncontrollable events in common with C.

MaxStates Add the automaton in N which has the largest number of states.

MinEvents Add the automaton in N which has the smallest number of events.

MinNewEvents Add the automaton in N which has the smallest number of events which are not currently contained in C.

MinStates Add the automaton in N which has the smallest number of states.

MinTransitions Add the automaton in N which has the smallest number of Transitions.

One Arbitrarily take the first automaton found in N.

RelMaxCommonEvents Add the automaton in N which has the highest proportion of its events in contained in C.

In addition each of these heuristics can be run in two modes. They can either consider all automata as being equal or they can consider plant automata as being superior to controller automata thus always choosing a plant automata over a controller automata if at all possible. The reasoning behind this is that whenever we add in an extra controller automaton to the composition we also add in new possible counterexamples which must be ruled out.

3.2 Results

This section contains tables of results for both the original modular controllability checker implemented in Valid [3], as well as the results for the checker implemented in WATERS. The results from Valid show the total number of states which had to be explored, whereas for the new implementation the number of seconds required to solve the model is also shown. For all cases, when the model checker was run it was set up so that whenever the model checker attempted to construct the synchronous product of a set of automata and the

The first automaton is A_1 and the second automata A_2.

1. If A_1 has more states than A_2, then A_1 is bigger.
2. If both A_1 and A_2 have the same number of states then, if A_1 has more transitions than A_2, A_1 is bigger.
3. If both A_1 and A_2 have the same number of transitions then, if A_1 has more events than A_2, A_1 is bigger.
4. If both A_1 and A_2 have the same number of events then, if A_1 has a bigger name than A_2, A_1 is bigger.

Figure 3.2: Comparator for automata

number of states which it has explored in that synchronous product becomes greater than two million states, then the model checker stops to prevent itself from running out of memory, in which case there will be a blank entry in the table where the number of states should be.

Here follows a description of the models used for testing.

- big_cmft_kl50, big_fh_cmftreq1, big_manual_cmft, bigcmft_reg, big_fh_cmftreq0, big_bmw are models describing the BMW E65 CAS window lift controller [6, 13].
- fzelle, ftechnik, ftechnik-nocoll, represent a case study of a production cell [11, 12].
- models beginning with profisafe represent the PROFIsafe field bus protocol [14, 15, 16].
- rhone_alps, rhone_tough, represent an AIP automated manufacturing system [2, 4, 10].
- tbed_uncont, tbed_nocoll, tbed_noderail, tbed_ctct, tbed_valid, represent a train testbed [10].
- verriegel4_vrprop, verriegel4_erprop, verriegel4, represent a central locking system.

All of these models represent real world systems and have state spaces which are to large to be explored by a monolithic controllability checker.

It can be seen between the two sets of tables, there can be in some cases quite marked differences in results between the two implementations. This can most likely be put down to the fact that both implementations use different methods of choosing which specification to prove first, both have different methods to break ties when a heuristic considers two automata equally desirable, and it is quite possible that in some cases the specific counterexample found at each iteration by either method could be different.

If we look at the differences in states required for any particular heuristic to solve a problem it can be seen that, for most heuristics, their performance is in fact comparable with the one heuristic. This is to say the performance of most heuristics is in fact comparable to just arbitrarily picking an automaton to add to the composition. The notable exceptions to this are MaxCommonEvents, MaxCommonUncontrollable, and RelMaxCommonEvents which have the capability of solving the controllable models in the tbed series of problems when having no preference for plants. From the test data it looks like MaxCommonEvents is the best bet for solving most problems as it shows the most consistency in requiring to look through a low number of states. Also when we look at the difference in performance in heuristics when comparing a preference for plants to no preference, we can see that for most models a preference for plants seems to give mildly better results, whereas for the tbed series of models no preference works a lot better. Finally, when we look at the difference between checking the larger specifications first or the smaller, it seems that with the exception of big_bmw for most models the strategy of solving the largest automata first is the better choice.

In addition one of the test cases used to benchmark the original algorithm were transferlines of varying sizes. The transferline model of an arbitrary number of functional blocks which can be combined into a large system with a regular structure [18].

The original algorithm was shown to be capable of solving the transferline model by exploring a number states which was linearly proportional to the number of blocks in the model. Thus as a test we also ran the modular checker implemented in this project using the MaxCommonEvents heuristic, on transferlines with numbers of functional blocks between 1 and 230 and plotted the number of states explored against the number of functional blocks in the transferline. The chart of this can be seen in Figure 3.3 and clearly shows a linear relationship between states explored and the size of the transferline.

Table 3.1
ORIGINAL MODULAR LANGUAGE INCLUSION CHECK

Model		Modular Language Inclusion									
		All	Early NotAccept	Late NotAccept	MaxCommon Events	Min Events	Min NewEvents	Min States	Min Transitions	One	RelMax Common
Name	Aut	States	States	States	States	States	States	States	States	States	States
big_cmft_kl50	32	753	118	77	1669	3868	1309	53	118	3305	432
big_fh_cmftreq1	32	23	7	7	7	52	2684	7	7	52	52
big_manual_cmft	32	765	6323	40	60	438	8789	13	40	1178	438
ftechnik_nocoll	42	271242	4309	2666552							
profisafe_i4_host_to	76	419	13571	2409	277605	77038	41148				
profisafe_i4_slave	76	24612	475262	4450	11496	36377	274723	64953	454711	12635	
profisafe_o4_host_to	85	436	13575	2454	277160	77044	40899				
profisafe_o4_slave	85	2907	1690	10611	1079247	17662					
tbed_nocoll	109	5144173	335791								
tbed_noderail	96	112	3287	8395939	623	4273	1245				
verriegel4_vrprop	66	34326	23709	23709	22614	23709	23709	11000	23709	11000	23709
big_cmft_req	32	1465	1548	1572	1548	1572	1385	1572	1572	1548	1572
big_fh_cmftreq0	32	2296	3584	5809	3851	3851	6261	6276	3894	4715	3851
verriegel4_erprop	66	538	759	759	759	1716	1716	759	759	1571	700

Table 3.2
ORIGINAL MODULAR CONTROLLABILITY CHECK PREFERRING PLANTS

Model		Modular controllability, not preferring plants									
		All	Early NotAccept	Late NotAccept	MaxCommon Events	MaxCommon Uncontr	Min NewEvents	Min States	Min Transitions	One	RelMax Common
Name	Aut	States	States	States	States	States	States	States	States	States	States
big_bmw	31	1096	190	190	411	223	1281	190	190	190	5219
fzelle	67	7793	18385	5366	4072	4072	7101	8990	8732	4901	3394
profisafe_i4	75	688	351	245	155	160	369	244	245	160	224
profisafe_o4	84	691	354	248	158	163	372	247	248	163	227
rhone_alps	35	224616	1035198	16021	224838	16432	16063	903	903	955	1037531
tbed_ctct	84	119934	29092	18522	18522	3557956					
tbed_valid	84	609040	3733989								
verriegel4	65	32027	655	23142	1730	19103	7859	7859	75704	2956	
ftechnik	36	159118	9879	221	221	221	6557213	6571455	4547036	683	
tbed_uncont	58	821906	310893	5772225	2158804	1536444	999512				

Table 3.3
ORIGINAL MODULAR CONTROLLABILITY CHECK NOT PREFERRING PLANTS

Model		Modular controllability, preferring plants									
		All	Early NotAccept	Late NotAccept	MaxCommon Events	MaxCommon Uncontr	Min NewEvents	Min States	Min Transitions	One	RelMax Common
Name		States	States	States	States	States	States	States	States	States	States
big_bmw	31	346	190	190	223	223	339	190	190	190	1110
fzelle	67	9711	9733	7799	6316	6826	7882	8990	8732	4901	5658
profisafe_i4	75	688	351	245	155	160	369	244	245	160	224
profisafe_o4	84	691	354	248	158	163	372	247	248	163	227
rhone_alps	35	224614	903	16021	8367	8333	16063	903	903	955	37030
tbed_ctct	84	119934	29092	18522	18522	3557956					
tbed_valid	84										
verriegel4	65	1227894	31666	7859	9454	8956	42931	7859	7859	75704	16605
ftechnik	36	1089179	2376834	6568826	3428725	3429005	6783622	6557213	6571455	4547036	5854300
tbed_uncont	58	281589	6517643	1536444							

Table 3.4
MODULAR LANGUAGE INCLUSION CHECK

Model		Modular Language Inclusion											
		All		Early NotAccept		Late NotAccept		MaxCommon Events		Max States		Min Events	
Name	Aut	States	Time	States	Time	States	Time	States	Time	States	Time	States	Time
profisafe_i4_slave	15	13179	0.15	927	0.16	16969	0.32	11515	0.24	16958	0.31	1238	0.18
profisafe_o4_slave	17	10426	0.19	668	0.2	18806	0.5	16428	0.37	12414	0.43	8214	0.36
big_bmw	32	1465	0.03	1548	0.04	1874	0.06	1548	0.06	1548	0.06	1572	0.06
ftechnik	37	71973	3.95		12.51		20.25		42.19		16.6		20.2
tbed_nocoll	85		33.28		38.97		38.96	3639	0.94	2835	14.14		32.4
tbed_noderail	85		14.38		57.66		32.52		27.15		56.7		24.27
verriegel4	66	580	0.02	805	0.04	1680	0.05	805	0.03	1680	0.05	1680	0.05
profisafe_i4_host	29	2155	0.18	2758	0.22	1418	0.21	2491	0.22	1671	0.22	1532	0.19
profisafe_o4_host	31	2155	0.18	2758	0.22	1418	0.21	2491	0.23	1671	0.22	1532	0.19
profisafe_i5_host	29	2462	0.22	3294	0.26	1616	0.25	2893	0.26	1869	0.25	1823	0.22
profisafe_o5_host	31	2462	0.21	3294	0.27	1616	0.25	2893	0.27	1869	0.25	1823	0.22
profisafe_i6_host	29	2769	0.24	3830	0.32	1814	0.28	3295	0.31	2067	0.3	2114	0.26
profisafe_o6_host	31	2769	0.25	3830	0.32	1814	0.28	3295	0.31	2067	0.29	2114	0.26

Model		Min NewEvents		Min States		Min Transitions		One		RelMax Common	
Name	Aut	States	Time	States	Time	States	Time	States	Time	States	Time
profisafe_i4_slave	15	18831	0.31	867	0.14	927	0.14	16969	0.31	17929	0.31
profisafe_o4_slave	17	25202	0.49	28063	0.47	14167	0.35	23820	0.47	6359	0.33
big_bmw	32	1572	0.03	1572	0.03	1572	0.04	1548	0.05	1572	0.03
ftechnik	37		20.62		65.95		52.52		10.59		23.92
tbed_nocoll	85		38.78		51.71		29.61		64.51		23.19
tbed_noderail	85		25.97		50.47		26.27		27.38		14.31
verriegel4	66	1680	0.05	805	0.03	805	0.03	1680	0.05	805	0.03
profisafe_i4_host	29	2223	0.24	2170	0.23	1532	0.19	1648	0.22	2223	0.23
profisafe_o4_host	31	2223	0.24	2170	0.23	1532	0.19	1648	0.22	2223	0.24
profisafe_i5_host	29	2530	0.28	2477	0.27	1514	0.21	1846	0.25	2530	0.28
profisafe_o5_host	31	2530	0.28	2477	0.27	1514	0.21	1846	0.25	2530	0.29
profisafe_i6_host	29	2837	0.32	2784	0.31	1712	0.24	2044	0.29	2837	0.32
profisafe_o6_host	31	2837	0.32	2784	0.31	1712	0.24	2044	0.29	2837	0.32

Table 3.5
MODULAR CONTROLLABILITY CHECK PREFERRING PLANTS, LARGEST CONTROLLER FIRST

Model		Modular controllability, preferring plants, largest controller first											
		All		Early NotAccept		Late NotAccept		MaxCommon Events		MaxCommon Uncontr		Max States	
Name	Aut	States	Time	States	Time	States	Time	States	Time	States	Time	States	Time
big_bmw	31	4931	0.11	5036	0.1	5447	0.09	5032	0.09	5013	0.08	5463	0.1
fzelle	67	7922	0.17	1421	0.15	1249	0.15	1545	0.16	2212	0.17	2122	0.15
rhone_alps	35	17319	0.1	10041	0.16	16327	0.11	842	0.04	10293	0.09	16755	0.11
tbed_ctct	84	119806	0.41	18391	0.12		34.07	18391	0.12	22298	0.13		26.68
tbed_nocoll	84		19.42		47.39		31.83		14.64		14.64		30.88
tbed_noderail	84		19.77		55.94		45		14.82		14.79		35.04
verriegel4	65	26428	0.27	30341	0.37	1310	0.12	2488	0.15	16250	0.23	33318	0.37
profisafe_i4	80	56	0.09	124	0.12	63	0.09	41	0.08	63	0.1	41	0.08
profisafe_i4_host	28	48	0.07	112	0.09	58	0.08	36	0.06	58	0.08	36	0.05
profisafe_i4_slave	14	8	0.02	12	0.04	5	0.03	5	0.03	5	0.02	5	0.03
profisafe_i5	88	56	0.1	124	0.14	63	0.11	41	0.1	63	0.11	41	0.09
profisafe_i5_host	28	48	0.07	112	0.1	58	0.09	36	0.06	58	0.09	36	0.07
profisafe_i6	94	56	0.12	124	0.16	63	0.14	41	0.1	63	0.14	41	0.11
profisafe_i6_host	28	48	0.09	112	0.12	58	0.1	36	0.07	58	0.1	36	0.07
profisafe_inclusion_i4host	78	184	0.08	143	0.11	52	0.07	66	0.07	276	0.13	130	0.09
profisafe_inclusion_o4host	84	184	0.09	143	0.11	52	0.08	66	0.08	276	0.13	130	0.09
profisafe_inclusion_o4slave	84	184	0.09	143	0.11	52	0.08	66	0.08	276	0.13	130	0.09
profisafe_o4	90	56	0.09	124	0.13	63	0.11	41	0.08	63	0.11	41	0.08
profisafe_o4_host	30	48	0.06	112	0.09	58	0.08	36	0.06	58	0.08	36	0.06
profisafe_o4_slave	16	8	0.03	12	0.04	5	0.04	5	0.03	5	0.03	5	0.03
profisafe_o5	99	56	0.11	124	0.15	63	0.13	41	0.1	63	0.13	41	0.1
profisafe_o5_host	30	48	0.08	112	0.11	58	0.09	36	0.06	58	0.09	36	0.06
profisafe_o6	106	56	0.13	124	0.18	63	0.14	41	0.12	63	0.15	41	0.12
profisafe_o6_host	30	48	0.09	112	0.12	58	0.1	36	0.08	58	0.1	36	0.07
ftechnik	36		45.01	177113	1.19		98.31	587588	2.87		30.92		101.97
rhone_tough	61		20.53		21.09		33.27		32.99		8.75		21.6
tbed_uncont	84		19.48		47.43		31.93		26.02		25.87		30.69

Model		Min Events		Min NewEvents		Min States		Min Transitions		One		RelMax Common	
Name	Aut	States	Time	States	Time	States	Time	States	Time	States	Time	States	Time
big_bmw	31	5317	0.08	5317	0.09	5613	0.08	5317	0.09	5056	0.09	5013	0.09
fzelle	67	1735	0.17	1653	0.16	3184	0.2	2959	0.19	2311	0.16	1625	0.16
rhone_alps	35	1158	0.05	872	0.05	1158	0.05	1158	0.05	1158	0.05	872	0.04
tbed_ctct	84		33.59		28	3563784	20.36		18.09		24.17	219497	0.82
tbed_nocoll	84		26.85		30.35		58.96		71.43		34.89		16.71
tbed_noderail	84		29.27		30.63		59.09		74.02		51.78		16.85
verriegel4	65	1915	0.13	1209	0.12	1209	0.12	1112	0.12	33118	0.34	1209	0.12
profisafe_i4	80	63	0.09	41	0.08	69	0.11	70	0.1	63	0.09	41	0.08
profisafe_i4_host	28	36	0.06	36	0.06	64	0.08	36	0.05	58	0.08	36	0.05
profisafe_i4_slave	14	5	0.03	5	0.03	5	0.03	12	0.03	5	0.02	5	0.03
profisafe_i5	88	70	0.13	41	0.09	69	0.12	70	0.13	63	0.1	41	0.09
profisafe_i5_host	28	36	0.06	36	0.06	64	0.1	36	0.06	58	0.09	36	0.06
profisafe_i6	94	70	0.14	41	0.11	69	0.15	70	0.14	63	0.13	41	0.11
profisafe_i6_host	28	36	0.07	36	0.07	64	0.12	36	0.07	58	0.1	36	0.07
profisafe_inclusion_i4host	78	1728	0.16	324	0.12	761	0.17	72	0.09	333	0.13	252	0.1
profisafe_inclusion_o4host	84	1721	0.16	324	0.13	761	0.18	72	0.1	333	0.14	252	0.1
profisafe_inclusion_o4slave	84	1728	0.17	324	0.14	761	0.18	72	0.11	333	0.14	252	0.11
profisafe_o4	90	63	0.11	41	0.09	69	0.12	70	0.11	63	0.1	41	0.08
profisafe_o4_host	30	36	0.06	36	0.06	64	0.09	36	0.05	58	0.08	36	0.05
profisafe_o4_slave	16	5	0.03	5	0.03	5	0.03	12	0.04	5	0.03	5	0.03
profisafe_o5	99	41	0.1	41	0.1	69	0.14	48	0.11	63	0.12	41	0.1
profisafe_o5_host	30	36	0.06	36	0.07	64	0.1	36	0.06	58	0.09	36	0.06
profisafe_o6	106	41	0.12	41	0.12	69	0.16	48	0.13	63	0.14	41	0.11
profisafe_o6_host	30	36	0.07	36	0.07	64	0.12	36	0.07	58	0.1	36	0.08
ftechnik	36		82.48		82.47		115.13		150.36		15.95	534861	2.67
rhone_tough	61		11.92		11.92		8.39		8.77		8.71		22.01
tbed_uncont	84		26.74		30.28		59.43		71.68		34.99		17

Table 3.6
MODULAR CONTROLLABILITY CHECK NOT PREFERRING PLANTS,
LARGEST CONTROLLER FIRST

Model		Modular controllability, not preferring plants, largest controller first											
		All		Early NotAccept		Late NotAccept		MaxCommon Events		MaxCommon Uncontr		Max States	
Name	Aut	States	Time	States	Time	States	Time	States	Time	States	Time	States	Time
big_bmw	31	1031	0.04	5036	0.08	4326	0.07	422	0.05	168	0.04	5463	0.08
fzelle	67	7922	0.16	23619	0.22	1249	0.14	1545	0.15	2212	0.17	2122	0.15
rhone_alps	35	17792	0.11	25217	0.15	16327	0.11	16018	0.09	10293	0.09	16857	0.11
tbed_ctct	84	119806	0.42	18391	0.13		17.66	18391	0.13	22298	0.14		26.09
tbed_nocoll	84		18.59		57.29		26.98	107872	0.61		76.86		41.71
tbed_noderail	84		18.87		57.2		30.07	111581	0.62		66.67		44.34
verriegel4	65	15625	0.19	31306	0.39	688	0.12	24642	0.3	12479	0.2	32858	0.39
profisafe_i4	80	44	0.08	124	0.12	63	0.09	41	0.07	41	0.09	41	0.07
profisafe_i4_host	28	36	0.05	112	0.32	58	0.08	36	0.06	36	0.07	36	0.05
profisafe_i4_slave	14	8	0.02	12	0.04	5	0.03	5	0.02	5	0.03	5	0.03
profisafe_i5	88	44	0.1	124	0.13	63	0.12	41	0.09	41	0.1	41	0.09
profisafe_i5_host	28	36	0.07	112	0.1	58	0.09	36	0.06	36	0.08	36	0.06
profisafe_i6	94	44	0.11	124	0.15	63	0.14	41	0.11	41	0.12	41	0.11
profisafe_i6_host	28	36	0.07	112	0.12	58	0.1	36	0.08	36	0.1	36	0.07
profisafe_inclusion_i4host	78	164	0.07	143	0.1	52	0.07	66	0.08	214	0.11	130	0.08
profisafe_inclusion_o4host	84	164	0.08	143	0.11	52	0.07	66	0.08	214	0.13	130	0.09
profisafe_inclusion_o4slave	84	164	0.08	143	0.11	52	0.08	66	0.08	214	0.13	130	0.09
profisafe_o4	90	44	0.09	124	0.12	63	0.11	41	0.08	41	0.1	41	0.08
profisafe_o4_host	30	36	0.05	112	0.09	58	0.08	36	0.06	36	0.07	36	0.05
profisafe_o4_slave	16	8	0.03	12	0.04	5	0.03	5	0.03	5	0.03	5	0.03
profisafe_o5	99	44	0.1	124	0.15	63	0.13	41	0.1	41	0.12	41	0.1
profisafe_o5_host	30	36	0.06	112	0.11	58	0.09	36	0.06	36	0.08	36	0.06
profisafe_o6	106	44	0.12	124	0.18	63	0.14	41	0.12	41	0.14	41	0.12
profisafe_o6_host	30	36	0.07	112	0.12	58	0.1	36	0.07	36	0.1	36	0.07
ftechnik	36		31.9	177113	1.18		89.16	560434	2.76		21.24		67.03
rhone_tough	61		9.3		24.76		11.82		12.7		8.69		20.78
tbed_uncont	84		18.43		57.22		26.93		30.74		38.73		41.5

Model		Min Events		Min NewEvents		Min States		Min Transitions		One		RelMax Common	
Name	Aut	States	Time	States	Time	States	Time	States	Time	States	Time	States	Time
big_bmw	31	1046	0.06	168	0.04	168	0.04	168	0.05	5056	0.07	168	0.05
fzelle	67	61907	0.37	1653	0.16	71229	0.39	94487	0.45	2311	0.15	1625	0.16
rhone_alps	35	1158	0.05	872	0.05	1158	0.05	1158	0.05	1158	0.05	872	0.04
tbed_ctct	84		17.68		13.51		20.79		41.93		24	1951443	8.01
tbed_nocoll	84		36.48		21.99		68.89		26.1		34.76	624389	2.33
tbed_noderail	84		30.96		23.25		68.58		15.12		51.52		30.31
verriegel4	65	1915	0.13	587	0.11	587	0.11	587	0.11	33118	0.34	587	0.11
profisafe_i4	80	17	0.07	17	0.07	39	0.09	24	0.07	63	0.09	17	0.07
profisafe_i4_host	28	12	0.05	12	0.05	34	0.08	12	0.04	58	0.07	12	0.05
profisafe_i4_slave	14	5	0.03	5	0.02	5	0.03	12	0.03	5	0.03	5	0.03
profisafe_i5	88	24	0.09	17	0.08	39	0.12	24	0.09	63	0.11	17	0.08
profisafe_i5_host	28	12	0.05	12	0.06	34	0.08	12	0.05	58	0.09	12	0.06
profisafe_i6	94	24	0.11	17	0.09	39	0.12	24	0.11	63	0.12	17	0.09
profisafe_i6_host	28	12	0.06	12	0.06	34	0.1	12	0.06	58	0.1	12	0.06
profisafe_inclusion_i4host	78	22	0.07	76	0.08	34	0.07	32	0.07	333	0.12	24	0.06
profisafe_inclusion_o4host	84	15	0.06	76	0.09	34	0.09	32	0.08	333	0.14	24	0.07
profisafe_inclusion_o4slave	84	22	0.07	76	0.09	34	0.09	32	0.08	333	0.13	24	0.07
profisafe_o4	90	17	0.08	17	0.08	39	0.1	24	0.08	63	0.1	17	0.08
profisafe_o4_host	30	12	0.05	12	0.05	34	0.07	12	0.05	58	0.08	12	0.05
profisafe_o4_slave	16	5	0.03	5	0.03	5	0.03	12	0.04	5	0.03	5	0.03
profisafe_o5	99	17	0.08	17	0.09	39	0.12	24	0.1	63	0.12	17	0.09
profisafe_o5_host	30	12	0.05	12	0.06	34	0.09	12	0.05	58	0.09	12	0.06
profisafe_o6	106	17	0.1	17	0.1	39	0.14	24	0.12	63	0.14	17	0.1
profisafe_o6_host	30	12	0.06	12	0.06	34	0.1	12	0.06	58	0.1	12	0.06
ftechnik	36		54.51		67.04		78.65	383559	100.32		15.86		1.92
rhone_tough	61		17.26		17.76		9.32		8.77		8.64		13.11
tbed_uncont	84		36.11		19.74		68.53		26.08		34.69		33.85

Table 3.7
MODULAR CONTROLLABILITY CHECK PREFERRING PLANTS, SMALLEST CONTROLLER FIRST

Model		Modular controllability, preferring plants, smallest controller first											
		All		Early NotAccept		Late NotAccept		MaxCommon Events		MaxCommon Uncontr		Max States	
Name	Aut	States	Time	States	Time	States	Time	States	Time	States	Time	States	Time
big_bmw	31	393	0.06	407	0.06	403	0.06	403	0.06	161	0.05	407	0.05
fzelle	67	9951	0.19	14748	0.3	6267	0.24	4168	0.19	4470	0.18	4485	0.2
rhone_alps	35	308992	1.21	1066042	4.9	12031	0.08	8951	0.08	12031	0.08	8951	0.08
tbed_ctct	84	119806	0.41	18391	0.13		22.37	18391	0.12	22298	0.14		33.73
tbed_nocoll	84		21.96		51.91		28.94		31.32		31.56		67.33
tbed_noderail	84		22.56		52.58		36.26		32.11		32.2		46.26
verriegel4	65	21271	0.2	25219	0.25	519	0.09	1165	0.1	12497	0.19	33306	0.35
profisafe_i4	80	41	0.09	51	0.1	38	0.09	24	0.07	48	0.09	48	0.09
profisafe_i4_host	28	33	0.06	39	0.07	33	0.06	19	0.05	43	0.07	43	0.07
profisafe_i4_slave	14	8	0.03	12	0.03	5	0.03	5	0.03	5	0.03	5	0.02
profisafe_i5	88	41	0.1	51	0.11	38	0.1	24	0.08	48	0.11	48	0.11
profisafe_i5_host	28	33	0.07	39	0.08	33	0.07	19	0.06	43	0.08	43	0.08
profisafe_i6	94	41	0.12	51	0.13	38	0.37	24	0.09	48	0.12	48	0.12
profisafe_i6_host	28	33	0.08	39	0.1	33	0.08	19	0.07	43	0.1	43	0.1
profisafe_inclusion_i4host	78	94	0.08	139	0.1	32	0.07	37	0.07	195	0.11	253	0.13
profisafe_inclusion_o4host	84	94	0.09	139	0.1	32	0.08	37	0.08	195	0.13	253	0.14
profisafe_inclusion_o4slave	84	94	0.09	139	0.1	32	0.07	37	0.08	195	0.13	253	0.14
profisafe_o4	90	41	0.1	51	0.11	38	0.09	24	0.08	48	0.1	48	0.1
profisafe_o4_host	30	33	0.06	39	0.07	33	0.06	19	0.05	43	0.07	43	0.07
profisafe_o4_slave	16	8	0.04	12	0.04	5	0.03	5	0.03	5	0.03	5	0.03
profisafe_o5	99	41	0.11	51	0.13	38	0.11	24	0.09	48	0.12	48	0.12
profisafe_o5_host	30	33	0.07	39	0.09	33	0.07	19	0.06	43	0.08	43	0.08
profisafe_o6	106	41	0.13	51	0.14	38	0.13	24	0.11	48	0.13	48	0.14
profisafe_o6_host	30	33	0.09	39	0.1	33	0.08	19	0.07	43	0.1	43	0.09
ftechnik	36	1663577	12.28	4180499	30.49		16.03	5511458	39.84		37.44		27.89
rhone_tough	61		10.45		9.82		13.31		8.47		8.9		10.2
tbed_uncont	84		21.84		51.95		28.88		31.42		31.55		67.22

Model		Min Events		Min NewEvents		Min States		Min Transitions		One		RelMax Common	
Name	Aut	States	Time	States	Time	States	Time	States	Time	States	Time	States	Time
big_bmw	31	161	0.05	161	0.04	161	0.05	161	0.04	363	0.05	161	0.05
fzelle	67	7132	0.23	4936	0.19	13670	0.27	14470	0.27	4722	0.18	5059	0.19
rhone_alps	35	12029	0.07	986	0.04	320169	1.25	320169	1.25	12029	0.08	986	0.04
tbed_ctct	84		15.85		32.01		34.52		11.31		25.75	1951443	7.55
tbed_nocoll	84		23.52		14.13		58.24		37.08		20.73		31.42
tbed_noderail	84		23.02		14.31		58.57		46.31		40.95		32.03
verriegel4	65	1933	0.14	597	0.1	597	0.11	597	0.1	33136	0.34	485	0.09
profisafe_i4	80	24	0.07	24	0.07	54	0.1	31	0.08	48	0.09	24	0.07
profisafe_i4_host	28	19	0.05	19	0.05	49	0.08	19	0.06	43	0.07	19	0.05
profisafe_i4_slave	14	5	0.03	5	0.03	5	0.03	12	0.03	5	0.02	5	0.03
profisafe_i5	88	31	0.09	24	0.08	54	0.12	31	0.09	48	0.11	24	0.08
profisafe_i5_host	28	19	0.06	19	0.06	49	0.1	19	0.06	43	0.08	19	0.06
profisafe_i6	94	31	0.1	24	0.09	54	0.14	31	0.11	48	0.12	24	0.09
profisafe_i6_host	28	19	0.07	19	0.07	49	0.11	19	0.07	43	0.09	19	0.07
profisafe_inclusion_i4host	78	29	0.07	83	0.08	49	0.09	47	0.08	252	0.12	31	0.07
profisafe_inclusion_o4host	84	22	0.07	83	0.09	49	0.09	47	0.09	252	0.14	31	0.08
profisafe_inclusion_o4slave	84	29	0.08	83	0.09	49	0.09	47	0.09	252	0.14	31	0.08
profisafe_o4	90	24	0.08	24	0.08	54	0.11	31	0.09	48	0.1	24	0.08
profisafe_o4_host	30	19	0.05	19	0.05	49	0.09	19	0.06	43	0.08	19	0.05
profisafe_o4_slave	16	5	0.03	5	0.03	5	0.03	12	0.04	5	0.03	5	0.03
profisafe_o5	99	24	0.09	24	0.09	54	0.13	31	0.1	48	0.12	24	0.09
profisafe_o5_host	30	19	0.06	19	0.06	49	0.1	19	0.06	43	0.09	19	0.06
profisafe_o6	106	24	0.11	24	0.11	54	0.15	31	0.12	48	0.13	24	0.11
profisafe_o6_host	30	19	0.07	19	0.07	49	0.11	19	0.07	43	0.09	19	0.07
ftechnik	36		25.2		21.69	3465198	25.3		23.89		37.05		21.59
rhone_tough	61		13.28		9.22		25.93		29.75		8.96		9.18
tbed_uncont	84		23.49		14.17		58.33		36.96		20.86		31.43

Table 3.8
MODULAR CONTROLLABILITY CHECK NOT PREFERRING PLANTS,
SMALLEST CONTROLLER FIRST

Model		Modular controllability, not preferring plants, largest controller first											
		All		Early NotAccept		Late NotAccept		MaxCommon Events		MaxCommon Uncontr		Max States	
Name	Aut	States	Time	States	Time	States	Time	States	Time	States	Time	States	Time
big_bmw	31	1728	0.05	407	0.05	403	0.04	435	0.05	161	0.04	8758	0.09
fzelle	67	7719	0.14	29194	0.28	3652	0.18	1839	0.13	1927	0.13	2026	0.12
rhone_alps	35	225231	0.89	1066042	4.91	12031	0.07	9009	0.07	12161	0.1	9849	0.08
tbed_ctct	84	119806	0.4	18391	0.13		22.21	18391	0.13	22298	0.14		33.77
tbed_nocoll	84		25.71		57.07		21.1	613031	2.19	1754041	6.91		25.44
tbed_noderail	84		25.89		56.73		72.4	657923	2.33	1775027	7.08		23.46
verriegel4	65		19.44	24576	0.28	519	0.1	24530	0.29	12497	0.2	32773	0.38
profisafe_i4	80	56	0.09	124	0.12	38	0.08	41	0.07	48	0.1	41	0.07
profisafe_i4_host	28	48	0.06	112	0.09	33	0.06	36	0.05	43	0.08	36	0.05
profisafe_i4_slave	14	8	0.03	12	0.03	5	0.03	5	0.03	5	0.03	5	0.02
profisafe_i5	88	56	0.1	124	0.14	38	0.1	41	0.09	48	0.11	41	0.08
profisafe_i5_host	28	48	0.08	112	0.1	33	0.07	36	0.06	43	0.09	36	0.07
profisafe_i6	94	56	0.12	124	0.16	38	0.11	41	0.1	48	0.13	41	0.11
profisafe_i6_host	28	48	0.08	112	0.12	33	0.08	36	0.07	43	0.1	36	0.07
profisafe_inclusion_i4host	78	184	0.08	143	0.1	32	0.07	66	0.07	195	0.12	130	0.08
profisafe_inclusion_o4host	84	184	0.09	143	0.1	32	0.08	66	0.08	195	0.13	130	0.09
profisafe_inclusion_o4slave	84	184	0.09	143	0.1	32	0.08	66	0.08	195	0.13	130	0.09
profisafe_o4	90	56	0.1	124	0.13	38	0.09	41	0.08	48	0.1	41	0.08
profisafe_o4_host	30	48	0.06	112	0.09	33	0.07	36	0.05	43	0.07	36	0.05
profisafe_o4_slave	16	8	0.04	12	0.04	5	0.03	5	0.03	5	0.03	5	0.03
profisafe_o5	99	56	0.11	124	0.15	38	0.11	41	0.1	48	0.13	41	0.1
profisafe_o5_host	30	48	0.07	112	0.1	33	0.07	36	0.06	43	0.09	36	0.06
profisafe_o6	106	56	0.13	124	0.17	38	0.13	41	0.12	48	0.14	41	0.12
profisafe_o6_host	30	48	0.08	112	0.11	33	0.08	36	0.07	43	0.1	36	0.07
ftechnik	36	159004	1.23	21063	0.16	128	0.04	128	0.03	128	0.03	336808	1.11
rhone_tough	61		11.39		15.75		15.89		11.14		33.37		14.58
tbed_uncont	84		25.69		56.48		21.05		30.72		54.33		77.84

Model		Min Events		Min NewEvents		Min States		Min Transitions		One		RelMax Common	
Name	Aut	States	Time	States	Time	States	Time	States	Time	States	Time	States	Time
big_bmw	31	161	0.04	161	0.04	161	0.04	161	0.04	363	0.04	161	0.04
fzelle	67	7132	0.24	2765	0.14	30054	0.29	30731	0.3	4722	0.16	2609	0.13
rhone_alps	35	12029	0.08	986	0.04	320169	1.25	320169	1.25	12029	0.08	986	0.04
tbed_ctct	84		15.77		31.76		34.19		11.25		25.68	1951443	7.52
tbed_nocoll	84		20.02		14.17		60.31		20		20.78	581749	2.25
tbed_noderail	84		22.94		29.46		60.34		36.01		40.73	549120	2.1
verriegel4	65	1933	0.13	597	0.11	597	0.11	597	0.11	33136	0.34	485	0.09
profisafe_i4	80	24	0.07	24	0.07	54	0.1	31	0.09	48	0.09	24	0.07
profisafe_i4_host	28	19	0.05	19	0.05	49	0.08	19	0.05	43	0.07	19	0.05
profisafe_i4_slave	14	5	0.03	5	0.03	5	0.03	12	0.03	5	0.03	5	0.02
profisafe_i5	88	31	0.1	24	0.08	54	0.13	31	0.1	48	0.11	24	0.09
profisafe_i5_host	28	19	0.06	19	0.06	49	0.1	19	0.06	43	0.09	19	0.06
profisafe_i6	94	31	0.11	24	0.1	54	0.14	31	0.11	48	0.12	24	0.1
profisafe_i6_host	28	19	0.07	19	0.07	49	0.11	19	0.07	43	0.1	19	0.07
profisafe_inclusion_i4host	78	29	0.07	83	0.08	49	0.09	47	0.09	252	0.13	31	0.07
profisafe_inclusion_o4host	84	22	0.07	83	0.09	49	0.09	47	0.09	252	0.13	31	0.08
profisafe_inclusion_o4slave	84	29	0.08	83	0.09	49	0.1	47	0.09	252	0.14	31	0.07
profisafe_o4	90	24	0.08	24	0.08	54	0.12	31	0.09	48	0.1	24	0.08
profisafe_o4_host	30	19	0.05	19	0.05	49	0.09	19	0.05	43	0.07	19	0.05
profisafe_o4_slave	16	5	0.03	5	0.03	5	0.03	12	0.04	5	0.03	5	0.03
profisafe_o5	99	24	0.1	24	0.09	54	0.14	31	0.11	48	0.12	24	0.09
profisafe_o5_host	30	19	0.06	19	0.06	49	0.1	19	0.06	43	0.08	19	0.06
profisafe_o6	106	24	0.11	24	0.11	54	0.16	31	0.12	48	0.13	24	0.11
profisafe_o6_host	30	19	0.07	19	0.07	49	0.11	19	0.07	43	0.1	19	0.07
ftechnik	36		18.94	128	0.03		26.16		14.45		36.91	128	0.03
rhone_tough	61		13.09		48.03		26.06		43.19		8.95		21.27
tbed_uncont	84		20.07		14.16		59.89		20.13		20.81		36.69

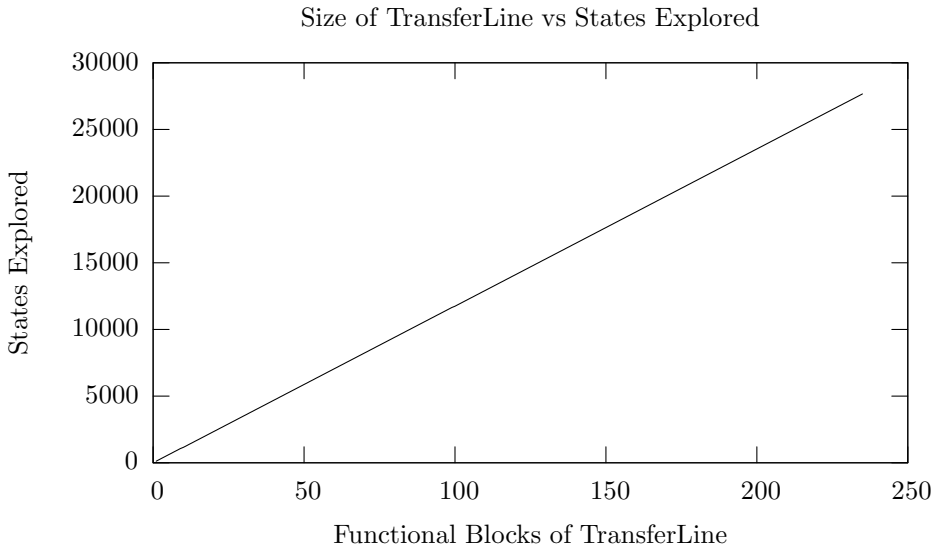
Size of TransferLine vs States Explored
States Explored
30000
25000
20000
15000
10000
5000
0
0
50
100
150
200
250
Functional Blocks of TransferLine

Figure 3.3

Chapter 4

Parallel Checker

The parallel checker instead of trying to prove that each controller is controllable one after the other instead tries interleave the process of proving each controller with proving every other controller. This is for two basic reasons. Firstly, because if there is one specification in the model for which it is easy to prove that it is not controllable, the parallel checker should be capable of discovering this relatively quickly, whereas the standard approach may have to waist time proving other specifications first. Secondly, if the system as a whole is controllable, it may be useful to use solve those specifications which, can be proved quickly, early on, so as to aid in proving the harder specifications.

4.1 Algorithm

The algorithm is detailed in Figure 4.1. This algorithm behaves like the standard modular algorithm detailed in Figure 3.1. The main difference is that, instead of just picking a controller to prove and then sticking with it, we instead for each controller do one iteration to attempt to prove it, and then proceed to do an iteration for the next controller down the line. In addition, if we manage to prove a controller controllable, we proceed to look through all the other compositions and remove from them any automata which were added after the proven automata. This is because now that the controller can be considered as a plant, it is quite likely that many counterexamples disappear and thus some of the plants added afterwards could very well be completely unnecessary.

4.2 Results

Most of what was stated in the results section for the modular controllability checker is also true here. The only thing of any particular significance is how efficient the parallel checker seems to be at solving the ftechnik example especially considering how much trouble the standard modular checker has trying to solve it. The reason for this is that ftechnik has a counterexample which

C equals the set of Controller automata in the model and P the set of Plant automata, and S is the queue of sets of composed automata.

1. Set the queue S as being empty.
2. For every controller C_i create the list S_i. Add C_i to the front of it and add it to S
3. If S is empty the model has been proven controllable. Otherwise, remove S_i from the front of S.
4. Check controllability of S_I using the monolithic method. Consider all automata which are elements of C as controllers and all automata which are elements of P as plants.
5. If no counterexample for S was found go to 9. Otherwise set t to be the counterexample found by the controllability check.
6. Set the set N to contain all automata in P and C which would not accept the counterexample t. Take into consideration for all automata in C that specifications in addition to not accepting t must also not consider t as being a counterexample to their controller.
7. If N is empty then the model has been proven uncontrollable and t represents a counterexample in the system. Otherwise pick a subset of N to add to the end of S_i.
8. Add S_i on to the end of S then go to 3.
9. For all elements C_j of S_i if they are also an element of C, remove them from C and add them to P, also for all lists in S remove all automata added after C_i if it is present. Then go to 3.

Figure 4.1: Modular Controllability checking algorithm

is only one event long, and thus can be found very quickly. Thus the parallel checker has an extreme advantage here in that it can start looking at the controller which speaks of this counterexample right of the bat without going to much effort trying to prove the other controllers. This however represents an extreme case which is not indicative of most models. Thus it would have been good if we had more uncontrollable models which were harder but not as hard as tbed_uncont or rhone_tough to test to see if this same effect can come in handy on less extreme cases as well.

It should be noted that language inclusion is not looked at here as for most language inclusion problems we only consider one property at a time. Thus, the parallel checker is unlikely to give interesting results.

Table 4.1
PARALLEL CONTROLLABILITY CHECKER PREFERRING PLANTS

Model		Parallel controllability, preferring plants											
		All		Early NotAccept		Late NotAccept		MaxCommon Events		MaxCommon Uncontr		Max States	
Name	Aut	States	Time	States	Time	States	Time	States	Time	States	Time	States	Time
big_bmw	31	1065	0.07	207	0.06	350	0.05	442	0.06	167	0.05	5479	0.11
fzelle	67	7885	0.19	9670	0.31	2240	0.18	2405	0.19	2311	0.19	2122	0.16
rhone_alps	35	225113	0.9	1040434	4.92	1720	0.06	1544	0.06	3440	0.07	1926	0.07
tbed_ctct	84	119806	0.43	18391	0.14		27.16	18391	0.14	22298	0.15		52.13
tbed_nocoll	84		24.14		267.03		41.64		52.25		65.13		107.54
tbed_noderail	84		24.39		267.11		233.02		50.75		75.25		78.47
verriegel4	65	3511	0.17	25888	0.34	597	0.14	6335	0.22	12575	0.24	33713	0.43
profisafe_i4	80	51	0.11	61	0.12	38	0.1	48	0.11	48	0.1	48	0.11
profisafe_i4_host	28	43	0.08	49	0.1	33	0.07	43	0.08	43	0.08	43	0.08
profisafe_i4_slave	14	8	0.03	12	0.04	5	0.03	5	0.02	5	0.03	5	0.03
profisafe_i5	88	51	0.13	61	0.14	38	0.11	48	0.13	48	0.14	48	0.12
profisafe_i5_host	28	43	0.1	49	0.11	33	0.08	43	0.1	43	0.09	43	0.09
profisafe_i6	94	51	0.14	61	0.17	38	0.13	48	0.15	48	0.15	48	0.14
profisafe_i6_host	28	43	0.11	49	0.13	33	0.09	43	0.11	43	0.11	43	0.11
profisafe_inclusion_i4host	78	184	0.09	177	0.1	59	0.09	88	0.1	195	0.13	137	0.11
profisafe_inclusion_o4host	84	184	0.1	177	0.12	59	0.1	88	0.11	195	0.14	137	0.11
profisafe_inclusion_o4slave	84	184	0.1	177	0.12	59	0.1	88	0.12	195	0.14	137	0.11
profisafe_o4	90	51	0.12	61	0.14	38	0.12	48	0.12	48	0.12	48	0.12
profisafe_o4_host	30	43	0.08	49	0.09	33	0.07	43	0.08	43	0.09	43	0.09
profisafe_o4_slave	16	8	0.04	12	0.04	5	0.04	5	0.04	5	0.04	5	0.04
profisafe_o5	99	51	0.14	61	0.16	38	0.13	48	0.14	48	0.14	48	0.13
profisafe_o5_host	30	43	0.1	49	0.11	33	0.09	43	0.1	43	0.1	43	0.09
profisafe_o6	106	51	0.17	61	0.18	38	0.14	48	0.16	48	0.16	48	0.16
profisafe_o6_host	30	43	0.11	49	0.13	33	0.09	43	0.11	43	0.11	43	0.11
ftechnik	36	113	0.05	113	0.05	113	0.04	113	0.05	113	0.04	113	0.05
rhone_tough	61		13.54		28.74		32.97		23.09		9.1		28.24
tbed_uncont	84		24.03		267		41.48		205.83		170.16		188.66

Model		Min Events		Min NewEvents		Min States		Min Transitions		One		RelMax Common	
Name	Aut	States	Time	States	Time	States	Time	States	Time	States	Time	States	Time
big_bmw	31	174	0.05	174	0.05	174	0.05	174	0.04	381	0.05	174	0.05
fzelle	67	5681	0.26	2574	0.18	7266	0.3	10792	0.31	4464	0.22	2670	0.18
rhone_alps	35	1697	0.07	1720	0.06	17100	0.12	17077	0.12	1720	0.05	1720	0.06
tbed_ctct	84		45.84		41.28		40.88		16.34		12.45	1951443	7.69
tbed_nocoll	84		62.23		64.08		158.22		67.25		155.68		55.7
tbed_noderail	84		170.83		202.59		157.93		177.41		160.45		56.03
verriegel4	65	2011	0.17	675	0.15	675	0.15	675	0.14	33214	0.39	563	0.13
profisafe_i4	80	38	0.09	38	0.09	54	0.12	45	0.11	48	0.1	38	0.09
profisafe_i4_host	28	33	0.08	33	0.07	49	0.09	33	0.07	43	0.08	33	0.07
profisafe_i4_slave	14	5	0.03	5	0.03	5	0.03	12	0.03	5	0.03	5	0.03
profisafe_i5	88	45	0.12	38	0.11	54	0.14	45	0.12	48	0.12	38	0.11
profisafe_i5_host	28	33	0.08	33	0.08	49	0.11	33	0.08	43	0.09	33	0.08
profisafe_i6	94	45	0.14	38	0.13	54	0.16	45	0.14	48	0.14	38	0.13
profisafe_i6_host	28	33	0.09	33	0.09	49	0.12	33	0.09	43	0.1	33	0.1
profisafe_inclusion_i4host	78	90	0.1	83	0.09	49	0.09	56	0.11	252	0.14	31	0.08
profisafe_inclusion_o4host	84	83	0.1	83	0.1	49	0.11	56	0.11	252	0.15	31	0.09
profisafe_inclusion_o4slave	84	90	0.1	83	0.1	49	0.1	56	0.11	252	0.15	31	0.09
profisafe_o4	90	38	0.1	38	0.1	54	0.14	45	0.11	48	0.12	38	0.1
profisafe_o4_host	30	33	0.07	33	0.07	49	0.1	33	0.07	43	0.08	33	0.08
profisafe_o4_slave	16	5	0.04	5	0.04	5	0.04	12	0.05	5	0.04	5	0.04
profisafe_o5	99	38	0.13	38	0.12	54	0.16	45	0.13	48	0.13	38	0.13
profisafe_o5_host	30	33	0.08	33	0.09	49	0.1	33	0.08	43	0.09	33	0.08
profisafe_o6	106	38	0.14	38	0.15	54	0.17	45	0.15	48	0.15	38	0.14
profisafe_o6_host	30	33	0.09	33	0.09	49	0.12	33	0.09	43	0.11	33	0.09
ftechnik	36	113	0.05	113	0.05	113	0.04	113	0.05	113	0.04	113	0.05
rhone_tough	61		11.95		21.71		15.29		11.19		10.01		36.87
tbed_uncont	84		62.59		64.53		158.55		67.3		156.23		140.03

Table 4.2
PARALLEL CONTROLLABILITY CHECKER NOT PREFERRING PLANTS

Model		Parallel checker, not preferring plants											
		All		Early NotAccept		Late NotAccept		MaxCommon Events		MaxCommon Uncontr		Max States	
Name	Aut	States	Time	States	Time	States	Time	States	Time	States	Time	States	Time
big_bmw	31	1065	0.06	207	0.05	350	0.05	442	0.06	167	0.05	5479	0.09
fzelle	67	7919	0.2	16054	0.43	2240	0.19	2245	0.18	2311	0.2	2122	0.17
rhone_alps	35	225085	0.9	1040434	4.88	56859	0.27	1544	0.06	11853	0.12	56173	0.25
tbed_ctct	84	119806	0.43	18391	0.14		27.25	18391	0.14	22298	0.15		52.09
tbed_nocoll	84		22.06		478.75		142.36	184691	1.03	337945	1.57		231.03
tbed_noderail	84		22.37		480.16		100.15	188302	1.06	339078	1.58		429.99
verriegel4	65	3799	0.18	23384	0.33	597	0.13	23399	0.33	12575	0.25	32833	0.42
profisafe_i4	80	56	0.11	124	0.13	38	0.1	41	0.09	48	0.1	41	0.09
profisafe_i4_host	28	48	0.08	112	0.1	33	0.07	36	0.07	43	0.08	36	0.06
profisafe_i4_slave	14	8	0.03	12	0.03	5	0.03	5	0.03	5	0.03	5	0.02
profisafe_i5	88	56	0.12	124	0.16	38	0.11	41	0.11	48	0.13	41	0.1
profisafe_i5_host	28	48	0.1	112	0.11	33	0.08	36	0.07	43	0.1	36	0.08
profisafe_i6	94	56	0.14	124	0.18	38	0.13	41	0.12	48	0.15	41	0.12
profisafe_i6_host	28	48	0.1	112	0.13	33	0.1	36	0.09	43	0.12	36	0.08
profisafe_inclusion_i4host	78	184	0.1	143	0.11	23	0.08	66	0.08	195	0.14	130	0.1
profisafe_inclusion_o4host	84	184	0.11	143	0.12	23	0.08	66	0.09	195	0.14	130	0.11
profisafe_inclusion_o4slave	84	184	0.11	143	0.12	23	0.08	66	0.09	195	0.14	130	0.1
profisafe_o4	90	56	0.11	124	0.15	38	0.11	41	0.1	48	0.12	41	0.1
profisafe_o4_host	30	48	0.08	112	0.1	33	0.08	36	0.06	43	0.08	36	0.06
profisafe_o4_slave	16	8	0.04	12	0.04	5	0.04	5	0.04	5	0.04	5	0.04
profisafe_o5	99	56	0.14	124	0.18	38	0.13	41	0.12	48	0.14	41	0.12
profisafe_o5_host	30	48	0.09	112	0.12	33	0.08	36	0.07	43	0.1	36	0.08
profisafe_o6	106	56	0.16	124	0.2	38	0.15	41	0.14	48	0.16	41	0.13
profisafe_o6_host	30	48	0.11	112	0.13	33	0.1	36	0.09	43	0.11	36	0.09
ftechnik	36	113	0.06	113	0.05	113	0.05	113	0.05	113	0.04	113	0.05
rhone_tough	61		17.38		77.38		33.2		23.09		9.11		28.6
tbed_uncont	84		22.01		479.59		142.31		169.32		128.65		231.22

Model		Min Events		Min NewEvents		Min States		Min Transitions		One		RelMax Common	
Name	Aut	States	Time	States	Time	States	Time	States	Time	States	Time	States	Time
big_bmw	31	174	0.05	174	0.04	174	0.05	174	0.05	381	0.05	174	0.05
fzelle	67	6183	0.28	2574	0.18	19728	0.48	16174	0.47	4464	0.22	2510	0.18
rhone_alps	35	2134	0.08	1022	0.06	1308	0.07	1308	0.06	1720	0.06	1022	0.06
tbed_ctct	84		46.1		41.07		40.97		16.33		12.42	1951443	7.69
tbed_nocoll	84		17.95		40.62		299.94		61.9		155.23	306856	1.48
tbed_noderail	84		235.13		281.84		300.34		403.78		160.95	300259	1.38
verriegel4	65	2011	0.18	675	0.15	675	0.16	675	0.15	33214	0.38	563	0.13
profisafe_i4	80	24	0.09	24	0.09	54	0.12	31	0.09	48	0.1	24	0.09
profisafe_i4_host	28	19	0.06	19	0.06	49	0.09	19	0.06	43	0.08	19	0.05
profisafe_i4_slave	14	5	0.03	5	0.03	5	0.03	12	0.04	5	0.02	5	0.03
profisafe_i5	88	31	0.11	24	0.1	54	0.14	31	0.11	48	0.12	24	0.1
profisafe_i5_host	28	19	0.07	19	0.07	49	0.11	19	0.07	43	0.09	19	0.08
profisafe_i6	94	31	0.13	24	0.12	54	0.16	31	0.13	48	0.14	24	0.12
profisafe_i6_host	28	19	0.09	19	0.08	49	0.12	19	0.08	43	0.11	19	0.08
profisafe_inclusion_i4host	78	29	0.08	83	0.1	49	0.1	47	0.09	252	0.14	31	0.08
profisafe_inclusion_o4host	84	22	0.08	83	0.11	49	0.1	47	0.11	252	0.14	31	0.09
profisafe_inclusion_o4slave	84	29	0.09	83	0.1	49	0.11	47	0.1	252	0.15	31	0.09
profisafe_o4	90	24	0.09	24	0.09	54	0.14	31	0.1	48	0.11	24	0.1
profisafe_o4_host	30	19	0.06	19	0.06	49	0.1	19	0.05	43	0.08	19	0.06
profisafe_o4_slave	16	5	0.04	5	0.04	5	0.04	12	0.04	5	0.04	5	0.04
profisafe_o5	99	24	0.12	24	0.12	54	0.15	31	0.12	48	0.13	24	0.12
profisafe_o5_host	30	19	0.07	19	0.08	49	0.11	19	0.08	43	0.09	19	0.07
profisafe_o6	106	24	0.14	24	0.13	54	0.18	31	0.14	48	0.16	24	0.13
profisafe_o6_host	30	19	0.08	19	0.08	49	0.13	19	0.08	43	0.11	19	0.08
ftechnik	36	113	0.04	113	0.05	113	0.04	113	0.05	113	0.04	113	0.05
rhone_tough	61		73.51		46.84		13.47		10.24		9.96		36.71
tbed_uncont	84		17.99		46.79		299.72		62.05		155.88		107.95

Chapter 5

Culling Checker

As can be seen in the previous sections, what heuristic we use to decide which new automata to add to the composition can have a drastic effect on how long it will take to prove a model controllable and even on whether we will be able to prove a model as being controllable before running out of memory. This shows that it is important that, when we choose automata to be added into the composition, we choose the right ones. Thus, the idea behind the culling controllability checker is that, as we build up the composition of automata that we check for controllability, we also second guess some of the choices we made earlier on and attempt to remove from the composition those automata which aren't particularly good choices.

5.1 Algorithm

This algorithm is detailed in Figure 5.1. The basic concept here is that we use a regular modular controllability checker, but that, whenever we add a new automaton into the composition, we remember all the other automata which could have been added into the composition instead. This is so that whenever we add a new automata to the composition we can check to see if that automaton could have been used previously, and if it could, whether or not it would be worth while to remove the automaton we previously added into the composition.

5.2 Results

Again most of the things related to heuristics and such stated with respect to the standard modular checker apply here also. From looking at the results it seems that most of the time this checker does worse than the standard checker with the same heuristic and the same order of proving automata. There is however the notable exception of MaxCommonEvents and MaxCommonUncontrollable for tbed_nocoll and tbed_noderail. In Tables 3.8 and 5.2 we can see that the standard modular approach took more states than it needed to because of

C equals the set of Controller automata in the model and P the set of Plant automata, S is the set of composed automata, and O is the set of automata we could of added to the composition at each step.

1. Set the set S and O as being empty.
2. if C is empty the model has been proven controllable. Otherwise, take an automaton from the set C and add it to S.
3. Check controllability of S using the monolithic method. Consider automata which are elements of C as controllers and all automata which are elements of P as plants.
4. If no counterexample for S was found go to 9. Otherwise set t to be the counterexample found by the controllability check.
5. Set the set N to contain all automata in P and C which would not accept the counterexample t. Take into consideration for all automata in C that specifications in addition to not accepting t must also not consider t as being a counterexample to their controller.
6. If N is empty then the model has been proven not controllable, and t represents a counterexample in the system. Otherwise pick an automaton n in N to add to S.
7. For all elements in O (a, mo) if $n \in mo$ then check to see if the synchronous product of S is smaller without a if it is remove a from S and (a, mo) from O. In addition set N to equal the intersection of N and mo.
8. Add (n, N) to the list O then go to 3.
9. For all elements of S if they are also an element of C remove them from C and add them to P. Then go to 2.

Figure 5.1: Modular Controllability checking algorithm

choosing unnecessary automata and the culling approach did manage to remove unnecessary automata from the composition, taking significantly fewer states, and qsuggesting there may be some merits to this approach. However, there is still the unfortunate drawback that this approach can take significantly longer than others to work out that it will in fact fail.

Table 5.1
CULLING LANGUAGE INCLUSION

Model		Culling language inclusion											
		All		Early NotAccept		Late NotAccept		MaxCommon Events		Max States		Min Events	
Name	Aut	States	Time	States	Time	States	Time	States	Time	States	Time	States	Time
profisafe_i4_slave	15	10519	0.95	1753	0.25	19377	1.14	12689	0.57	23441	1.24	1660	0.27
profisafe_o4_slave	17	7674	1.09	1664	0.43	23227	1.29	25475	1.09	28701	1.43	3243	0.68
big_bmw	32	1637	0.08	1637	0.06	2197	0.09	1637	0.09	1637	0.09	1985	0.07
ftechnik	37		17.79	4460133	119.07		136.2		445.22		63.2		147.92
tbed_nocoll	85		273.88		158.39		253.06	2301	1.6	2255	95.97		88.18
tbed_noderail	85		305.45		36.51		125.55		194.72		91.55		151.25
verriegel4	66	1680	0.06	1201	0.05	1680	0.06	1201	0.05	1680	0.06	1680	0.06
profisafe_i4_host	29	2730	0.71	5022	0.82	2321	0.51	4666	0.54	3081	0.76	2942	0.42
profisafe_o4_host	31	2730	0.7	5022	0.82	2321	0.52	4666	0.54	3081	0.77	2942	0.42
profisafe_i5_host	29	2928	0.82	5503	0.96	2519	0.6	5342	0.64	3368	0.89	3443	0.5
profisafe_o5_host	31	2928	0.81	5503	0.97	2519	0.6	5342	0.64	3368	0.89	3443	0.51
profisafe_i6_host	29	3126	0.93	5984	1.09	2717	0.69	6018	0.73	3655	1.02	3944	0.58
profisafe_o6_host	31	3126	0.93	5984	1.1	2717	0.69	6018	0.74	3655	1.03	3944	0.59

Model		Min NewEvents		Min States		Min Transitions		One		RelMax Common	
Name	Aut	States	Time	States	Time	States	Time	States	Time	States	Time
profisafe_i4_slave	15	3636	0.45	1413	0.36	1753	0.26	10519	0.95	4190	0.71
profisafe_o4_slave	17	5051	0.65	22822	1.32	24632	1	7674	1.08	12240	1.36
big_bmw	32	1985	0.05	1985	0.04	1985	0.06	1637	0.07	1985	0.05
ftechnik	37		117.52		410.73		757.87		17.75		123.87
tbed_nocoll	85		110.38		208.46		128.6		271.88		48.93
tbed_noderail	85		108.32		110.12		83.86		304.9		139.91
verriegel4	66	1680	0.06	2103	0.07	2103	0.07	1680	0.06	2103	0.07
profisafe_i4_host	29	2101	0.52	2219	0.48	2942	0.42	2730	0.69	2101	0.52
profisafe_o4_host	31	2101	0.52	2219	0.48	2942	0.42	2730	0.7	2101	0.52
profisafe_i5_host	29	2337	0.59	2455	0.57	2066	0.4	2928	0.81	2337	0.58
profisafe_o5_host	31	2337	0.59	2455	0.56	2066	0.4	2928	0.81	2337	0.6
profisafe_i6_host	29	2573	0.66	2691	0.64	2302	0.46	3126	0.92	2573	0.67
profisafe_o6_host	31	2573	0.67	2691	0.66	2302	0.46	3126	0.92	2573	0.67

Table 5.2
CULLING CONTROLLABILITY CHECKER PREFERRING PLANTS, LARGEST CONTROLLER FIRST

Model		Culling checker, preferring plants, largest controller first											
		All		Early NotAccept		Late NotAccept		MaxCommon Events		MaxCommon Uncontr		Max States	
Name	Aut	States	Time	States	Time	States	Time	States	Time	States	Time	States	Time
big_bmw	31	10929	0.12	13970	0.17	13139	0.15	12017	0.13	11011	0.12	13160	0.14
fzelle	67	2392	0.23	1451	0.18	1309	0.2	1660	0.21	2293	0.22	2263	0.21
rhone_alps	35	1142	0.06	4970	0.14	16231	0.13	842	0.04	5254	0.14	16912	0.14
tbed_ctct	84		159.07	21503	0.17		167.65	21503	0.17	22670	0.22		63.19
tbed_nocoll	84		81.65		327.97		47.43		130.3		318.06		232.36
tbed_noderail	84		200.47		450.92		47.72		84.74		343.47		233.33
verriegel4	65	28906	0.46	29611	0.52	923	0.13	2247	0.17	12888	0.29	28929	0.49
profisafe_i4	80	94	0.13	86	0.13	94	0.13	41	0.09	94	0.12	41	0.09
profisafe_i4_host	28	89	0.1	77	0.09	89	0.11	36	0.05	89	0.1	36	0.06
profisafe_i4_slave	14	5	0.02	9	0.03	5	0.02	5	0.03	5	0.03	5	0.03
profisafe_i5	88	94	0.15	86	0.15	94	0.15	41	0.1	94	0.15	41	0.09
profisafe_i5_host	28	89	0.12	77	0.11	89	0.12	36	0.07	89	0.12	36	0.07
profisafe_i6	94	94	0.17	86	0.18	94	0.17	41	0.12	94	0.18	41	0.11
profisafe_i6_host	28	89	0.13	77	0.13	89	0.13	36	0.08	89	0.13	36	0.08
profisafe_inclusion_i4host	78	554	0.34	104	0.11	48	0.08	62	0.08	342	0.22	118	0.1
profisafe_inclusion_o4host	84	554	0.74	104	0.12	48	0.09	62	0.09	342	0.23	118	0.1
profisafe_inclusion_o4slave	84	554	0.36	104	0.12	48	0.08	62	0.09	342	0.24	118	0.11
profisafe_o4	90	94	0.14	86	0.14	94	0.14	41	0.09	94	0.14	41	0.09
profisafe_o4_host	30	89	0.1	77	0.1	89	0.11	36	0.06	89	0.11	36	0.06
profisafe_o4_slave	16	5	0.03	9	0.04	5	0.04	5	0.03	5	0.03	5	0.04
profisafe_o5	99	94	0.17	86	0.56	94	0.17	41	0.11	94	0.17	41	0.11
profisafe_o5_host	30	89	0.12	77	0.49	89	0.12	36	0.07	89	0.12	36	0.07
profisafe_o6	106	94	0.2	86	0.19	94	0.19	41	0.13	94	0.19	41	0.13
profisafe_o6_host	30	89	0.13	77	0.13	89	0.13	36	0.08	89	0.14	36	0.08
ftechnik	36		45.9	253964	3.34		580.29	688070	4.54		146.88		173.8
rhone_tough	61		8.74		21.23		46.27		51.86		8.81		33.29
tbed_uncont	84		60.18		326.51		86.8		38.79		39.13		232.53

Model		Min Events		Min NewEvents		Min States		Min Transitions		One		RelMax Common	
Name	Aut	States	Time	States	Time	States	Time	States	Time	States	Time	States	Time
big_bmw	31	11797	0.13	11797	0.13	14265	0.18	11797	0.13	10929	0.1	11011	0.12
fzelle	67	1633	0.24	1586	0.23	2251	0.27	2096	0.24	2392	0.21	1544	0.22
rhone_alps	35	1142	0.06	842	0.05	1142	0.06	1142	0.06	1142	0.06	842	0.04
tbed_ctct	84		90.53		51.89	5520413	30.36		44.58		158.07	328595	1.6
tbed_nocoll	84		315.55		140.13		139.4		697.72		81.1		296.62
tbed_noderail	84		227.38		147.47		129.35		826.56		200.72		296.29
verriegel4	65	1665	0.16	882	0.14	882	0.13	837	0.13	28906	0.44	882	0.14
profisafe_i4	80	94	0.13	41	0.09	94	0.12	98	0.14	94	0.13	41	0.08
profisafe_i4_host	28	36	0.06	36	0.06	89	0.11	36	0.06	89	0.1	36	0.06
profisafe_i4_slave	14	5	0.03	5	0.03	5	0.02	9	0.04	5	0.02	5	0.03
profisafe_i5	88	98	0.16	41	0.1	94	0.15	98	0.16	94	0.14	41	0.1
profisafe_i5_host	28	36	0.07	36	0.07	89	0.12	36	0.07	89	0.12	36	0.07
profisafe_i6	94	98	0.18	41	0.12	94	0.18	98	0.19	94	0.16	41	0.12
profisafe_i6_host	28	36	0.08	36	0.08	89	0.14	36	0.08	89	0.13	36	0.08
profisafe_inclusion_i4host	78	1891	0.4	714	0.31	844	0.25	93	0.12	554	0.32	272	0.13
profisafe_inclusion_o4host	84	1887	0.42	714	0.32	844	0.27	93	0.13	554	0.32	272	0.13
profisafe_inclusion_o4slave	84	1891	0.42	714	0.32	844	0.26	93	0.13	554	0.33	272	0.14
profisafe_o4	90	94	0.13	41	0.1	94	0.14	98	0.15	94	0.13	41	0.09
profisafe_o4_host	30	36	0.06	36	0.06	89	0.11	36	0.06	89	0.1	36	0.06
profisafe_o4_slave	16	5	0.04	5	0.03	5	0.03	9	0.04	5	0.03	5	0.03
profisafe_o5	99	41	0.11	41	0.11	94	0.17	45	0.11	94	0.16	41	0.12
profisafe_o5_host	30	36	0.07	36	0.07	89	0.12	36	0.07	89	0.12	36	0.07
profisafe_o6	106	41	0.13	41	0.13	94	0.19	45	0.14	94	0.18	41	0.13
profisafe_o6_host	30	36	0.08	36	0.08	89	0.16	36	0.08	89	0.14	36	0.08
ftechnik	36		483.01		183.51		743.3		669.58		45.72	422917	3.59
rhone_tough	61		11.88		11.89		8.64		8.71		8.78		65.82
tbed_uncont	84		315.69		140.47		139.62		696.67		60.02		94.26

Table 5.3
CULLING CONTROLLABILITY CHECKER NOT PREFERRING PLANTS,
LARGEST CONTROLLER FIRST

Model		Culling checker, not preferring plants, largest controller first											
		All		Early NotAccept		Late NotAccept		MaxCommon Events		MaxCommon Uncontr		Max States	
Name	Aut	States	Time	States	Time	States	Time	States	Time	States	Time	States	Time
big_bmw	31	10929	0.1	4825	0.11	9718	0.1	355	0.06	168	0.04	13160	0.15
fzelle	67	2392	0.24	27576	0.31	1309	0.2	1660	0.22	2293	0.24	2263	0.22
rhone_alps	35	1142	0.06	20214	0.19	16231	0.13	16086	0.11	5254	0.14	16986	0.14
tbed_ctct	84		157.87	21503	0.17		20.63	21503	0.17	22670	0.22		63.23
tbed_nocoll	84		81.62		571.92		106.39	143639	1.15		410.83		224.93
tbed_noderail	84		201.48		586.78		146.09	144789	1.19		467.51		223.76
verriegel4	65	28906	0.47	33960	0.55	628	0.13	21431	0.35	11203	0.27	33138	0.52
profisafe_i4	80	94	0.17	86	0.13	94	0.13	41	0.08	47	0.11	41	0.08
profisafe_i4_host	28	89	0.1	77	0.09	89	0.1	36	0.06	42	0.09	36	0.06
profisafe_i4_slave	14	5	0.02	9	0.03	5	0.03	5	0.02	5	0.03	5	0.03
profisafe_i5	88	94	0.15	86	0.23	94	0.15	41	0.1	47	0.13	41	0.09
profisafe_i5_host	28	89	0.11	77	0.11	89	0.12	36	0.07	42	0.1	36	0.07
profisafe_i6	94	94	0.17	86	0.19	94	0.18	41	0.12	47	0.15	41	0.11
profisafe_i6_host	28	89	0.13	77	0.12	89	0.14	36	0.08	42	0.11	36	0.08
profisafe_inclusion_i4host	78	554	0.34	104	0.11	48	0.08	62	0.09	323	0.2	118	0.09
profisafe_inclusion_o4host	84	554	0.34	104	0.12	48	0.09	62	0.09	323	0.22	118	0.1
profisafe_inclusion_o4slave	84	554	0.34	104	0.12	48	0.09	62	0.09	323	0.22	118	0.1
profisafe_o4	90	94	0.15	86	0.12	94	0.13	41	0.09	47	0.13	41	0.09
profisafe_o4_host	30	89	0.1	77	0.1	89	0.11	36	0.06	42	0.09	36	0.06
profisafe_o4_slave	16	5	0.03	9	0.04	5	0.03	5	0.04	5	0.03	5	0.03
profisafe_o5	99	94	0.15	86	0.21	94	0.16	41	0.11	47	0.14	41	0.11
profisafe_o5_host	30	89	0.15	77	0.11	89	0.12	36	0.07	42	0.1	36	0.07
profisafe_o6	106	94	0.21	86	0.19	94	0.19	41	0.13	47	0.16	41	0.13
profisafe_o6_host	30	89	0.14	77	0.13	89	0.14	36	0.08	42	0.11	36	0.08
ftechnik	36		45.95	235864	3.24		1033.98	2156487	10.49	31549956	158.24		259.57
rhone_tough	61		8.76		24.83		50.63		81.83		8.8		14.01
tbed_uncont	84		60.41		573.15		105.87		200.49		93.64		143.35

Model		Min Events		Min NewEvents		Min States		Min Transitions		One		RelMax Common	
Name	Aut	States	Time	States	Time	States	Time	States	Time	States	Time	States	Time
big_bmw	31	2058	0.07	168	0.04	168	0.05	168	0.04	10929	0.1	168	0.05
fzelle	67	62259	0.51	1586	0.24	71847	0.55	71041	0.54	2392	0.21	1544	0.22
rhone_alps	35	1142	0.06	842	0.05	1142	0.06	1142	0.06	1142	0.06	842	0.05
tbed_ctct	84		37.43		18.83		74.24		208.01		157.74	2060309	8.1
tbed_nocoll	84		188.52		189.94		109.15		170.97		81.33	893607	4.22
tbed_noderail	84		45.54		287.14		109.03		419.47		201.62		61.47
verriegel4	65	1665	0.17	587	0.13	587	0.13	587	0.13	28906	0.46	587	0.52
profisafe_i4	80	17	0.07	17	0.07	44	0.1	21	0.08	94	0.13	17	0.07
profisafe_i4_host	28	12	0.05	12	0.05	39	0.08	12	0.05	89	0.1	12	0.05
profisafe_i4_slave	14	5	0.03	5	0.03	5	0.03	9	0.03	5	0.03	5	0.03
profisafe_i5	88	21	0.09	17	0.08	44	0.12	21	0.1	94	0.14	17	0.09
profisafe_i5_host	28	12	0.06	12	0.06	39	0.09	12	0.05	89	0.11	12	0.06
profisafe_i6	94	21	0.11	17	0.1	44	0.14	21	0.11	94	0.16	17	0.1
profisafe_i6_host	28	12	0.07	12	0.07	39	0.1	12	0.07	89	0.13	12	0.07
profisafe_inclusion_i4host	78	19	0.07	87	0.09	38	0.09	33	0.09	554	0.31	29	0.08
profisafe_inclusion_o4host	84	15	0.07	87	0.1	38	0.09	33	0.09	554	0.32	29	0.09
profisafe_inclusion_o4slave	84	19	0.08	87	0.1	38	0.09	33	0.09	554	0.32	29	0.09
profisafe_o4	90	17	0.08	17	0.08	44	0.11	21	0.1	94	0.14	17	0.08
profisafe_o4_host	30	12	0.05	12	0.05	39	0.09	12	0.04	89	0.1	12	0.05
profisafe_o4_slave	16	5	0.04	5	0.03	5	0.03	9	0.04	5	0.03	5	0.04
profisafe_o5	99	17	0.1	17	0.1	44	0.14	21	0.1	94	0.16	17	0.1
profisafe_o5_host	30	12	0.06	12	0.06	39	0.09	12	0.05	89	0.12	12	0.06
profisafe_o6	106	17	0.11	17	0.11	44	0.16	21	0.12	94	0.18	17	0.11
profisafe_o6_host	30	12	0.07	12	0.07	39	0.11	12	0.07	89	0.14	12	0.07
ftechnik	36		416.33		677.16		629.97		908.02		45.84	1539941	8.58
rhone_tough	61		24.59		28.03		9.55		8.73		8.78		25.55
tbed_uncont	84		187.71		79.48		108.86		170.25		60.1		69.13

Table 5.4
CULLING CONTROLLABILITY CHECKER PREFERRING PLANTS,
SMALLEST CONTROLLER FIRST

Model		Culling checker, preferring plants, smallest controller first											
		All		Early NotAccept		Late NotAccept		MaxCommon Events		MaxCommon Uncontr		Max States	
Name	Aut	States	Time	States	Time	States	Time	States	Time	States	Time	States	Time
big_bmw	31	407	0.06	451	0.06	447	0.06	447	0.07	161	0.04	451	0.06
fzelle	67	6190	0.29	18506	0.54	8618	0.41	4976	0.28	5157	0.27	5200	0.27
rhone_alps	35	25320	0.15	615983	3.22	15715	0.12	3210	0.08	25306	0.15	3630	0.08
tbed_ctct	84		107.17	21503	0.17		36.9	21503	0.17	22670	0.23		115.22
tbed_nocoll	84		57.29		287.46		226.58		79.67		61.65		97.81
tbed_noderail	84		71.95		239.34		76.64		82.51		62.44		235.96
verriegel4	65	41175	0.57	21867	0.34	563	0.15	21797	0.34	11411	0.32	44791	0.6
profisafe_i4	80	54	0.11	48	0.11	44	0.1	24	0.08	54	0.11	54	0.11
profisafe_i4_host	28	49	0.09	39	0.08	39	0.08	19	0.05	49	0.09	49	0.09
profisafe_i4_slave	14	5	0.02	9	0.03	5	0.03	5	0.03	5	0.03	5	0.02
profisafe_i5	88	54	0.14	48	0.13	44	0.12	24	0.09	54	0.13	54	0.14
profisafe_i5_host	28	49	0.1	39	0.09	39	0.09	19	0.06	49	0.1	49	0.1
profisafe_i6	94	54	0.16	48	0.15	44	0.14	24	0.1	54	0.15	54	0.15
profisafe_i6_host	28	49	0.11	39	0.11	39	0.1	19	0.07	49	0.11	49	0.12
profisafe_inclusion_i4host	78	406	0.24	145	0.13	28	0.07	33	0.08	266	0.17	407	0.24
profisafe_inclusion_o4host	84	406	0.25	145	0.13	28	0.08	33	0.08	266	0.18	407	0.26
profisafe_inclusion_o4slave	84	406	0.25	145	0.13	28	0.08	33	0.08	266	0.18	407	0.26
profisafe_o4	90	54	0.12	48	0.12	44	0.11	24	0.08	54	0.12	54	0.12
profisafe_o4_host	30	49	0.09	39	0.08	39	0.08	19	0.05	49	0.09	49	0.09
profisafe_o4_slave	16	5	0.03	9	0.04	5	0.03	5	0.03	5	0.03	5	0.03
profisafe_o5	99	54	0.15	48	0.14	44	0.13	24	0.1	54	0.15	54	0.15
profisafe_o5_host	30	49	0.1	39	0.09	39	0.09	19	0.06	49	0.1	49	0.1
profisafe_o6	106	54	0.17	48	0.16	44	0.15	24	0.12	54	0.17	54	0.16
profisafe_o6_host	30	49	0.12	39	0.11	39	0.1	19	0.07	49	0.11	49	0.12
ftechnik	36		399.21		166.16		91.74		674.43		402.85		300.83
rhone_tough	61		9.34		10.67		17.82		8.52		9.4		17.13
tbed_uncont	84		57.34		288.05		227.27		79.83		61.85		98.18

Model		Min Events		Min NewEvents		Min States		Min Transitions		One		RelMax Common	
Name	Aut	States	Time	States	Time	States	Time	States	Time	States	Time	States	Time
big_bmw	31	161	0.04	161	0.05	161	0.05	161	0.04	407	0.07	161	0.05
fzelle	67	10431	0.38	5096	0.29	12422	0.41	13999	0.41	6190	0.27	6101	0.28
rhone_alps	35	25320	0.15	1108	0.05	429079	1.61	428133	1.58	25320	0.15	1108	0.06
tbed_ctct	84		53.61		40.52		51.94		44.35		107.12	2060309	7.92
tbed_nocoll	84		61.07		62.89		197.72		72.46		57.03		62.54
tbed_noderail	84		28.33		62.9		199.46		179.52		72.19		63.68
verriegel4	65	2471	0.23	751	0.17	751	0.18	751	0.19	41175	0.56	751	0.18
profisafe_i4	80	24	0.08	24	0.08	54	0.11	28	0.09	54	0.11	24	0.08
profisafe_i4_host	28	19	0.06	19	0.06	49	0.09	19	0.06	49	0.09	19	0.06
profisafe_i4_slave	14	5	0.03	5	0.03	5	0.03	9	0.03	5	0.03	5	0.03
profisafe_i5	88	28	0.1	24	0.09	54	0.14	28	0.1	54	0.13	24	0.09
profisafe_i5_host	28	19	0.06	19	0.06	49	0.1	19	0.06	49	0.1	19	0.06
profisafe_i6	94	28	0.12	24	0.11	54	0.15	28	0.12	54	0.15	24	0.1
profisafe_i6_host	28	19	0.07	19	0.07	49	0.12	19	0.07	49	0.11	19	0.07
profisafe_inclusion_i4host	78	26	0.08	94	0.1	48	0.09	43	0.09	406	0.24	36	0.08
profisafe_inclusion_o4host	84	22	0.07	94	0.1	48	0.1	43	0.1	406	0.25	36	0.09
profisafe_inclusion_o4slave	84	26	0.08	94	0.1	48	0.1	43	0.1	406	0.25	36	0.09
profisafe_o4	90	24	0.09	24	0.08	54	0.13	28	0.1	54	0.12	24	0.08
profisafe_o4_host	30	19	0.06	19	0.06	49	0.09	19	0.06	49	0.1	19	0.06
profisafe_o4_slave	16	5	0.03	5	0.03	5	0.03	9	0.04	5	0.04	5	0.03
profisafe_o5	99	24	0.11	24	0.1	54	0.15	28	0.12	54	0.14	24	0.1
profisafe_o5_host	30	19	0.06	19	0.06	49	0.1	19	0.06	49	0.1	19	0.07
profisafe_o6	106	24	0.12	24	0.12	54	0.17	28	0.13	54	0.16	24	0.12
profisafe_o6_host	30	19	0.07	19	0.07	49	0.12	19	0.07	49	0.11	19	0.07
ftechnik	36		327.32		167.67	125961955	758.24		293.29		399.58		168.26
rhone_tough	61		18.49		9.76		25.09		29.93		9.39		9.85
tbed_uncont	84		60.98		62.99		197.24		72.85		57.39		62.7

Table 5.5
CULLING CONTROLLABILITY CHECKER NOT PREFERRING PLANTS,
SMALLEST CONTROLLER FIRST

Model		Culling checker, not preferring plants, smallest controller first											
		All		Early NotAccept		Late NotAccept		MaxCommon Events		MaxCommon Uncontr		Max States	
Name	Aut	States	Time	States	Time	States	Time	States	Time	States	Time	States	Time
big_bmw	31	407	0.06	451	0.06	447	0.06	333	0.06	161	0.05	8213	0.1
fzelle	67	6190	0.3	17298	0.39	3556	0.3	1949	0.17	2006	0.18	2105	0.18
rhone_alps	35	25320	0.17	615983	3.23	15715	0.11	3308	0.07	6192	0.1	3992	0.07
tbed_ctct	84		107.49	21503	0.17		37.34	21503	0.17	22670	0.23		115.69
tbed_nocoll	84		57.58		408.99		35.58	268037	1.69	283815	2.1		142.88
tbed_noderail	84		72.21		410.83		252.11	272714	1.77	287397	2.17		56.62
verriegel4	65	41175	0.58	22395	0.34	563	0.16	22379	0.36	11411	0.32	39641	0.61
profisafe_i4	80	54	0.12	86	0.12	44	0.1	41	0.08	54	0.12	41	0.08
profisafe_i4_host	28	49	0.09	77	0.09	39	0.08	36	0.05	49	0.09	36	0.06
profisafe_i4_slave	14	5	0.02	9	0.03	5	0.03	5	0.03	5	0.03	5	0.03
profisafe_i5	88	54	0.14	86	0.15	44	0.12	41	0.1	54	0.14	41	0.1
profisafe_i5_host	28	49	0.1	77	0.1	39	0.09	36	0.06	49	0.11	36	0.06
profisafe_i6	94	54	0.16	86	0.16	44	0.14	41	0.11	54	0.16	41	0.11
profisafe_i6_host	28	49	0.2	77	0.11	39	0.1	36	0.07	49	0.11	36	0.07
profisafe_inclusion_i4host	78	406	0.23	104	0.11	28	0.07	62	0.07	266	0.15	118	0.23
profisafe_inclusion_o4host	84	406	0.25	104	0.22	28	0.18	62	0.18	266	0.17	118	0.09
profisafe_inclusion_o4slave	84	406	0.24	104	0.11	28	0.08	62	0.08	266	0.17	118	0.09
profisafe_o4	90	54	0.12	86	0.12	44	0.11	41	0.08	54	0.12	41	0.08
profisafe_o4_host	30	49	0.08	77	0.09	39	0.08	36	0.05	49	0.09	36	0.05
profisafe_o4_slave	16	5	0.03	9	0.04	5	0.03	5	0.03	5	0.03	5	0.03
profisafe_o5	99	54	0.16	86	0.16	44	0.13	41	0.11	54	0.15	41	0.11
profisafe_o5_host	30	49	0.11	77	0.11	39	0.1	36	0.06	49	0.11	36	0.07
profisafe_o6	106	54	0.18	86	0.2	44	0.15	41	0.12	54	0.16	41	0.12
profisafe_o6_host	30	49	0.11	77	0.11	39	0.13	36	0.14	49	0.11	36	0.07
ftechnik	36		401.65	105403	0.74	128	0.04	128	0.04	128	0.04	11940	0.11
rhone_tough	61		9.52		15.83		25.46		11.61		104		16.05
tbed_uncont	84		57.24		411.73		35.72		96.96		116.68		191.67

Model		Min Events		Min NewEvents		Min States		Min Transitions		One		RelMax Common	
Name	Aut	States	Time	States	Time	States	Time	States	Time	States	Time	States	Time
big_bmw	31	161	0.05	161	0.04	161	0.05	161	0.04	407	0.06	161	0.05
fzelle	67	10431	0.4	1960	0.19	21266	0.48	22843	0.47	6190	0.28	1991	0.17
rhone_alps	35	25320	0.16	1108	0.05	429079	1.61	428133	1.59	25320	0.15	1108	0.05
tbed_ctct	84		53.78		40.55		51.99		44.48		107.68	2060309	7.95
tbed_nocoll	84		127		109.12		182.56		126.99		57.13	471700	2.63
tbed_noderail	84		29.28		159.07		181.45		25.2		71.98	405748	2.3
verriegel4	65	2471	0.23	751	0.18	751	0.18	751	0.18	41175	0.56	751	0.18
profisafe_i4	80	24	0.08	24	0.08	54	0.12	28	0.09	54	0.11	24	0.08
profisafe_i4_host	28	19	0.06	19	0.06	49	0.1	19	0.06	49	0.09	19	0.05
profisafe_i4_slave	14	5	0.02	5	0.03	5	0.03	9	0.03	5	0.03	5	0.03
profisafe_i5	88	28	0.1	24	0.09	54	0.14	28	0.1	54	0.13	24	0.09
profisafe_i5_host	28	19	0.07	19	0.06	49	0.1	19	0.07	49	0.1	19	0.07
profisafe_i6	94	28	0.12	24	0.11	54	0.16	28	0.12	54	0.2	24	0.1
profisafe_i6_host	28	19	0.07	19	0.07	49	0.11	19	0.09	49	0.11	19	0.09
profisafe_inclusion_i4host	78	26	0.07	94	0.1	48	0.09	43	0.09	406	0.21	36	0.08
profisafe_inclusion_o4host	84	22	0.07	94	0.1	48	0.13	43	0.1	406	0.22	36	0.08
profisafe_inclusion_o4slave	84	26	0.08	94	0.11	48	0.1	43	0.1	406	0.22	36	0.08
profisafe_o4	90	24	0.08	24	0.08	54	0.12	28	0.16	54	0.11	24	0.08
profisafe_o4_host	30	19	0.05	19	0.05	49	0.13	19	0.05	49	0.08	19	0.05
profisafe_o4_slave	16	5	0.03	5	0.04	5	0.03	9	0.04	5	0.03	5	0.03
profisafe_o5	99	24	0.11	24	0.12	54	0.15	28	0.12	54	0.15	24	0.11
profisafe_o5_host	30	19	0.07	19	0.07	49	0.24	19	0.06	49	0.09	19	0.07
profisafe_o6	106	24	0.16	24	0.12	54	0.17	28	0.26	54	0.15	24	0.12
profisafe_o6_host	30	19	0.07	19	0.07	49	0.15	19	0.07	49	0.11	19	0.07
ftechnik	36		90.45	128	0.03		151.3		59.66		401.63	128	0.03
rhone_tough	61		21.67		79.52		25.05		40.62		9.42		10.83
tbed_uncont	84		129.08		110.11		181.71		127.64		58.42		88.84

Chapter 6

Projecting Checker

One of the limitations of the modular technique of controllability checking is that regardless of how well automata are chosen for the composition, it may be the case that the smallest subset of states which can be used to prove the model controllable may still have a synchronous product which is too large to handle.

In this chapter we outlined a method of automatically abstracting a model using projection [**?**, 9] into a much smaller model, which is controllable if and only if the original model is controllable and how we can use this method in a controllability checker to make more models solvable.

This chapter is broken up into several sections. Section 1 discusses the transformation which we designed to convert a model into a form which we can use projection on it. Section 2 discusses what exactly projection is and and how we can use it to simplify the automata in the model. Section 3 details two algorithms for iteratively using projection to simplify a model and how to choose which set of automata to project at each step. Section 4 details algorithm that was designed for converting a counterexample from a projected model into an equivalent counterexample for the original model. Section 5 discusses two controllability checker which were designed to use projection to help check controllability. Section 6 discusses areas where caching was used to improve performance. Finally Section 7 discusses the results achieved by the various algorithms.

6.1 Automata Transformation

To be able to use projection effectively, it is necessary to convert the standard controllability problem of asking whether uncontrollable events can happen at an inopportune time into the much simpler problem of simply whether or not certain bad events can happen. To do this, we transform the plant P and the controller C in the original model into a plant P' and a controller C' by adding new γ events into the model such that if we enumerate all the controllers in the model C_i and uncontrollable events υ_j, then the event $\gamma_{i,j}$ can occur only

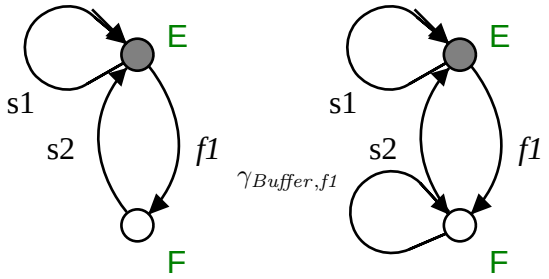

Figure 6.1: Transformation of Buffer

in a situation where υ_j is allowed by P but not allowed by C_i. Thus, the controllability equation is converted from

$$\forall t, j : t\upsilon_j \in L(P) \wedge t \in L(C) \rightarrow t\upsilon_j \in L(C) \tag{6.1}$$

into

$$\forall t, i, j : t\gamma_{i,j} \notin L(P' \| C') \tag{6.2}$$

Every event $\gamma_{i,j}$ introduced must have the property that it is not mentioned in the alphabet of any automaton A in the original model and thus is allowable in every state in the original model.

$$\forall A, t, i, j : t \in L(A) \rightarrow t\gamma_{i,j} \in L(A) \tag{6.3}$$

Next we describe how controllers are transformed from C_i into C_i'. In each controller C_i for every uncontrollable event υ_j and for all states s in C_i, if there is no outgoing transition for υ_j in s, we add the transition $(s, \gamma_{i,j}, s)$. Thus, for every trace t through the controller C_i, the event $\gamma_{i,j}$ can occur in the modified controller C_i' if and only if υ_j can't occur in C_i.

$$\forall t, i, j : t \in L(C_i) \wedge t\upsilon_j \notin L(C_i) \leftrightarrow t\gamma_{i,j} \in L(C_i') \tag{6.4}$$

Also we never add a γ event related to any other controller. Therefore no controller can block another controller's γ event.

$$\forall t, i, j : t \in L(C_i') \rightarrow \forall k \neq i : t\gamma_{k,j} \in L(C_i') \tag{6.5}$$

Now we can also describe how a plant P_l is converted into its modified counterpart P_l'. In each plant P_l for every uncontrollable event υ_j and for all states s where there exists an outgoing transition for υ_j, we add the transition $(s, \gamma_{i,j}, s)$ for every possible i. Therefore whenever the plant P_l would allow the event υ_j it will also allow any γ event related to υ_j.

$$\forall l, t, i, j : t\upsilon_j \in L(P_l) \leftrightarrow t\gamma_{i,j} \in L(P_l') \tag{6.6}$$

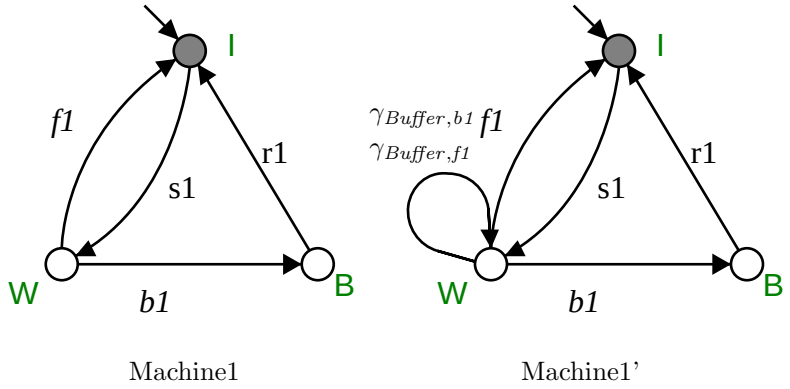

Figure 6.2: Transformation of Machine1

Further only selfloops with γ events are ever added to any automaton A in the model when converting it into A', and for each original automaton A in the model no γ event is ever referred to in its alphabet. Therefore rather than adding selfloops on states, we are actually deleting the implicit selfloops which were present on all the other states in A as described in "Running in Parallel" in Chapter 2.3. Therefore rather than adding to the language of A when converting it into A' we actually restrict its language.

$$\forall a : L(A') \subseteq L(A) \tag{6.7}$$

Furthermore, we can infer from (6.6) that $t\gamma_{i,j}$ is in $L(P')$ if and only if tv_j is in $L(P)$. This is because $t\gamma_{i,j}$ is in the language of every modified plant automaton $L(P'_l)$ if and only if tv_j is in the language of each of the original plant automata $L(P_l)$.

$$\forall t, i, j : tv_j \in L(P) \leftrightarrow t\gamma_{i,j} \in L(P') \tag{6.8}$$

Similarly we can infer from (6.4) that $t\gamma_{i,j}$ is in $L(C')$ if and only if t is in $L(C)$ and tv_j is not in $L(C)$. This is because if t is in $L(C)$ then t is in the language of all C_k and tv_j is not in $L(C)$ only if there is some i for which tv_j is not in $L(C_i)$. Therefore from (6.4), $t\gamma_{i,j}$ must be in $L(C'_i)$, and using the fact that t is in all $L(C_k)$, we can use (6.5) to infer that all other languages $L(C'_k)$ also contain $t\gamma_{i,j}$.

$$\forall t, j : t \in L(C) \wedge tv_j \notin L(C) \leftrightarrow \exists i : t\gamma_{i,j} \in L(C') \tag{6.9}$$

To be able to use the modified version of the controllability problem (6.2) instead of the original controllability problem (6.1) we now prove that, if this transformation is used, they are equivalent.

Proposition 1. *Let P be a plant model and let C be a controller model. Then (6.1) is equivalent to (6.2)*

Proof. Firstly let us prove that if (6.1) is true then (6.2) must also be true.

Let us assume that (6.2 is false thus our claim does not hold. Therefore there must exist a trace $t\gamma_{i,j}$ such that t contains no γ for which

$$t\gamma_{i,j} \in L(P'\|C')$$

Then $t\gamma_{i,j}$ is an element of both $L(P')$ and $L(C')$.

$$t\gamma_{i,j} \in L(P') \wedge t\gamma_{i,j} \in L(C') \tag{6.10}$$

Then from the fact $t\gamma_{i,j}$ is in $L(P')$ we can use (6.8) to infer tv_j is in $L(P)$.

$$tv_j \in L(P) \tag{6.11}$$

Also seeing as how $t\gamma_{i,j}$ is in the language $L(C')$, t must also be in the language $L(C')$. Therefore we can use (6.7) to infer that t is in $L(C)$

$$t \in L(C) \tag{6.12}$$

Then from (6.11) and (6.12) we can use (6.1) to infer that tv_j is in $L(C)$

$$tv_j \in L(C) \tag{6.13}$$

However we can also infer that tv_j is not in $L(C)$ using the fact that $t\gamma_{i,j}$ is in the language $L(C')$ and (6.9)

$$tv_j \notin L(C) \tag{6.14}$$

(6.13) and (6.14) clearly contradict each other. Therefore, if (6.1) is true then (6.2) is true also.

Now we must also prove that if (6.1) is false then (6.2) is also false.

To prove this we make the observation that (6.1) is only false if there exists a counterexample which shows that C is not controllable with respect to P.

$$\exists t, j : tv_j \in L(P) \wedge t \in L(C) \wedge tv_j \notin L(C) \tag{6.15}$$

We can then observe from (6.9) that for such a counterexample, there would have to exist the a trace $t\gamma_{i,j}$ which is an element of $L(C'_i)$ for some i.

$$\exists i : t\gamma_{i,j} \in L(C'_i) \tag{6.16}$$

Since $tv_j \in L(P)$ we can infer from (6.8) that there must be a trace $t\gamma_{i,j}$ in $L(P')$ for all i.

$$\forall i : t\gamma_{i,j} \in L(P') \tag{6.17}$$

It follows that $t\gamma_{i,j}$ is in both $L(P')$ and $L(C')$, and from that $t\gamma_{i,j}$ is also in $L(C'\|P')$.

$$t\gamma_{i,j} \in L(C'\|P') \tag{6.18}$$

This proves that if (6.1) is false then (6.2) is also false. □

6.2 Projection

Projection can be used to simplify an automaton by removing certain events [**?**, 9]. When we decide to project events out of an automaton, we first select the set of events Σ' from the alphabet of the automaton Σ to keep in the automaton. We then go through all the transitions in the automaton and label any transition labelled with an event not contained in Σ' with the non-event τ. For our purposes, a transition labelled with τ does not require any event to occur to travel along it. This modifies the language of the automaton such that for all traces t which are in the language of the original automaton, the trace $\pi_{\Sigma'}(t)$ is in the language of the projected automaton. Here, $\pi_{\Sigma'}(t)$ is the trace t with all events not in Σ' removed.

Thus for any trace t in the language of an automaton projected with respect to Σ', $\pi_{\Sigma'}(L')$, there must also exist a trace s such that s is in the language L of the automaton before projection, and if s has all the events in Σ' projected out of it, it will be equal to t.

$$\forall t : t \in \pi_{\Sigma'}(L') \rightarrow \exists s : t = \pi_{\Sigma'}(s) \wedge s \in L \tag{6.19}$$

Given a model made up of several automata we can select one of the model's automata to project with respect to Σ' preserving (6.3) under two conditions. Firstly all γ events must be in the set Σ', as removing a γ event could cause uncontrollable behaviour to be hidden. Secondly, we may only leave an event outside the set Σ', if that event is only mentioned in the alphabet of the automaton we have chosen to project the events out of. This is so that we don't introduce traces into the system which could otherwise be blocked by another automaton in the model. Unfortunately most events in most models occur in more than one automaton. Therefore, by the second condition we cannot remove them from any automata in the model. However, if an event occurs in only a few of the automata in the model, it is possible to replace those few automata in the model with their synchronous product, and then project the event out of that. For example if a model has 4 automata A, B, C, and D, and the event β only occurs in A and C then, if we want to project out the event β, we can compose A and C together so that the model is now made up of $A\|C$, B, and D. Then we can project β out of $A\|C$.

Once events are projected out of an automaton as described above the automaton is non-deterministic. Therefore we must determinise (convert into a deterministic automaton) the projected automaton using subset construction [8]. In most cases, the resultant automaton will be smaller than the original, in the worst case the resultant automaton could have exponentially more states than the original. This is rare however and in practice not much of an issue as if during subset construction we notice that the new automaton we are producing is getting to large we can simply terminate and try a different set of automata to compose and project. After this as an optional extra process we can attempt to minimise the automaton further by running the minimisation algorithm [9] on this automaton. This minimisation algorithm was chosen as it had the worst

case time complexity of $O(n \log(n))$ in the number of states.

6.3 Iterative Projection

If it is required to check the controllability of a model which is made of several hundred automata, the ability to reduce a single automaton is unlikely to be enough to have a serious impact on our ability to check it. Fortunately, it is often possible to continue using projection on further automata in the model.

Consider for example that we have a model which consists of the automata which follow.

A With the alphabet a, b

B With the alphabet a, b, e

C With the alphabet c, d, f

D With the alphabet b, c, e

E With the alphabet d, e, f

As automata A and B are the only automata with the event a contained in their alphabet, we can compose these two automata together and then project out the event a. After doing this we would have a model made up of the automata $\pi_{\Sigma'_1}(A \| B), C, D, E$ where $\Sigma'_1 = \{b, e\}$. Similarly automata C and E are the only ones whose alphabet contains d and f so we can compose and project these two automata so that we have the model $\pi_{\Sigma'_1}(A \| B), \pi_{\Sigma'_2}(C \| E), D$ where $\Sigma'_2 = \{c, e\}$. Now, we can also notice that $\pi_{\Sigma'_1}(A \| B)$ and D are the only two automata which contain b thus we can compose and project these two to get $\pi_{\Sigma'_3}(\pi_{\Sigma'_1}(A \| B) \| D), \pi_{\Sigma'_2}(C \| E)$ where $\Sigma'_3 = \{c, e\}$. It should be noted that the series of projections given are not the only ones which could have been taken. For example on the previous step the automaton D could have been composed with $\pi_{\Sigma'_2}(C \| E)$ instead to remove the event c.

An algorithm for iteratively projecting out events in a model is given in Figure 6.3.

In this algorithm, we simply find every set of automata for which we can project out an event. Then we take the set of automata which we believe will give us the smallest projection. We evaluate a set of automata by the multiplying the number of states in each automaton together then raising that value to the power of the number of events which appear in at least one of the automata in the set but not in every automaton, and then say whichever set of automata has the smallest value of this is best. The rationale behind this metric is that the greater the number of states in each automaton, the greater the potential size of the final synchronous product, and that the more events which aren't common to each automata in the set the less related to one another the automata in the set are. Once we have selected a set of automata we generate its synchronous product, then calculate the smallest possible set of events for

A is the set of automata in the model which we are considering to project events out of. Γ is the set of γ events introduced in Section 6.1. S is a set of sets of automata. Every set of automata contained in S represents a minimal set of automata to cover at least one event.

1. Set *attempts* to equal 0.
2. For all events σ in Σ except those in Γ find the the set of all automata in A whose alphabet contains σ and add it to the set S, unless σ occurs in all automata in A.
3. Sort S such that the set of automata which is likely to be smallest when projected is listed first.
4. If S is empty then return A.
5. Remove the first set s from S and calculate its synchronous product *prod*, unless it is greater than $maxstates * 10$, in which case go to 11.
6. Let the set of kept events Σ' be equal to Γ plus all events which occur in any automata not in s.
7. Create the automaton $\pi_{\Sigma'}(prod)$
8. Set *det* to equal the determinised version of $\pi_{\Sigma'}(prod)$ unless the number of states exceeds *maxstates* in which case go to 11.
9. Find the minimal version *min* of *det* using a minimisation algorithm.
10. Remove all elements of s from A, then add *min* to A and go to 1.
11. Increment *attempts*, then check to see whether *attempts* is greater than or equal to *maxattempts*, in which case return A. Otherwise go to 3.

Figure 6.3: Non-Exhaustive Iterative Projection

Σ'. Then we proceed to project and determinise the synchronous product with respect to Σ'. Finally we proceed to minimise using a minimisation algorithm then replace the automata we projected with the minimise automata and start the process again. However, during either the synchronous product step or the determinisation step, we may encounter an automaton which is larger than we are willing to deal with, in which case we try the next best set of automata, and so on. The stopping conditions are that we have no sets of automata left to look at, or we have attempted to generate automata which are too large too many times.

In some cases simply taking the first projection which we were capable of projecting successfully is not good enough. Thus the modified algorithm in Figure 6.4 can also be used. In this algorithm instead of guessing which set of

automata will give us the smallest automaton after projection and minimisation we instead compose project and minimise every automaton within a certain range and choose the one which gave the smallest resulting automata at the end.

A is the set of automata in the model which we are considering to project events out of. Γ is the set of γ events introduced in Section 6.1. S is a set of sets of automata. Every set of automata contained in S represents a minimal set of automata to cover at least one event.

1. Set *smallest* and *smallset* to *null*.
2. For all events σ in Σ except those in Γ find the the set of all automata in A whose alphabet contains σ and add it to the set S, unless σ occurs in all automata in A, or the set contains more automata than *maxautomata*.
3. If S is empty then go to 10. Otherwise remove the first set s from S and calculate its synchronous product *prod*, unless it is greater than $maxstates * 10$ in which case go to 3.
4. Set the set of kept events Σ' to equal Γ plus all events which occur in any automata not in s.
5. Create the automaton $\pi_{\Sigma'}(prod)$.
6. Set *det* to equal the determinised version of $\pi_{\Sigma'}(prod)$, unless the number of states exceeds *maxstates* in which case go to 3.
7. Find the minimal version *min* of *det* using a minimisation function.
8. If *min* has less states than *smallest* or *smallest* is *null* replace *smallest* with *min* and *smallset* with s.
9. Go to 3.
10. Remove all elements of *smallset* from A then add *smallest* to A and go to 1.

Figure 6.4: Exhaustive Iterative Projection

Because of the differences between these two algorithms we call them Non-Exhaustive Iterative Projection and Exhaustive iterative projection respectively.

6.4 Extracting Traces

If there exists a trace which contains an event in Γ in the model before projection, there must also exist a trace in the model after projection which contains Γ. Unfortunately the trace found in the projected model may not be a proper counterexample for the original system. Fortunately using (6.19) we know that,

for any trace t through a projected automaton, there must also exist a trace s through the original automaton such that $t = \pi_{\Sigma'}(s)$. Therefore we can attempt to insert events not in the set Σ' into the trace t in such a way as to find a trace s which is accepted by the original automaton. It should be pointed out, that seeing as how any events not in Σ' do not occur in any other automata, we can insert them anywhere into the trace without worrying that it might cause another automaton to reject the trace. Figure 6.5 outlines an algorithm for finding such a trace. This algorithm basically consists of a breadth first search through the automaton to find a trace which will both be accepted by the automaton and is a reverse projection of t. To do this we keep a queue of tuples of a state we have reached, the number of steps through the original trace t we have gotten, and the events we followed to get there. To start with the queue contains a tuple consisting of the initial state, 0 and the empty trace. When we look through the tuples in this queue we consider whether we can move from the state in the tuple to another state in the automaton using either the current event in t or any one of the events not contained in Σ' and if we can we add a tuple for the new state we can get to, how many events in the original trace we have consumed, and the trace to get to this new state on to the end of the queue. It should be noted that we only ever add a tuple to our queue if we haven't already added a tuple with the same state and index into the old trace, this is because finding a state from which we can consume the next event in t from a given state is in no way dependant on the trace followed to get to that state. To save time on string copying instead of creating a new string of events s for each node we can just represent s as the event we are adding on to the end of our string plus a pointer to the old string. Because we only look at a node if we haven't already explored one with the same state and depth through t. this algorithm has worst case complexity of $O(n * |t| * e)$. where n is the number of states in A, $|t|$ is the length of the trace t, and e is the number of events not in Σ'.

Now if we have a counterexample for a model which has been iteratively projected, we can use the algorithm in Figure 6.6 to find a correct counterexample. Here we just use the algorithm in Figure 6.5 to find the trace before projection for the last automaton we projected using either the algorithm in Figure 6.3 or 6.4. Then we continue to use the new trace with the second to last automaton and so forth. This algorithm of course requires that we remember what automata we projected with respect to which events.

6.5 Projecting Checker

Now that we have actually defined what projection is and how to transform the automata in the model to take advantage of it, we can go into how we use these facts in an algorithm for checking the controllability of a system. Figure 6.7 is a description of the Modular Projecting Controllability algorithm. We can notice that the algorithm is very similar to the algorithm in Figure 3.1 for modular controllability checking. The main difference being simply that before we check

Let t be a counterexample found in the model made of the events $\sigma_0, \sigma_1 \ldots \sigma_{n-1}$ where n is the length of the trace t, also let A be the original automaton before it was projected let Q be a queue of tuples of states in A, length through t, and built-up trace, and let S be a set of pairs of state and length through t, and Σ' be the set of kept events.

1. Add the tuple $(initialstate(A), 0, [])$ to Q and $(initialstate(A), 0)$ to S.
2. Remove the first tuple $(state, i, s)$ in Q.
3. If $i = n$ return s.
4. If there is an outgoing transition from *state* labelled with σ_i to *next* add $(next, i+1, s\sigma_i)$ to Q and $(next, i+1)$ to S unless $(next, i+1) \in S$
5. For every event α not in Σ', if there is an outgoing transition from *state* labelled with α to *next*, then add $(next, i, s\alpha)$ to Q and $(next, i)$ to S unless $(next, i) \in S$.
6. Go to 2.

Figure 6.5: Find Trace Algorithm

the controllability of S we now reduce the size of S using projection first and that because of this whenever we get a counterexample we must also convert it back into a trace for the original system.

Once again, the methods of choosing automata in 3 and 6 are the same as those already discussed above in Chapter 3.

An alternative algorithm is to simply use iterative projection on the entire model to begin with and then use a regular modular controllability checker on the results. Then if the system is found to be not controllable, the counterexample is converted back.

6.6 Caching

If we look at the algorithm for iterative projection in Figure 6.4, as we have to project every set of automata in the set S and we do this for every iteration of the algorithm we will find that on each subsequent iteration we will often have to compute a projection which we have already calculated on previous iterations. Clearly, it is wasteful for us to do work which we have already done. Thus, when we do a projection we cache the results remembering that this set of automata when projected with respect to this set of events results in this minimise automaton, or that its resultant automaton will exceed the state limit. Then, whenever we project a set of automata we can check the cache to see if we have already attempted this projection before, and if we have, quickly retrieve the resultant automaton or realise that it is bigger than we are willing to look

Let $A_1, \ldots, A_n$ be the automata simplified by iterative projection, and let Σ'_i be the set of events we projected A_i with respect to, and let t be the projected trace.

1. set i to equal n
2. If i equals 0 return t
3. find the trace s using the algorithm in Figure 6.5 with A_i and Σ'_i as input.
4. set the new value of t to s, and decrement i, then go to 2

Figure 6.6: Iterative Find Trace Algorithm

at.

Furthermore, if we look at the controllability algorithm outlined in Figure 6.7 we will have to run one of the iterative projection algorithms on every subsequent composition S. Seeing as how most of the time every subsequent composition S is the same as the previous composition just with an extra automaton, it also follows that it is likely that when we run the iterative projection algorithms on any composition S, that at least some of the projections which will have to be carried out on S, will have already been done for the previous composition. Thus, when using either iterative projection algorithm, we can cache the results of the projections we carried out on the previous composition to help when projecting the current composition. To save memory, however, it is a good idea that, after we finish running the iterative projection algorithm on a composition, to remove from the cache any stored projections which weren't looked at when iteratively projecting that composition.

6.7 Results

The algorithms discussed in the previous section have been run to gather data as to how well the perform under various configurations. For all tables in this section, when deciding which controller to prove first the largest is always chosen to save time when gathering data and because the trend in previous results suggested that no interesting results would be found by looking at results for proving the smallest controller first as well. Further seeing as how all controllers have been converted into plants into the transformation step in makes no difference whether we choose the prefer plant mode of the heuristics or the no preference mode. Also when dealing with non-exhaustive iterative projection the maximum number of projections we attempt before we give up is always set to two, and for dealing with exhaustive iterative projection the maximum number of automata composed is always set to four.

It should be noted that the number of states which had to be explored by the checker in the tables is not as good an indicator of performance as in the

C' equals the set of modified Controller automata in the model and P' the set of modified Plant automata, and S is the set of composed automata.

1. Set the set S as being empty.
2. If C' is empty, the model has been proven controllable. Otherwise take an automaton C'_i from the set C' and add it to S. Also create a property p_i with only a single state which specifies that no γ event related to C_i can occur.
3. Use either algorithm in Section 6.3 to convert S into S'
4. Check controllability of S' with respect to p_i using the monolithic method. consider the automaton p_i as being a controller and all other automata as plants.
5. If no counterexample for S' was found go to 9. Otherwise set t' to be the counterexample found by the controllability check.
6. Use the algorithm in Section 6.6 to convert t' into t.
7. Set the set N to contain all automata in P and C which would not accept the counterexample t.
8. If N is empty then the model has been proven not controllable, and t represents a counterexample in the system. Otherwise pick a subset of N to add to S then go to 3.
9. For all elements of S if they are also an element of C, remove them from C and add them to P. Then go to 2.

Figure 6.7: Modular Projecting Controllability Checker Algorithm

previous sections. This is because, for the other checkers the lion's share of the work was performed while creating the synchronous product, whereas for the projecting checker it is quite possible that the majority of the work could be spent projecting, determinising, or minimising the automata.

Again it appears that all the heuristics are comparable to one another in their overall effectiveness. When looking at the performance of the algorithm used to produce Table 6.3 we can see that it has very good performance although it takes slightly longer than the standard modular controllability checker for the smaller problem. For example, where the standard modular controllability checker took 0.05 seconds to prove big_bmw the projecting checker took 0.18 seconds. However, for the larger models which the standard checker has trouble solving, the projecting checker seems to be able to solve faster and furthermore is capable of solving most of the models in the table regardless of which heuristic it is using. That said, for the profisafe series of models there are certain cases in which the projecting checker algorithm takes much longer than it should.

C' equals the set of modified Controller automata in the model, and P' the set of modified Plant automata, and S is the set of automata specifying events in Γ can't happen.

1. Set the set S as being empty.
2. For every automaton C'_i in C' add an automaton with only a single state which specifies that no γ event related to C_i can occur, to the set S.
3. Use either algorithm in Section 6.3 to convert the union of the sets P' and C' into *projected*.
4. Run the modular controllability algorithm outlined in Figure 3.1 using *projected* as the set of plant automata and S as the set of controllers.
5. If no counterexample was found, the model has been proven controllable. Otherwise set t' to be the counterexample found by the controllability check.
6. Use the algorithm in Section 6.6 to convert t' into t.
7. Return t as the counterexample to the system.

Figure 6.8: Modular Controllability Checker using Projection as a Pre-Process

Then looking at Table 6.4, we can see that for the modular projecting checker algorithm when used in conjunction with the exhaustive iterative projection algorithm, it doesn't seem to give any better results. It only seems that in some case the algorithm takes far longer to complete, suggesting that it should only really be necessary to use this algorithm when attempting to solve models which are unsolvable using the non-exhaustive iterative projection algorithm.

Next, we look at using non-exhaustive iterative projection as a pre-process before giving the problem to a standard modular checker as in Table 6.5. This seems to either work better than the modular projecting checker or otherwise work a lot worse. An example of when it works a lot better than the modular projecting checker is when solving tbed_nocoll and tbed_noderail with the Early-NotAccept heuristic here the modular projecting checker took roughly 3 minutes for both of them whereas the pre-process checker took 10 seconds. However, it was incapable of solving rhone_tough with any heuristic.

Further, we go on to Tables 6.6 and 6.7. These tables show results for solving controllability problems for both the algorithm modular projection algorithm and the pre-process algorithm. Both use the non-exhaustive iterative projection algorithm with varying values for the maximum number of states projected. In both cases the MaxCommonEvents heuristic is used. Only the tbed series of models and rhone_tough are considered in these tables as these are the models which require the most effort. From looking at these tables, we see that generally as we increase maxstates the time required to solve a model increases although

for Table 6.6 we see that for rhone_tough it is in fact impossible to solve this problem with the lower state limits for projection and for tbed_uncont while it is possible to be solved with a state limit of 100 it is in fact faster to solve with a state limit of 200. This suggests that there is a point at which the benefits of minimising automata using projection are outweighed by its extra overhead of projecting automata and determinising them. Also when the time taken seems to remain static it can probably be put down to the fact that projecting automata that are any larger than a certain maximum number of states never comes up.

Furthermore in Tables 6.8 and 6.9 we see results for using iterative projection to simplify a model before giving it to a standard monolithic controllability checker. The first table used standard iterative projection whereas the second used exhaustive iterative projection. It should be noted that whereas the measure of states in the previous tables takes into account those states explored in the projection steps, these tables only take into account the states explored in the final exploration of the synchronous product. This is so that we can get an idea of just how much the total synchronous product has been reduced by the projection. The results are surprisingly good with it being possible to reduce many models to the point where they possess one state and no transitions at all. This is quite impressive given that all of these models are so large as to be impossible to be solved using a regular controllability checker and thus have synchronous products which are at least have a total of ten million states most having significantly more. In the case of rhone_tough it wasn't even possible to solve it using modular or any other method for proving controllability for that matter. This also somewhat showcases the merits of the exhaustive method of iterative projection, where the standard model proved incapable of reducing the state space of certain models below two million states such as tbed_ctct and many of the profisafe models. Exhaustive was capable of reducing many of these automata to a reasonable size, or even to the point where they contained no uncontrollable events using a state limit of just 100 or 200. That said, in general if it was capable of solving the problem once again non-exhaustive ran faster than exhaustive. Figures 6.9 to 6.12 show charts of both the states in simplified tbed_ctct and rhone_tough for the two methods and the time to solve them. They both clearly show diminishing reduction in size for in the simplified model as max states increases. Also of note is the sharp increase in the time required to solve a model for exhaustive iterative projection as max states increases suggesting that when using this method special care should be taken to use a small number of for max states.

It should be noted that both profisafe_i4 and profisafe_o4 in addition to their controllability problems also have language inclusion problems. It turns out that these problems are in fact much harder to prove than there controllability problems, and in fact are very difficult to solve even using projection. Actually, while profisafe_i4 was capable of being solved by the projecting checker after roughly 10 minutes, a configuration capable of solving profisafe_o4 has not been found.

Table 6.1
MODULAR PROJECTING LANGUAGE INCLUSION, NON-EXHAUSTIVE,
MAX PROJECTION 1000

Model		Modular projecting language inclusion, max projection 1000											
		All		Early NotAccept		Late NotAccept		MaxCommon Events		Max States		Min Events	
Name	Aut	States	Time	States	Time	States	Time	States	Time	States	Time	States	Time
profisafe_i4_slave	15	1695	3.01	6194	2.42	4982	5.18	8656	6.03	15495	7.58	2063	1.64
profisafe_o4_slave	17	2330	7.26	6021	3.22	38609	11.09	4850	4.78	15275	9.49	14868	8.8
big_bmw	32	926	0.46	89	0.14	114	0.12	73	0.12	118	0.12	92	0.15
ftechnik	37	4532	5.99	2529	3.81	16498	26.98	13162	17.48	899	8.46	2743	6.21
tbed_nocoll	85	13490	42.6	343604	296.92	76	14.74	106	1.99	141	11.51	74	82.07
tbed_noderail	85	11564	18.38	30887	86.12	5893	140.89	574	26.57	743	74.88	12042	145.09
verriegel4	66	2812	0.99	174	0.07	200	0.09	330	0.15	175	0.07	201	0.1
profisafe_i4_host	29	33249	4.74	49386	45.2	3931	11.19	11697	11.2	29627	12.91	35044	11.94
profisafe_o4_host	31	17870	8.14	33805	13.22	9552	5.73	2874	7.04	25404	13.44	19681	24.79
profisafe_i5_host	29	39108	6.09	24482	22.84	4958	21.41	8787	16.42	17257	58.44	46586	24.76
profisafe_o5_host	31	20470	8.67	47540	16.22	4801	9.5	3632	12.89	11080	17.35	54604	23.65
profisafe_i6_host	29	44908	7.59	14296	52.6	5069	22.2	15970	28.04	29267	50.14	90160	36.19
profisafe_o6_host	31	23102	10.11	97728	63.09	15015	88.12	20261	25.2	14570	44.48	85031	25.2

Model		Min NewEvents		Min States		Min Transitions		One		RelMax Common	
Name	Aut	States	Time	States	Time	States	Time	States	Time	States	Time
profisafe_i4_slave	15	6674	4.28	1495	1.64	2931	2.09	2075	2.24	1232	5.7
profisafe_o4_slave	17	8124	10.43	27916	6.21	6227	2.8	15067	9.11	16552	7.13
big_bmw	32	73	0.09	68	0.07	92	0.12	114	0.12	73	0.08
ftechnik	37	3110	10.62	5299	16.74	3623	5.39	2867	4.71	4365	10.73
tbed_nocoll	85	88	21.79	158	83.78	74	171.02	1239	87.95	99	11.29
tbed_noderail	85	14713	72.34	9595	50.06	146540	97.5	22861	154.69	2543	30.98
verriegel4	66	202	0.1	184	0.08	201	0.1	200	0.09	179	0.07
profisafe_i4_host	29	8295	18.8	5126	7.87	24567	45.8	51868	11.98	12818	12.9
profisafe_o4_host	31	3521	9.18	3685	13.29	16954	23.34	52057	11.44	13979	9.97
profisafe_i5_host	29	7529	15.02	14115	28.78	41712	15.7	28415	76.19	2846	22.06
profisafe_o5_host	31	4529	7.15	4814	9.88	78219	42.82	15792	19.33	4507	7.14
profisafe_i6_host	29	15607	33.67	5161	22.3	55752	19.64	20969	80.42	2562	12.39
profisafe_o6_host	31	5108	18.7	10775	25.55	81239	101.16	19476	50.52	3033	14.91

Table 6.2
MODULAR LANGUAGE INCLUSION USING NON-EXHAUSTIVE
PROJECTION AS PRE-PROCESS, MAX PROJECTION 1000

Model		Modular language inclusion, non-exhaustive projection as pre-process, max projection 1000											
		All		Early NotAccept		Late NotAccept		MaxCommon Events		Max States		Min Events	
Name	Aut	States	Time	States	Time	States	Time	States	Time	States	Time	States	Time
profisafe_i4_slave	15	1699	3.3	1699	2.68	1699	2.36	1699	2.32	1699	2.25	1699	2.34
profisafe_o4_slave	17	3548	4.27	3548	4.32	3548	4.2	3548	4.32	3548	4.37	3548	4.45
big_bmw	32	1498	0.53	1498	0.54	1498	0.54	1498	0.54	1498	0.71	1498	0.52
ftechnik	37	91260	11.91	91713	13.52	207322	15.7	164277	14.78	96446	14.15	116379	13.46
tbed_nocoll	85	631197	68.93	1752235	79.95		55.34	599089	67.1		64.37		62.54
tbed_noderail	85	305898	30.44	470839	33.71		31.97	377208	31.81	401580	33.14		33.11
verriegel4	66	6062	1.27	6063	1.39	6062	1.24	6062	1.4	6062	1.23	6063	1.4
profisafe_i4_host	29	6723	10.97	6723	10.16	6723	9.82	6723	10.01	6723	9.77	6723	9.99
profisafe_o4_host	31	7971	11.75	7971	11.26	7971	11.09	7971	11.07	7971	11.34	7971	11.26
profisafe_i5_host	29	7098	16.34	7098	16.24	7098	16.6	7098	16.76	7098	16.85	7098	16.89
profisafe_o5_host	31	8074	14.18	8074	13.98	8074	13.58	8074	13.6	8074	13.58	8074	13.53
profisafe_i6_host	29	7353	27.58	7353	27.3	7353	27.6	7353	27.67	7353	27.74	7353	27.51
profisafe_o6_host	31	8209	16.79	8209	16.63	8209	16.58	8209	16.46	8209	16.34	8209	16.49

Model		Min NewEvents		Min States		Min Transitions		One		RelMax Common	
Name	Aut	States	Time	States	Time	States	Time	States	Time	States	Time
profisafe_i4_slave	15	1699	2.79	1699	2.42	1699	2.43	1699	2.7	1699	2.51
profisafe_o4_slave	17	3548	4.61	3548	4.68	3548	4.71	3548	4.52	3548	4.52
big_bmw	32	1498	0.54	1498	0.54	1498	0.53	1498	0.53	1498	0.53
ftechnik	37	244026	14.06	105113	13.11	113521	13.43	166371	15.07	212045	14.49
tbed_nocoll	85	4672136	86.34		47.18		46.57	5058519	86.88	2394067	75.51
tbed_noderail	85		32.73		24.43		24.59		32.63	2088487	38.1
verriegel4	66	6063	1.26	6063	1.4	6063	1.24	6062	1.39	6062	1.24
profisafe_i4_host	29	6723	10.04	6723	9.97	6723	9.93	6723	9.93	6723	9.95
profisafe_o4_host	31	7971	10.78	7971	10.76	7971	11.07	7971	11.24	7971	10.73
profisafe_i5_host	29	7098	16.6	7098	16.58	7098	16.61	7098	16.56	7098	16.6
profisafe_o5_host	31	8074	14	8074	13.51	8074	13.48	8074	13.44	8074	13.43
profisafe_i6_host	29	7353	27.52	7353	27.5	7353	27.49	7353	27.47	7353	27.19
profisafe_o6_host	31	8209	16.36	8209	16.53	8209	16.38	8209	16.54	8209	16.35

Table 6.3
MODULAR PROJECTING CONTROLLABILITY, NON-EXHAUSTIVE, MAX PROJECTION 1000

Model		Modular projecting controllability, non-exhaustive, max projection 1000											
		All		Early NotAccept		Late NotAccept		MaxCommon Events		MaxCommon Uncontr		Max States	
Name	Aut	States	Time	States	Time	States	Time	States	Time	States	Time	States	Time
big_bmw	31	269	0.32	328	0.2	314	0.18	323	0.18	264	0.15	314	0.17
fzelle	67	1090	0.29	744	0.24	700	0.24	875	0.26	863	0.25	655	0.23
rhone_alps	35	429	0.16	902	0.43	557	0.23	316	0.15	388	0.18	445	0.2
tbed_ctct	84	2093	0.62	323	0.23	26454	6.61	323	0.22	532	0.31	38028	5.39
tbed_nocoll	84	129556	11.53	1253502	188.74	476532	97.63	116905	12.97	297005	41.97	404925	62.9
tbed_noderail	84	145612	10.84	1114607	147.26	269866	43.45	118250	11.25	250258	31.23	525335	86.98
verriegel4	65	1084	0.33	2932	1.42	1062	0.34	1147	0.35	2360	1.43	2761	2.12
profisafe_i4	80	81	0.17	401	0.51	1335	4.44	37	0.13	4790	40.73	37	0.14
profisafe_i4_host	28	74	0.13	138	0.14	337	0.56	32	0.09	95	0.17	32	0.09
profisafe_i4_slave	14	7	0.04	105	0.07	5	0.03	5	0.03	5	0.04	5	0.03
profisafe_i5	88	98	0.19	192	0.27	8630	4.81	37	0.16	2189	1.53	37	0.16
profisafe_i5_host	28	91	0.14	138	0.17	1070	3.56	32	0.11	3344	7.16	32	0.11
profisafe_i6	94	98	0.23	197	0.28	304	0.78	37	0.18	13889	24.23	37	0.18
profisafe_i6_host	28	91	0.16	1223	1.16	7990	12.91	32	0.13	3287	11.5	32	0.12
profisafe_inclusion_i4host	78	103	0.15	477	0.49	44	0.12	44	0.12	227	0.39	86	0.16
profisafe_inclusion_o4host	84	103	0.17	248	0.2	44	0.13	44	0.13	227	0.41	179	0.31
profisafe_inclusion_o4slave	84	103	0.17	658	0.82	44	0.13	44	0.13	165	0.27	86	0.17
profisafe_o4	90	81	0.19	286	0.29	860	1.87	37	0.15	189	0.28	37	0.15
profisafe_o4_host	30	74	0.13	1550	1.43	102	0.2	32	0.1	102	0.19	32	0.1
profisafe_o4_slave	16	7	0.05	59	0.08	5	0.04	5	0.05	5	0.04	5	0.05
profisafe_o5	99	98	0.21	1293	1.33	134	0.33	37	0.17	187	0.33	37	0.18
profisafe_o5_host	30	91	0.14	642	0.6	1027	3.49	32	0.11	6512	5.71	32	0.12
profisafe_o6	106	98	0.25	235	0.34	124	0.38	37	0.2	124	0.38	37	0.2
profisafe_o6_host	30	91	0.16	181	0.25	1170	5.06	32	0.13	277	0.39	32	0.13
ftechnik	36	25484	3.52	6795	2.98	26445	4.48	25307	4.77	6776	3.09	66800	10.09
rhone_tough	61	453398	6.14		10.1	1679910	19.11	175489	8.6	636432	8.78		6.13
tbed_uncont	84	176777	12.03	842054	114.53	490531	36.37	292917	30.87	274112	28.58	340548	44.68

Model		Min Events		Min NewEvents		Min States		Min Transitions		One		RelMax Common	
Name	Aut	States	Time	States	Time	States	Time	States	Time	States	Time	States	Time
big_bmw	31	281	0.15	281	0.15	282	0.14	281	0.14	272	0.15	277	0.14
fzelle	67	966	0.28	892	0.26	1241	0.37	1256	0.37	863	0.24	892	0.26
rhone_alps	35	458	0.16	289	0.12	533	0.21	533	0.21	458	0.16	278	0.1
tbed_ctct	84	30451	6.37	21896	4.42	8098	5.08	17414	4.83	36406	6.94	6121	1.63
tbed_nocoll	84	1594044	236.95	475835	54.75	986750	115.62	1391084	185.36	265472	50.43	277491	33.13
tbed_noderail	84	1345867	191.09	406660	41.7	946799	119.49	1352890	184.28	320888	55.1	275954	26.89
verriegel4	65	1036	0.31	1095	0.36	1095	0.36	1145	0.41	2360	1.4	1095	0.36
profisafe_i4	80	96	0.2	37	0.13	383	1.05	196	0.24	4790	40.72	37	0.13
profisafe_i4_host	28	32	0.09	32	0.1	19223	29.53	32	0.09	95	0.17	32	0.09
profisafe_i4_slave	14	5	0.03	5	0.03	5	0.03	105	0.07	5	0.03	5	0.03
profisafe_i5	88	2195	1.38	37	0.16	3357	5.76	123	0.22	2189	1.5	37	0.16
profisafe_i5_host	28	32	0.11	32	0.11	375	1.01	32	0.11	3344	7.23	32	0.11
profisafe_i6	94	43	0.19	37	0.17	8847	58.03	91	0.21	13889	23.67	37	0.18
profisafe_i6_host	28	32	0.13	32	0.13	12944	8.26	32	0.13	3287	11.29	32	0.13
profisafe_inclusion_i4host	78	137	0.22	121	0.17	163	0.37	160	0.19	260	0.43	99	0.16
profisafe_inclusion_o4host	84	131	0.23	121	0.19	152	0.37	112	0.19	260	0.47	99	0.17
profisafe_inclusion_o4slave	84	2791	0.89	121	0.19	559	0.63	112	0.18	182	0.28	99	0.17
profisafe_o4	90	37	0.15	37	0.15	189	0.39	43	0.16	189	0.28	37	0.15
profisafe_o4_host	30	32	0.09	32	0.09	120	0.23	32	0.09	102	0.2	32	0.09
profisafe_o4_slave	16	5	0.04	5	0.05	5	0.04	11	0.06	5	0.05	5	0.04
profisafe_o5	99	37	0.18	37	0.17	141	0.34	43	0.18	187	0.32	37	0.18
profisafe_o5_host	30	32	0.1	32	0.12	387	0.89	32	0.11	6512	5.64	32	0.12
profisafe_o6	106	37	0.2	37	0.2	131	0.4	43	0.21	124	0.37	37	0.2
profisafe_o6_host	30	32	0.13	32	0.13	1168	9.71	32	0.13	277	0.39	32	0.12
ftechnik	36	7468	4.3	6799	2.85	10760	5.23	5878	3.92	6513	2.88	6970	2.36
rhone_tough	61	550741	10.92	249062	9.46		11.98		9.5	397758	8.68		10.85
tbed_uncont	84	1113150	130.02	513985	46.26	574284	70.14	871199	101.93	279414	30.87	377358	36.51

Table 6.4
MODULAR PROJECTING CONTROLLABILITY, EXHAUSTIVE, MAX PROJECTION 1000

Model		Modular projecting controllability, exhaustive, max projection 1000											
		All		Early NotAccept		Late NotAccept		MaxCommon Events		MaxCommon Uncontr		Max States	
Name	Aut	States	Time	States	Time	States	Time	States	Time	States	Time	States	Time
big_bmw	31	31528	0.4	13431	0.19	16391	0.18	14432	0.18	14418	0.16	16391	0.18
fzelle	67	9935	0.27	1655	0.22	633	0.23	2869	0.25	3850	0.25	3557	0.22
rhone_alps	35	17865	0.26	66143	0.61	24700	0.3	10424	0.19	19784	0.33	23602	0.29
tbed_ctct	84	74419	1.42	33509	0.53	2516207	43.54	27390	0.46	38924	0.96	604457	10.24
tbed_nocoll	84	2656809	114.09	16546885	1103.35	16604262	664.74	1934572	35.78	7445386	343.62	8707170	420.07
tbed_noderail	84	2117338	59.27	17559141	1368.08	12670069	645.38	2425961	82.5	7250563	340.16	11088228	617.63
verriegel4	65	67574	0.67	231228	1.94	29860	0.84	32982	0.87	186632	1.98	281829	2.97
profisafe_i4	80	6571	0.56	41764	0.9	10429	0.52	45	0.14	18781	0.95	45	0.13
profisafe_i4_host	28	3171	0.23	62785	0.82	9403	0.54	40	0.1	4234	0.29	40	0.1
profisafe_i4_slave	14	8	0.04	2100	0.09	5	0.03	5	0.03	5	0.03	5	0.03
profisafe_i5	88	8588	0.72	76781	1.35	10429	0.61	45	0.16	8420	0.58	45	0.16
profisafe_i5_host	28	5192	0.41	75878	1.11	17537	0.87	40	0.12	34041	1.88	40	0.12
profisafe_i6	94	9612	1.06	58858	1.11	10419	0.7	45	0.18	21796	1.6	45	0.18
profisafe_i6_host	28	9604	1.1	58785	1.07	3133	0.23	40	0.13	21791	1.73	40	0.13
profisafe_inclusion_i4host	78	5217	0.22	1212	0.22	51	0.12	51	0.12	11181	0.28	79	0.14
profisafe_inclusion_o4host	84	5217	0.23	23609	0.52	51	0.13	51	0.13	15402	0.46	8259	0.3
profisafe_inclusion_o4slave	84	7342	0.34	37875	0.7	51	0.13	51	0.13	11181	0.3	125706	0.99
profisafe_o4	90	5214	0.37	59821	0.9	14726	1.01	45	0.15	75715	3.29	45	0.15
profisafe_o4_host	30	4148	0.24	12386	0.35	9398	0.55	40	0.1	5475	0.46	40	0.1
profisafe_o4_slave	16	8	0.05	73	0.09	5	0.05	5	0.04	5	0.05	5	0.04
profisafe_o5	99	8588	0.96	41498	0.78	18536	0.86	45	0.18	55460	14.9	45	0.18
profisafe_o5_host	30	8580	0.74	44531	0.95	24327	0.96	40	0.12	38992	1.92	40	0.12
profisafe_o6	106	7194	0.52	73978	1.36	4239	0.45	45	0.2	28497	0.95	45	0.21
profisafe_o6_host	30	5192	0.31	1305	0.45	10424	0.94	40	0.13	10424	0.7	40	0.13
ftechnik	36	335976	8.13	321765	4.4	396638	11.69	429044	11.23	251360	4.85	563667	21.86
rhone_tough	61	638699	21.89		34.11	1107071	37.12	746546	23.95	1108472	43.27		8.75
tbed_uncont	84	2229655	147.83	12049242	686.84	9601683	550.94	4091566	171.08	4489598	187.52	5984090	337.31

Model		Min Events		Min NewEvents		Min States		Min Transitions		One		RelMax Common	
Name	Aut	States	Time	States	Time	States	Time	States	Time	States	Time	States	Time
big_bmw	31	12420	0.15	12420	0.15	16427	0.16	12420	0.15	14432	0.15	14436	0.17
fzelle	67	1973	0.27	1869	0.26	2362	0.36	3375	0.36	3850	0.24	1869	0.26
rhone_alps	35	9704	0.18	3508	0.15	17848	0.24	17848	0.24	6669	0.16	2477	0.14
tbed_ctct	84	3034759	70.64	1564168	46.64	465969	4.79	3189913	62.43	2449591	58.01	261723	3.36
tbed_nocoll	84	30622807	1820.36	9761331	448.45	12179623	695.32	25119535	1271.89	9972105	356.81	6703498	385.03
tbed_noderail	84	24161489	1442.44	10716432	743.17	14946347	922.07	26328158	1506.52	11772799	626.69	4546061	235.95
verriegel4	65	60039	0.83	26979	0.79	72157	0.92	41864	0.78	186632	2.08	26979	0.79
profisafe_i4	80	5347	0.34	45	0.13	10440	0.54	5233	0.26	18781	0.96	45	0.13
profisafe_i4_host	28	2058	0.12	2058	0.11	27713	1.08	2058	0.12	4234	0.29	2058	0.12
profisafe_i4_slave	14	5	0.03	5	0.03	5	0.03	2100	0.09	5	0.03	5	0.03
profisafe_i5	88	14600	0.98	45	0.16	10454	0.68	3206	0.28	8420	0.56	45	0.16
profisafe_i5_host	28	7128	0.24	7128	0.24	17548	0.89	2058	0.14	34041	1.84	2058	0.15
profisafe_i6	94	6257	0.59	2063	0.21	18567	1.06	2131	0.25	21796	1.61	2063	0.21
profisafe_i6_host	28	7136	0.29	2058	0.16	3144	0.25	2058	0.16	21791	1.7	2058	0.16
profisafe_inclusion_i4host	78	7309	0.33	8282	0.3	13286	0.36	4147	0.19	10263	0.31	6158	0.21
profisafe_inclusion_o4host	84	7302	0.34	8282	0.32	14394	0.49	5247	0.28	14511	0.52	6158	0.23
profisafe_inclusion_o4slave	84	7309	0.35	8282	0.32	13478	0.53	4251	0.28	10263	0.33	6158	0.24
profisafe_o4	90	3051	0.2	45	0.15	14737	1.02	52	0.16	75715	3.28	45	0.15
profisafe_o4_host	30	2063	0.12	40	0.1	5482	0.47	40	0.1	16982	1.91	40	0.1
profisafe_o4_slave	16	5	0.05	5	0.04	5	0.05	12	0.06	5	0.05	5	0.04
profisafe_o5	99	6240	0.48	2063	0.21	10430	0.64	2070	0.21	55460	15.06	2063	0.2
profisafe_o5_host	30	40	0.11	40	0.11	10449	0.67	40	0.11	38992	1.9	40	0.12
profisafe_o6	106	45	0.21	45	0.2	4250	0.46	52	0.22	28497	0.93	45	0.21
profisafe_o6_host	30	40	0.13	40	0.13	10435	0.73	40	0.13	10424	0.7	40	0.13
ftechnik	36	296140	9.72	242536	8.44	436762	7.29	374514	7.1	254631	4.15	211651	5.18
rhone_tough	61	993382	39.92	969499	23.85		11.44		9.67	922415	30.05	736023	29.05
tbed_uncont	84	15641854	876.92	9185235	486.96	9264324	517.08	15294820	863.85	5364884	184.03	6164871	403.37

Table 6.5
MODULAR CONTROLLABILITY USING NON-EXHAUSTIVE PROJECTION
AS PRE-PROCESS, MAX PROJECTION 1000

Model		Modular controllability using non-exhaustive projection as pre-process, max projection 1000											
		All		Early NotAccept		Late NotAccept		MaxCommon Events		MaxCommon Uncontr		Max States	
Name	Aut	States	Time	States	Time	States	Time	States	Time	States	Time	States	Time
big_bmw	31	287	0.29	287	0.17	287	0.16	287	0.2	287	0.16	287	0.15
fzelle	67	965	0.55	965	0.64	965	0.63	965	0.63	965	0.56	965	0.63
rhone_alps	35	236	0.16	236	0.17	236	0.16	236	0.16	236	0.16	236	0.16
tbed_ctct	84	161540	1.09	2117575	10.65		41.18	553605	3.24	2725576	14.77		17.58
tbed_nocoll	84	399609	5.97	495402	12.64		45.92	346256	11.69	494710	12.59		87.29
tbed_noderail	84	395986	4.41	496700	10.96		68.14	356087	9.54	482816	10.16		48.78
verriegel4	65	2162	0.89	2162	0.9	2162	0.88	2162	0.96	2162	0.89	2162	0.96
profisafe_i4	80	4552	4.59		87.31	4504	3.67	4458	3.56	4504	3.65	4917	4.41
profisafe_i4_host	28	1141	6.6	1141	6.46	1141	6.09	1141	6.22	1141	6.32	1141	6.13
profisafe_i4_slave	14	1005	0.79	1005	0.94	1005	0.78	1005	0.78	1005	0.93	1005	0.79
profisafe_i5	88	5148	6.49		136.36	5100	6.46	5054	6.19	5100	6.47	5504	7.12
profisafe_i5_host	28	1433	9.87	1433	8.86	1433	8.64	1433	8.51	1433	8.56	1433	8.52
profisafe_i6	94	3252	4.01		134.79	3250	3.7	3229	3.59	3250	3.72	3329	4.05
profisafe_i6_host	28	1296	8.62	1296	8.63	1296	8.71	1296	9.06	1296	8.76	1296	9.03
profisafe_inclusion_i4host	78	4539	2.96		24.71	4259	3.02	4277	2.96	4329	2.76	5038	3.43
profisafe_inclusion_o4host	84	4548	2.39		77.8	3565	2.19	3816	2.44	3868	2.25	6485	2.97
profisafe_inclusion_o4slave	84	4093	2.09		68.23	3562	2.2	3813	2.4	3834	2.22	4955	2.45
profisafe_o4	90	4452	3.56		133.72	4417	3.61	4371	3.75	4417	3.83	4815	4.08
profisafe_o4_host	30	1219	6.3	1219	6.34	1219	6	1219	6.32	1219	6	1219	6.24
profisafe_o4_slave	16	351	0.46	351	0.45	351	0.46	351	0.46	351	0.69	351	0.45
profisafe_o5	99	3619	4.31		95.77	3607	4.35	3610	4.01	3607	4.33	3681	4.08
profisafe_o5_host	30	1315	8.58	1315	8.19	1315	7.88	1315	8.19	1315	7.8	1315	8.2
profisafe_o6	106	3603	4.7		141.34	3591	4.72	3594	5.38	3591	4.71	3665	4.69
profisafe_o6_host	30	1437	9.56	1437	9.47	1437	9.04	1437	9.54	1437	9.48	1437	9.22
ftechnik	36	5824	3.38	4170	4.1	6677	4.69	8167	4.32	5051	4.15	8129	4.43
rhone_tough	61		9.25		8.56		10.36		7.72		10.3		7.92
tbed_uncont	84	157937	2.76	133635	4.87	752769	7.93	109304	4.22	113948	4.27	1762111	15.88

Model		Min Events		Min NewEvents		Min States		Min Transitions		One		RelMax Common	
Name	Aut	States	Time	States	Time	States	Time	States	Time	States	Time	States	Time
big_bmw	31	287	0.2	287	0.15	287	0.15	287	0.2	287	0.14	287	0.15
fzelle	67	965	0.64	965	0.63	965	0.55	965	0.62	965	0.63	965	0.63
rhone_alps	35	236	0.16	236	0.16	236	0.16	236	0.16	236	0.16	236	0.16
tbed_ctct	84		27.34		14.38		25.25		21.38		18.09		10.23
tbed_nocoll	84		86.06	803777	14.44	263891	5.2	442041	6.39		82.17	337247	8.21
tbed_noderail	84	3519361	29.62	888508	12.34	266790	4.2	769310	6.54	3127840	30.93	354693	7.28
verriegel4	65	2162	0.92	2162	0.94	2162	0.89	2162	0.86	2162	0.86	2162	0.87
profisafe_i4	80	4440	3.56	4440	3.57	4441	3.57	4441	3.55	4441	3.55	4440	3.57
profisafe_i4_host	28	1141	5.93	1141	5.99	1141	5.99	1141	5.99	1141	5.95	1141	5.93
profisafe_i4_slave	14	1005	0.8	1005	0.94	1005	0.82	1005	0.95	1005	0.81	1005	0.95
profisafe_i5	88	5036	6.12	5036	6.36	5037	6.32	5037	6.13	5037	6.3	5036	6.31
profisafe_i5_host	28	1433	8.59	1433	8.5	1433	8.61	1433	8.47	1433	8.62	1433	8.43
profisafe_i6	94	3211	3.62	3211	3.59	3234	3.78	3266	3.65	3236	3.86	3211	3.63
profisafe_i6_host	28	1296	8.59	1296	8.65	1296	8.6	1296	8.57	1296	8.56	1296	8.56
profisafe_inclusion_i4host	78	4307	3.28	4272	2.88	4248	3.12	4248	2.86		72.22	4272	2.89
profisafe_inclusion_o4host	84	4306	2.08	4272	2.08	31685	2.73	31678	2.52		64.21	4272	2.08
profisafe_inclusion_o4slave	84	4303	2.28	4269	2.07	14888	2.49	14881	2.28		64.39	4269	2.28
profisafe_o4	90	4339	3.58	4339	3.37	4354	3.75	4340	3.35	4354	3.69	4339	3.35
profisafe_o4_host	30	1219	6.03	1219	6.31	1219	6.05	1219	6.26	1219	5.99	1219	6.33
profisafe_o4_slave	16	351	0.46	351	0.45	351	0.45	351	0.46	351	0.46	351	0.46
profisafe_o5	99	3592	4.28	3592	4.03	3619	4.44	3597	4.07	3593	4.22	3592	4.01
profisafe_o5_host	30	1315	7.88	1315	8.21	1315	8.34	1315	7.92	1315	8.2	1315	8.35
profisafe_o6	106	3576	4.92	3576	4.68	3603	4.81	3581	4.98	3577	4.54	3576	4.68
profisafe_o6_host	30	1437	9.53	1437	9.07	1437	9.49	1437	9.71	1437	9.16	1437	9.59
ftechnik	36	4946	4.35	5051	4.27	4057	4.01	4057	3.87	4142	3.98	4164	4.18
rhone_tough	61		10.46		10.33		10.25		10.97		10.17		10.3
tbed_uncont	84	3601903	23.97	156609	4.57	275630	3.3	358016	3.77	1674127	14	111386	3.55

Table 6.6
MODULAR PROJECTING CONTROLLABILITY, NON-EXHAUSTIVE, MAXCOMMONEVENTS

Model		Modular projecting controllability, non-exhaustive, maxcommonevents											
		100		200		400		800		1600		3200	
Name	Aut	States	Time	States	Time	States	Time	States	Time	States	Time	States	Time
tbed_ctct	84	1853	0.53	323	0.29	323	0.34	323	0.26	323	0.26	323	0.32
tbed_nocoll	84	201870	6.54	65161	7.52	74063	9.41	109016	13.89	163289	17.75	267650	34.61
tbed_noderail	84	176516	6.11	74124	6.69	90175	10.87	129152	16.28	184904	22.73	313859	38.8
rhone_tough	61		11.09		12.5		10.41		16	109870	10.07	121524	11.9
tbed_uncont	84	863808	11.54	374537	9.96	374855	11.48	240165	18.93	357395	43.02	489759	74.29

Table 6.7
MODULAR PROJECTING CONTROLLABILITY, NON-EXHAUSTIVE, MAXCOMMONEVENTS

Model		Modular controllability using non-exhaustive projection as pre-process, maxcommonevents											
		100		200		400		800		1600		3200	
Name	Aut	States	Time	States	Time	States	Time	States	Time	States	Time	States	Time
tbed_ctct	84	551689	3.55	552005	3.54	552405	3.43	553205	3.49	554805	3.5	1059024	7.09
tbed_nocoll	84	440041	4.39	337351	9.46	337705	9.57	345856	12.76	1007593	44.82	1012393	43.11
tbed_noderail	84	477971	4.5	350404	9.41	354888	10.04	355688	10.03	192790	23.64	185627	49.27
rhone_tough	61		25.58		25.51		23.29		7.52		10.72		12.88
tbed_uncont	84	312859	2.63	100549	3.41	100903	3.42	108904	4.2	75361	9.43	117404	22.45

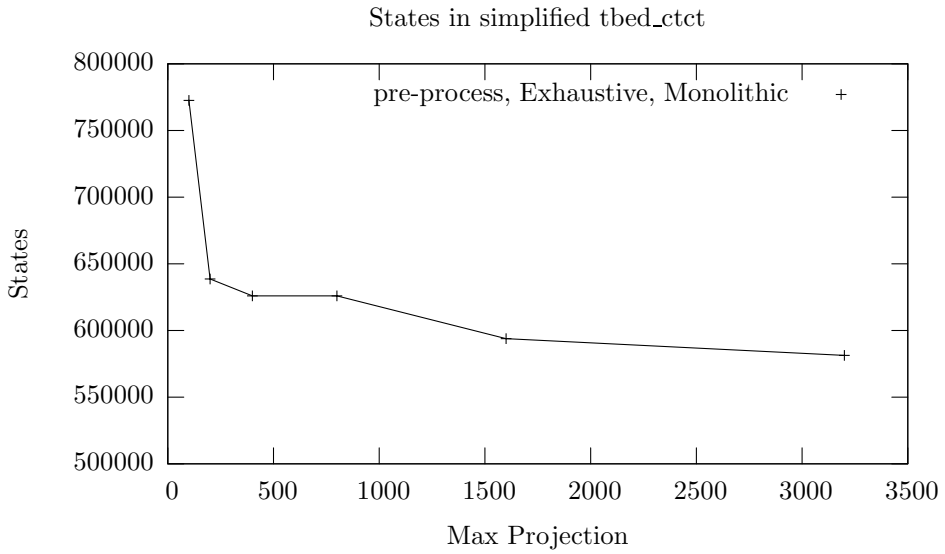

Figure 6.9: States in simplified tbed_ctct

Table 6.8
MONOLITHIC CONTROLLABILITY USING NON-EXHAUSTIVE PROJECTION AS PRE-PROCESS

Model		Monolithic controllability using non-exhaustive projection as pre-process											
		100		200		400		800		1600		3200	
Name	Aut	States	Time	States	Time	States	Time	States	Time	States	Time	States	Time
big_bmw	31	0	0.31	0	0.18	0	0.16	0	0.21	0	0.16	0	0.16
fzelle	67	4794	0.41	0	0.45	0	0.59	0	0.52	0	0.61	0	0.53
rhone_alps	35	0	0.18	0	0.18	0	0.18	0	0.18	0	0.18	0	0.18
tbed_ctct	84		11.8		11.25		11.09		11.09		11.46		12
tbed_nocoll	84	27659	0.87	9001	0.89	9001	0.93	8662	1.42	6780	7.77	6780	7.99
tbed_noderail	84	29928	0.95	9001	1.01	8658	1.22	8658	1.05	6561	6.08	6495	13.9
verriegel4	65	19470	0.49	1224	0.43	1224	0.47	0	0.7	0	0.81	0	0.82
profisafe_i4	80		19.41		19.23		21.67		24.48		24.12		25.8
profisafe_i4_host	28	106250	3.63	0	4.52	0	5.94	0	6.42	0	5.86	0	5.76
profisafe_i4_slave	14	3564	0.12	3564	0.11	3564	0.13	0	0.79	0	0.81	0	0.93
profisafe_i5	88		19.12		19.15		19.47		22.17		21.7		22.14
profisafe_i5_host	28	176130	6.16	0	5.66	0	8.25	0	9.11	0	8.62	0	8.36
profisafe_i6	94		20.32		20.6		21.34		23.86		24.6		27.43
profisafe_i6_host	28	187747	7.37	183957	8.5	0	9.72	0	9.41	0	9.08	0	9.1
profisafe_inclusion_i4host	78		14.76		15.29		15.47		15.78		16.55		18.93
profisafe_inclusion_o4host	84		17.99		17.86		17.73		17.92		18.09		20.51
profisafe_inclusion_o4slave	84		17.83		17.75		17.7		18.29		17.9		21.05
profisafe_o4	90		19.78		19.37		19.24		22.95		23.06		23.95
profisafe_o4_host	30	198375	5.99	192809	6.23	0	6.51	0	6.13	0	6.44	0	6.14
profisafe_o4_slave	16	0	0.41	0	0.7	0	0.47	0	0.47	0	0.48	0	0.48
profisafe_o5	99		19.3		19.61		20.27		22.53		22.34		22.94
profisafe_o5_host	30	351431	11.38	343294	11.66	0	8.31	0	8.23	0	8.14	0	8.26
profisafe_o6	106		20.42		20.91		21.14		24.86		24.93		25.54
profisafe_o6_host	30	567931	20.46	556623	20.76	0	9.69	0	9.54	0	9.54	0	9.76
ftechnik	36	2	0.42	2	0.7	2	1.07	2	1.95	2	1.82	1	3.5
rhone_tough	61		10.18		10.32	848746	4.85	304138	2.34	256830	3.55	53538	7.44
tbed_uncont	84	8577	0.75	2903	0.85	2903	0.91	2788	1.45	2312	3.84	2183	8.47

Table 6.9
MONOLITHIC CONTROLLABILITY USING EXHAUSTIVE PROJECTION AS PRE-PROCESS

Model		Monolithic controllability using exhaustive projection as pre-process											
		100		200		400		800		1600		3200	
Name	Aut	States	Time	States	Time	States	Time	States	Time	States	Time	States	Time
big_bmw	31	0	1.04	0	1.11	0	0.97	0	0.97	0	1	0	0.99
fzelle	67	0	0.74	0	0.78	0	0.89	0	0.95	0	1.03	0	0.99
rhone_alps	35	0	0.3	0	0.4	0	0.6	0	0.61	0	0.71	0	0.58
tbed_ctct	84	772545	4.39	638645	4.48	625946	5.87	625946	7.24	593879	17.17	581339	62.98
tbed_nocoll	84	9745	1.6	8703	2.15	8566	3.39	7624	10.54	6930	31.65	6930	143.19
tbed_noderail	84	9745	1.75	8703	2.24	7847	4.07	7606	10.94	6646	27.75	0	403.31
verriegel4	65	19470	0.94	0	1.4	0	2.03	0	2.52	0	3.16	0	3.63
profisafe_i4	80	0	19.47	0	45.68	0	84.26	0	194.78	0	411.53	0	731.33
profisafe_i4_host	28	0	8.92	0	15.86	0	27.66	0	30.33	0	32.06	0	40.55
profisafe_i4_slave	14	0	0.65	0	0.83	0	0.95	0	1	0	0.97	0	0.96
profisafe_i5	88		29.21	0	25.94	0	40.35	0	90.94	0	245.38	0	537.46
profisafe_i5_host	28	0	10.53	0	16.29	0	25.1	0	38.43	0	55.49	0	62.4
profisafe_i6	94		32.21	0	33.5	0	59.87	0	113.33	0	217.53	0	691.63
profisafe_i6_host	28	0	11.88	0	20.06	0	32.9	0	53.68	0	66.23	0	96.33
profisafe_inclusion_i4host	78	0	4.8	0	7.71	0	12.26	0	27.03	0	60.13	0	139.25
profisafe_inclusion_o4host	84	0	5.49	0	10.04	0	17.95	0	35.85	0	66.25	0	160.32
profisafe_inclusion_o4slave	84	0	4.16	0	7.26	0	11.54	0	26.5	0	72.69	0	161.99
profisafe_o4	90	0	14.1	0	25.25	0	38.13	0	84.69	0	160.48	0	448.42
profisafe_o4_host	30	0	11.42	0	16.87	0	27.2	0	31.66	0	34.53	0	41.56
profisafe_o4_slave	16	0	0.95	0	1.01	0	1.01	0	1.02	0	1.03	0	1.03
profisafe_o5	99		30.93	0	27.05	0	42.53	0	90.94	0	166.02	0	898.49
profisafe_o5_host	30	0	13.61	0	19.16	0	30.55	0	49.13	0	64.26	0	70.48
profisafe_o6	106		34.91	0	32.42	0	49.81	0	126.79	0	239.45	0	498.96
profisafe_o6_host	30	0	15.28	0	22.97	0	31.95	0	65.41	0	75.62	0	107.68
ftechnik	36	2	2.42	1	4.52	1	6.48	1	13.66	1	35.71	1	62.17
rhone_tough	61	482058	3.29	177807	3.14	155964	6.41	90515	10.94	51975	41.6	41070	95.89
tbed_uncont	84	3159	1.56	2885	2.07	2878	3.27	2455	10.23	2231	23.81	2166	146.03

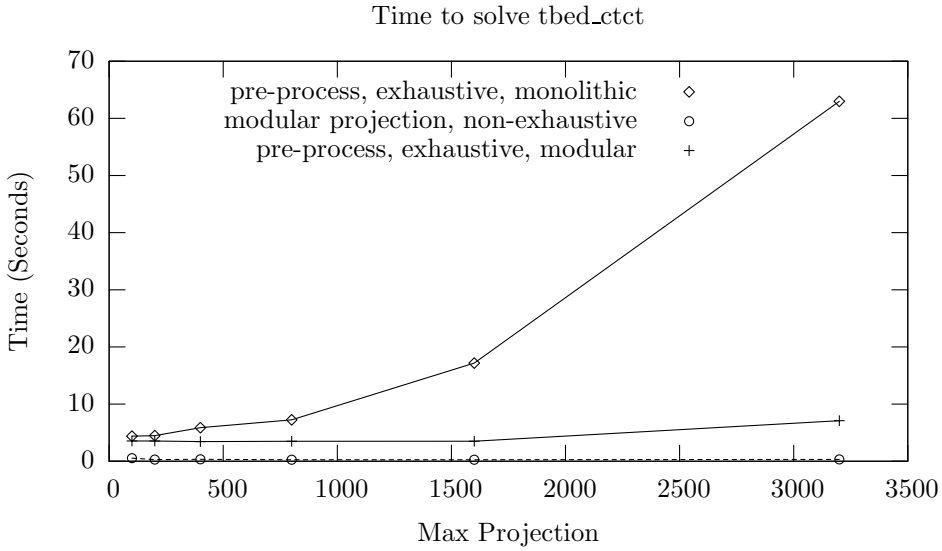

Figure 6.10: Time to solve tbed_ctct

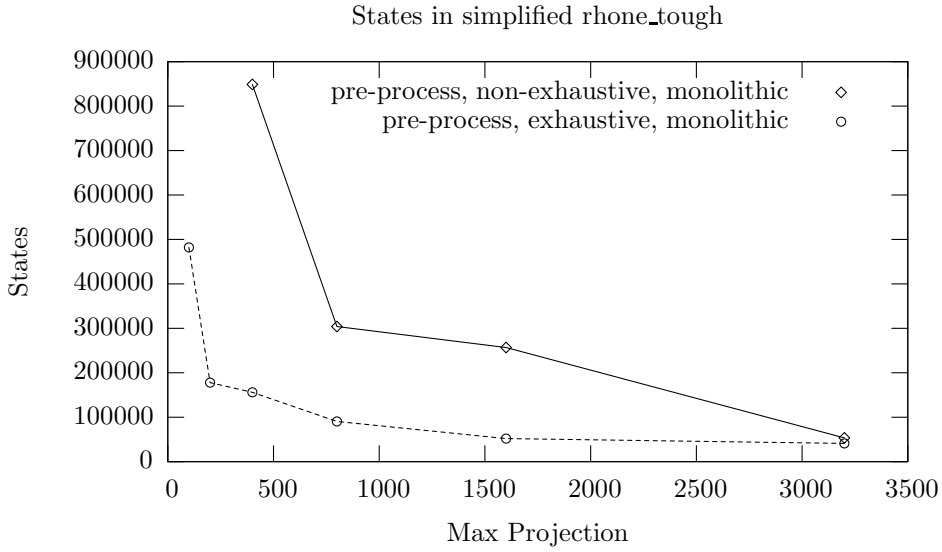

Figure 6.11: States in simplified rhone_tough

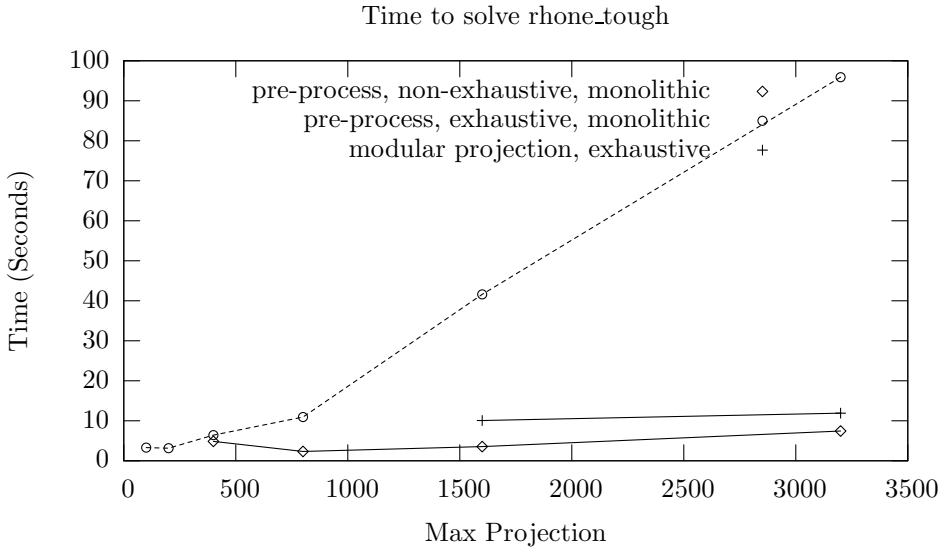

Figure 6.12: Time to solve rhone_tough

Chapter 7

Related Work

Ordered binary decision diagrams (OBDDs) are a method of representing large boolean formulae in with relatively little memory and can in some cases represent formulae with linear memory in relation with respect to the number of variables [5]. One of the ways of checking controllability on larger systems is to represent the states in the system and it's transition relation as an OBDD. Such a representation can in many cases deal with far larger state spaces than can regularly be explored.

Partial order reduction is a method of reducing the amount of the synchronous product of a set of automata we search by observing that many events in a model are actually independent of one another, and as such we don't care in what order they occur [5]. We can exploit this by when it is appropriate only exploring the states reached by taking one arbitrary ordering of these events sometimes dramatically reduce the number of states considered. It should be noted that projection seems to remove independence between events anyway.

Chapter 8

Conclusion

The modular method of checking controllability has been implemented in the WATERS toolkit and the experimental results have shown its performance to be equivalent to that shown by the original implementation [3]. In addition to this, several variations of the controllability checker were developed. Of these the Parallel and Culling checker, while showing some promise did not bring much improvement over the standard modular approach to controllability checking. The projecting approach, however showed lots of improvement in performance, in particular being capable of solving the rhone_tough problem. This had up till this point never been solved before.

Future work could include looking into ways of improving the performance of the determinisation step of the controllability checker. A possible way of doing this could be to use a OBDD [5] to represent the automaton which we are attempting to determinise. Also, as intuitively projection should gain the best results when two automata which are related to each other are projected together, and the person who designed the model should ideally know which parts of the model are most related to one another, it could be useful to make it possible for the person modelling to put in information stating that these automata naturally go together and then use that information when projecting. Also it seems that the heuristics for selecting new automata to add to the composition for modular checking could be improved. One way of doing this could be to instead of finding just one counterexample in a composition, to find a number of counterexamples and then perhaps select which new automata to add to the composition based upon what number of these counterexamples it rejects.

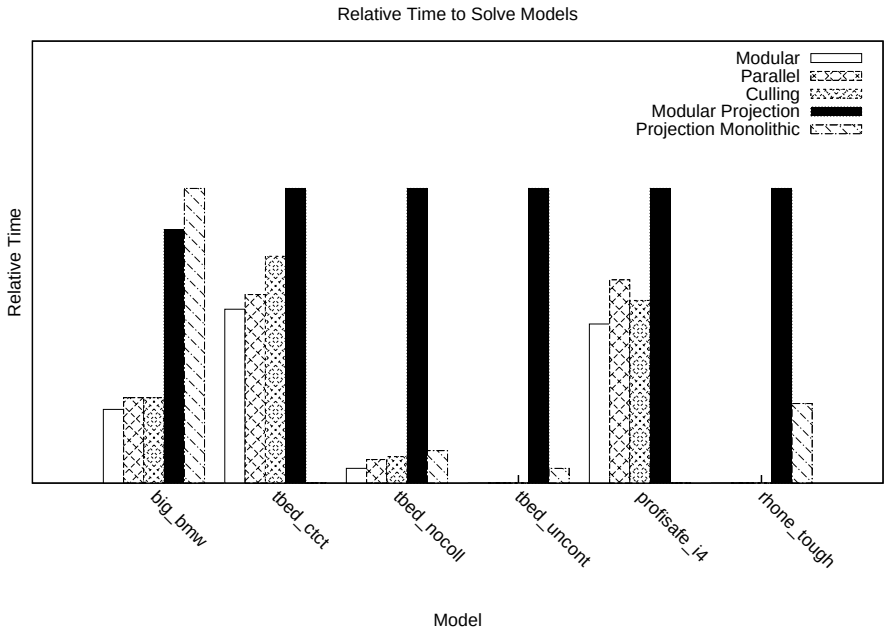

Figure 8.1: Synchronous product of Small factory with modified buffer

Bibliography

[1] K. Åkesson, H. Flordal, and M. Fabian. Exploiting modularity for synthesis and verification of supervisors. In *15th IFAC World Congress on Automatic Control*, 2002.

[2] Bertil Brandin and François Charbonnier. The supervisory control of the automated manufacturing system of the AIP. In *Rensselaer's 4th Computer Integrated Manufacturing and Automation Technology*, pages 319–324, 1994.

[3] Bertil A. Brandin, Robi Malik, and Petra Malik. Incremental verification and synthesis of discrete-event systems guided by counter-examples. 12(3):387–401, May 2004.

[4] F. Charbonnier. Commande par supervision des systèmes à événements discrets: application à un site expérimental l'Atelier Inter-établissement de Productique. Technical report, Laboratoire d'Automatique de Grenoble, 1994.

[5] Edmund M. Clarke, Jr., Orna Grumberg, and Doron A. Peled. *Model Checking*. 1999.

[6] Petra Dietrich. Projekt BMW E65 CAS — FH-Master — eine Modellierung in DCD. Technical report, , Corporate Technology, Software and Engineering 4, 2000.

[7] Hugo Flordal and Robi Malik. Modular nonblocking verification using conflict equivalence. In *8th '06*, pages 100–106, July 2006.

[8] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. 1979.

[9] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. 2001.

[10] R. J. Leduc. PLC implementation of a DES supervisor for a manufacturing testbed: An implementation perspective. Master's thesis, 1996.

[11] C. Lewerentz and T. Linder. *Case Study "Production Cell"*, volume 891. 1995.

[12] Annette Lötzbeyer and R. Mühlfeld. Task description of a flexible production cell with real time properties. Technical report, 1996.

[13] Petra Malik. *From Supervisory Control to Nonblocking Controllers for Discrete Event Systems*. PhD thesis, 2003.

[14] R. Malik and R. Mühlfeld. A case study in verification of UML statecharts: the PROFIsafe protocol. 9(2):138–151, February 2003.

[15] Robi Malik and Reinhard Mühlfeld. Testing the PROFIsafe protocol using automatically generated test cases based on a formally verified model. Technical report, , Corporate Technology, Software and Engineering 1, 2002.

[16] Profibus Nutzerorganisation e. V. PROFIsafe—profile for safety technology, version 1.12, 2002.

[17] Peter J. G. Ramadge and W. Murray Wonham. The control of discrete event systems. 77(1):81–98, January 1989.

[18] W. M. Wonham. Notes on control of discrete event systems, 1999. , ; at `http://www.control.utoronto.ca/` under "Research Areas".