A COMPARISON OF SYNTHESIS TOOLS FOR SUPERVISORY CONTROLLERS

A. Sanchez[‡†*], J. Reza[‡], J. Douriet[‡] and R. Gonzalez[‡]

[‡] Depto. de Ingenieria Electrica y Computacion
Centro de Investigacion y Estudios Avanzados (Cinvestav)
Apdo. Postal 31-438, Guadalajara 45091, Jalisco, Mexico
[†]Instituto Mexicano del Petroleo
Eje Central Lazaro Cardenas 152, Mexico D.F. 007800, Mexico

Keywords: Discrete-event systems, supervisory synthesis

Abstract

This paper presents a numeric performance comparison of synthesis tools for supervisory controllers. First, a BDD implementation of supervisory synthesis operations is presented. This implementation is based on a predicate representation of SCT previously proved to be successful on establishing a symbolic calculation framework. The implementation is compared with UKDES and Supremica, two tools for supervisory control synthesis using explicit algorithms. Benchmark problems are established for asynchronous product, synchronous product and supremal controllable language calculations. Results of numerical experiments are presented showing a better performance of the symbolic implementation. However, for solving industrial applications it is still needed to improve the computational performance as well as using other design approaches (e.g. modularity and hierarchy).

1 Introduction

Supervisory Control Theory (SCT) [10] is recognized as one of the most solid synthesis theories for discrete-event controllers. The process is modeled using finite state machines (FSMs) and their associated set of finite trajectories (i.e. *-languages). The control mechanism is known as supervisor and controls the process by synchronizing shared transitions. SCT guarantees the existence of a supremal controllable language satisfying maximally a set of specifications, usually of safety nature. Applications have been reported in a wide range of areas such as communication protocols, data base management, processing and manufacturing systems [3].

Synthesis calculations are based on FSM operations and their complexity is polynomial in the best of cases [13]. However, in practice, it can be difficult to achieve efficient implementations. Moreover, handling processes of realistic size may be hampered by the state-explosion problem. Regarding the supremal controllable language calculations, several alternatives can be found in the open literature. Initially, a language fix–point operator and an explicit algorithm were proposed by [14]. The latter was implemented in the well-known TCT software. Non-recursive formulas have been also devised for closed languages with a temporal complexity of $O(mn^2)$, where m and n are the state cardinality of the process and specification FSMs, respectively [2].

The use of predicates and predicate transformers as a modeling framework for SCT has also been recognized as an useful approach for considering structural properties of FSMs and for avoiding the explicit enumeration of state spaces [7, 16]. A predicate-based fix–point operator was implemented in a synthesis tool[5] encoded symbolically with Binary Decision Diagrams (BDDs) and reportedly solved using *Ver*, a package for BDD handling [6]. An example of 1.3×10^6 potentially reachable states was used to show the applicability of the approach. The largest case solved used approximately 20 min and 7.7 Mb of a DEC300. However details were not given as to asses the performance of this approach and the associated tool.

Vector-addition systems have been also used to model DES [8]. In this case, the synthesis problem was posed as an LIP problem. The approach was applied to a manufacturing facility with 8.2×10^7 potential states.

Recently, a predicate-based incremental algorithm with variable reordering was proposed [16]. The encoding is carried out using integer decision diagrams (IDDs). Modularity in the plant and specification is exploited and specific problems up to 10^{23} potential states are solved in reasonable times and use of memory. However, the synthesis algorithm produces as a result the set of reachable and coreachable states of the supremal controllable language generator. Moreover, the tool is not currently available as a public domain software [15].

Public domain tools for supervisory control synthesis include:

- TCT, implements in C language the algorithms proposed by Ramadge and Wonham using explicit calculations. With a command-based terminal and simple syntax for input information, TCT is available for several operating platforms.
- Supremica ([1]) and UKDES ([4]), implement in C language with a Java GUI explicit calculations of Ramadge and Wonham's algorithms.

This paper presents a BDD-based implementation, termed SSPC, of a predicate-based framework for calculating the

*Corresponding author. e-mail address: arturo@gdl.cinvestav.mx

supremal controllable language. Following [16], predicates are built for modeling FSMs and defining the required operations for calculating the supremal controllable language. These include: synchronous and asynchronous products, image and pre-image calculations and the fix-point operator proposed by [5]. The performance of synchronous and asynchronous products as well as the supremal controllable language calculations are compared with their counterparts in UKDES V1.0 and Supremica V1104.

In the next section, the predicate-based operations mentioned previously for the calculation of the supremal controllable language are presented together with details of their implementation using BDDs. Section 3 shows the results of the benchmarking using specific examples for each exercise. Performance is characterized by CPU usage and calculation time vs. state space size of FSMs used as input data. A PC Pentium 4, 1.4 Ghz with 640 Mb RAM was used for all benchmark runs. UKDES and SSPC were executed on Mandrake Linux 8.2 operating system, while Supremica was executed on Windows 2000.

2 Predicate representation of FSM operations and their BDD implementation

As suggested by [7] and [16], predicates can be used to express structural properties of FSMs. Let Pr be a predicate defined over a state set of an FSM $M = \{Q, \Sigma, \delta, q_0, Q_m\}$ such that $Pr : Q \rightarrow \{0, 1\}$. Thus, Pr identifies a subset of states

$$\|Pr\| := \{q \in Q | Pr(q) = 1\} \tag{1}$$

In the same fashion, predicate $Re : Q \times \Sigma \times Q \longrightarrow \{0, 1\}$ defines a relation subset such that

$\|Re\| := \{(q, \sigma, q') \in Q \times \Sigma \times Q | Re(q, \sigma, q') = 1\}$

Thus, an FSM M and its associated languages $L(M)$, $L_m(M)$ can be represented by $M' = \{Q, \Sigma, \|T\|, \|I\|, \|Mr\|\}$ with

- $T : Q \times \Sigma \times Q \longrightarrow \{0, 1\}$ as the transition relation. The triplet (q, σ, q') satisfies T if and only if there is a transition σ such that $q' = \delta(q, \sigma)$, with $q, q' \in Q$ and $\sigma \in \Sigma$.
- $I : Q \longrightarrow \{0, 1\}$ as the initial state predicate. q satisfies I if and only if $q = q_0$.
- $Mr : Q \longrightarrow \{0, 1\}$ as the marked states predicate such that q satisfies Mr if and only if $q \in Q_m$.

2.1 BDD Encoding

The implementation was carried out in C using the BDD package Buddy [9]. Separated variable blocks were defined, in this order, for transitions, plant and specification state sets. In the case of plant and specification blocks, a sub-block is assigned to each FSM modeling a plant component or specification. In each sub-block all source states are codified by even variables and destination states are captured by odd variables.

2.2 Image and Pre-image of a Relation

Given a predicate Pr and a transition relation T, the image $Img\|Pr\|$ and pre-image $Pre\|Pr\|$[1] for obtaining state successors and predecessors are defined as:

$Img\|Pr\| := \{q' \in Q | (q, \sigma, q') \in \|T\|$ for some $\sigma \in \Sigma$ and some $q \in \|Pr\|\}$

$Pre\|Pr\| := \{q \in Q | (q, \sigma, q') \in \|T\|$ for some $\sigma \in \Sigma$ and some $q' \in \|Pr\|\}$

Thus, the definitions of reachability and coreachability can be written in terms of predicates.

Definition 1 ***Reachability***

Given an FSM $M = \{Q, \Sigma, \|T\|, \|I\|, \|Mr\|\}$, the set obtained by the succesive computation of the image, starting from the initial state, is given by

$$\|Al\| := \bigcup_{i=0}^{\infty} Img^i \|I\| \tag{2}$$

where $Img^i\|I\|$ is inductivey defined as

$Img^0\|I\| := \|I\|$,

$Img^{k+1}\|I\| := Img(Img^k\|I\|)$

Definition 2 ***Coreachability***

In the same fashion, for an FSM $M = \{Q, \Sigma, \|T\|, \|I\|, \|Mr\|\}$, the subset obtained by the calculation of the pre-image starting with the marked set is given by

$$\|Co\| := \bigcup_{i=0}^{\infty} Pre^i \|Mr\| \tag{3}$$

where $Pre^i\|Mr\|$ is defined inductively as in the case of reachability.

The calculation of the reachable and coreachable subsets is straightforward from the definition as shown in the following pseudo-code.

Reachable set

```
Reached = (*BDDInitialStateSystemSpec);
    do {
    ReachedBefore = Reached;
    /* Image calculation */
    Reached = Reached &
              (*BDDTranRelationSystemSpec);
    Reached = bdd_exist(Reached,
```

[1] The transition relation used is inferred from the context

```
            (*BDDCurrentSystemSpecSet)
            & (*BDDTransitionSet));
    tmp = bdd_replace(Reached,
          changePairSystemSpecF);
    Reached = ReachedBefore | tmp;
} while(ReachedBefore != Reached);
  return Reached;
```

Coreachable set

```
co_Reached = (*BDDMarkedStatesSystemSpec);
    do {
    co_ReachedBefore = co_Reached;
    co_Reached = bdd_replace(co_Reached,
                 changePairSystemSpecB);
    /* Preimage calculation */
    tmp = (*BDDTranRelationSystemSpec)
          & co_Reached;
    tmp = bdd_exist(tmp,
          (*BDDNextSystemSpecSet)
          & (*BDDTransitionSet));
    co_Reached = co_ReachedBefore | tmp;
} while(co_ReachedBefore != co_Reached);
  return co_Reached;
```

Definition 3 ***Asynchronous product***

Let two FSMs $M_1 = \{Q_1, \Sigma_1, \|T_1\|, \|I_1\|, \|Mr_1\|\}$, $M_2 = \{Q_2, \Sigma_2, \|T_2\|, \|I_2\|, \|Mr_2\|\}$, *with* $\Sigma_1 \cap \Sigma_2 = \emptyset$. *The asynchornous product* $M = M_1 \parallel M_2 = \{Q, \Sigma, \|T\|, \|I\|, \|Mr\|\}$

is given by:

$Q := \{Q_1 \times Q_2\}$
$\Sigma := \Sigma_1 \cup \Sigma_2$
$\|I\| := \{(q_1, q_2) \in Q | q_1 \in \|I_1\|$ *and* $q_2 \in \|I_2\|\}$
$\|Mr\| := \{(q_1, q_2) \in Q | q_1 \in \|Mr_1\|$ *and* $q_2 \in \|Mr_2\|\}$
$\|T\| := \{((q_1, q_2), \sigma, (q_1', q_2)) | (q_1, \sigma, q_1') \in \|T_1\|\} \cup \{((q_1, q_2), \sigma, (q_1, q_2')) | (q_2, \sigma, q_2') \in \|T_2\|\}$

The corresponding algorithm is obtained directly from the definition. Its time complexity is $O(|Q_1| card(\|T_2\|) + card(\|T_1\|)|Q_2|)$.

Definition 4 ***Exact synchronous product***

Let M_1 *and* M_2, *two FSMs as in the previous definition with* $\Sigma_1 \cap \Sigma_2 \neq \emptyset$. *The exact synchronous product* $M = M_1 \S M_2 = \{Q, \Sigma, \|T\|, \|I\|, \|Mr\|\}$ *is defined as:*

$Q := \{Q_1 \times Q_2\}$
$\Sigma := \Sigma_1 \cap \Sigma_2$
$\|I\| := \{(q_1, q_2) \in Q | q_1 \in \|I_1\|$ *and* $q_2 \in \|I_2\|\}$
$\|Mr\| := \{(q_1, q_2) \in Q | q_1 \in \|Mr_1\|$ *and* $q_2 \in \|Mr_2\|\}$
$\|T\| := \{((q_1, q_2), \sigma, (q_1', q_2')) | (q_1, \sigma, q_1') \in \|T_1\|$ *and* $(q_2, \sigma, q_2') \in \|T_2\|\}$

The algorithm is obtained in a straightforward fashion from the definition. The time complexity is $O(max(card(\|T_1\|), card(\|T_2\|)))$.

2.3 Fix-point operator for supremal controllable sublanguage

Let $P = \{Q, \Sigma, \|T_P\|, \|I_P\|, \|Mr_P\|\}$ and $E = \{X, \Sigma, \|T_E\|, \|I_E\|, \|Mr_E\|\}$ the plant and specification FSMs. The starting point for calculating the supremal controllable language can be the synchronous product of these two FSMs

$PE = P\S E = \{Q \times X, \Sigma, \|T_{PE}\|, \|I_{PE}\|, \|Mr_{PE}\|\}$ with
$\|I_{PE}\| := \{(q, x) \in Q \times X | q \in \|I_P\|$ and $x \in \|I_E\|\}$
$\|Mr_{PE}\| := \{(q, x) \in Q \times X | q \in \|Mr_P\|$ and $x \in \|Mr_E\|\}$
$\|T_{PE}\| := \{((q, x), \sigma, (q', x')) | (q, \sigma, q') \in \|T_P\|$ and $(x, \sigma, x') \in \|T_E\|\}$

Thus, the transition relation $\|T_{PE}\|$ is a function with domain $W := Q \times X \times \Sigma \times Q \times X$. Now, let $\|T_Z\|$ be a transition relation defined on the same domain. $\|T_Z\|$ establishes the relation between a plant and specification states. Using this predicate representation, a slightly rewritten version of the fix-point operator $\Omega : 2^W \longrightarrow 2^W$ proposed by [5] is:

$\Omega(\|T_Z\|) := \|T_{PE}\| \cap \|T_Z\| -$
$\{((q, x), \sigma_u, (q_1, x_1)) \in \{T_Z\} | (q, x) \in K_1, \sigma_u \in \Sigma_u\} -$
$\{((q, x), \sigma, (q_1, x_1)), ((q_1, x_1), \sigma, (q, x))$
$\in \|T_Z\| | (q, x) \notin \|Co\|\}$

where

$K_1 := \{(q, x) | \exists (q, \sigma_u, q_2) \in \|T_P\|, \sigma_u \in \Sigma_u$ and
$((q, x), \sigma_u, (q_2, x_2)) \notin \|T_Z\|$

K_1 is the set of state pairs (q, x) from which at least one uncontrollable transition is enabled from q in the process FSM and is not enabled in the equivalent state of $\|T_Z\|$. $\|Co\|$ guarantees that all states in each iteration are coreachable.

It can be demonstrated that this fix–point operator produces the supremal controllable language [11]. The FSM thus obtained is

$$Z := \{Q_Z, \Sigma, \|T_Z\|, \|I_Z\|, \|Mr_Z\|\} \tag{4}$$

with
$Q_Z := \{(q, x) | ((q, x), \sigma, (q', x'))$ or
$((q', x'), \sigma, (q, x)) \in \|T_Z\|\}$
$\|I_Z\| := \{(q, x) \in Q_Z | (q, x) \in \|I_{PE}\|\}$
$\|Mr_Z\| := \{(q, x) \in Q_Z | (q, x) \in \|Mr_{PE}\|\}$

The corresponding pseudo-code follows.

```
S = S_ant =
(*BDDTransitionRelationSystemSpec);
do {
  S_ant = S;
/* Calculate the set of states in
   the transition relation */
  cstates = bdd_exist(S,
            (*BDDSetNextSystemSpec)
            & (*BDDSetTransition));
  nstates = bdd_exist(S,
```

```
            (*BDDSetCurrentSystemSpec)
            & (*BDDSetTransition));
  states  = cstates |
            bdd_replace(nstates,
            changePairSystemSpecF);

/* Calculate the uncontrollable
   transitions from all the states */
  uncontSys = states &
     (*BDDTransitionRelationSystem)
     & (*BDDUncontrollable);
    uncontS   = states & S
                & (*BDDUncontrollable);
    uncontSys = bdd_exist(uncontSys,
                (*BDDSetNextSystemSpec));
    uncontS   = bdd_exist(uncontS,
                (*BDDSetNextSystemSpec));

    /* Calculate the problematic
       transitions */
    badTransitions = uncontSys -
                     uncontS;

    /* Calculate the problematic states */
    badStates = bdd_exist(badTransitions,
                (*BDDSetTransition));

    /* Erase uncontrollable states */
    S = S & !badStates;
    badStates = bdd_replace(badStates,
                changePairSystemSpecB);
    S = S & !badStates;

    /* Set co-reachability */
    co_ReachableStates =
       Get_co_ReachableStates(S,
       (*BDDMarkedStatesSystemSpec),
       (*BDDSetCurrentSystemSpec),
       (*BDDSetNextSystemSpec),
       (*BDDSetTransition));
    S = S & co_ReachableStates;
} while (S != S_ant);

   /* TRIM the superstructure controller */
   ReachableStates =
      GetReachableStates(S,
      (*BDDInitialStateSystemSpec),
      (*BDDSetCurrentSystemSpec),
      (*BDDSetNextSystemSpec),
      (*BDDSetTransition));
```

3 Benchmarking

3.1 Asynchronous Product

The benchmark for the asynchronous product operations is taken from [12]. It consists of a tank that is pressurized to a set (i.e. normal) value using a solenoid valve operated by an on-off button. Pressure is measured using a sensor with three states (i.e. low, normal and high). The objective is to synthesize a control device to supervise the proper operation of the tank. The process components and their associated FSM models are shown in figure 1. The asynchronous product of the three components is used as the basis to obtain the process model. The size of the process is increased by including extra tank blocks in parallel. Thus, space size of the final result increases in powers of 12. The results for memory usage and CPU time required are shown in table 1.

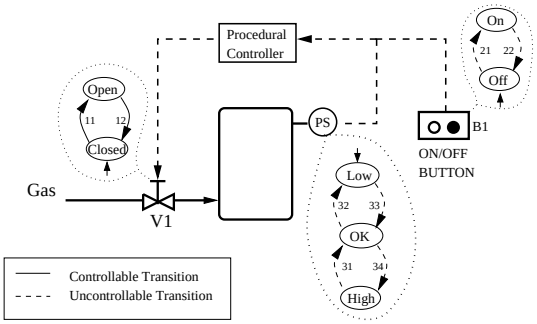
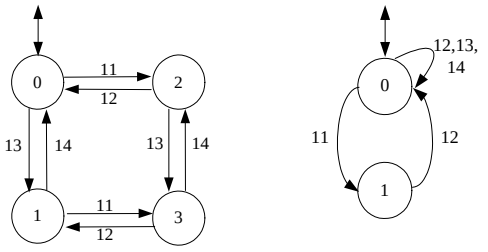


Figure 1: Pressurized tank system

Minimum memory quotas for SSPC and Supremica are due to software implementation. Regarding SSPC, table 1 also presents the number of BDD-nodes vs. number of states of the resultant FSM. Notice that in this case the relation is linear.

3.2 Exact synchronous Product

FSMs M_1 and M_2 shown in figure 2 conform the first block of the benchmark FSMs for the exact synchronous product. Subsequent blocks are built by duplicating M_1 and M_2 of the previous block but with different transition labels and then performing the asynchronous product between duplicated FSMs. Table 2 shows the results for this operation and the number of BDD-nodes vs. number of states of the resultant FSM.



(a) M1 (b) M2

Figure 2: Basic FSMs for the benchmark of the exact synchronous product

3.3 Supremal controllable sublanguage

The example used is taken from [16]. It consists of a production line with two machines $M1$, $M2$ and a test unit connected by two one-part buffers $B1$, $B2$ as shown in figure 3. The controllable transitions are the inputs to machines or test units. Once a part is in the test unit, it can be accepted or rejected and reprocessed by returning to buffer $B1$. Initially, the machines and test units are idle and buffers are empty. A cycle is completed when the production line returns to its initial state. The synthesis objective is to find a supervisor that guarantees proper operation of the buffers (not underflow or overflow). The FSM models for the line transfer components and the specification are shown in figures 4 and 5, respectively. Uncontrollable transitions are depicted as shadow arrows. The FSM candidate for supervisor is built by first performing the asynchronous product of all components and then synchronizing with the specification FSM. In order to test the effect of size, production lines were added working in parallel with no relation among them. Table 3 shows the results for memory usage and CPU time considering all states as marked.

4 Conclusions

Results show a considerable better performance of the symbolic implementations when compared with their explicit counterpart in the synchronous and asynchronous products. In the case of the computations of supremal controllable languages, the difference was of up to four orders of magnitude for this particular example. Neither variable reordering, modularity or incremental solutions were exploited. Thus, it is expected a much better performance once these aspects are considered.

Acknowledgements.
The authors wish to express their gratitude to Dr. Alan Hu for his generous advice. AS kindly acknowledges partial financial support from Conacyt, Mexico in the form of a sabbatical fellowship and project 365935U

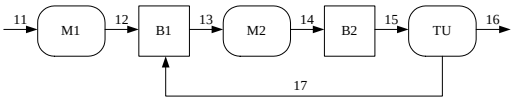
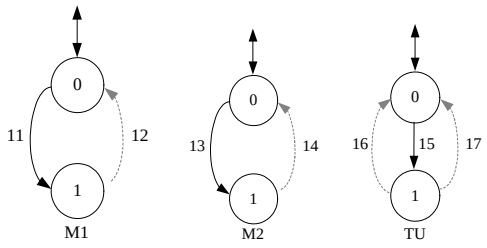


Figure 3: Production line system



(a) M1 (b) M2 (c) TU

Figure 4: Production line components

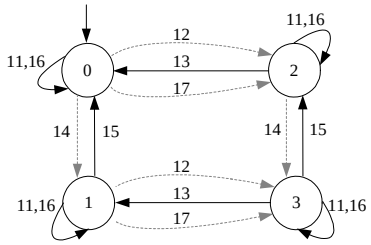


Figure 5: Production line specification (all states are marked)

References

[1] K. Akesson, H. Flordal, and M. Fabian, *Exploiting modularity for synthesis and verification of supervisors*, 15th IFAC World Congress. Barcelona, Spain, July 2002.

[2] R. D. Brandt, V. Garg, R. Kumar, F. Lin, S. Marcus, and W. M. Wonham, *Formulas for calculating supremal controllable and normal sublanguages*, Syst. and Cont. Lett. **15** (1990), 111–117.

[3] C. G. Cassandras and S. Lafortune, *Introduction to discrete event systems*, Kluwer Academic, 1999.

[4] V. Chandra, B. Oruganti, and R. Kumar, *Ukdes: A graphical software tool for the design, analysis and control of discrete event systems*, IEEE Transactions on Control Systems Technology, Submitted, 2000.

[5] G. Hoffmann and H. Wong–Toi, *Symbolic synthesis of supervisory controllers*, Proc. of the 1992 American Control Conference, June 24-26 1992, pp. 2789–2793.

[6] A. J. Hu, *Techniques for efficient formal verification using binary decision diagrams*, Technical Report CS-TR-95-1561, Stanford University Department of Computer Science, 1995.

[7] R. Kumar and V. K. Garg, *Modeling and control of logical discrete event systems*, Kluwer Academic Publisher, 1995.

[8] Y. Li and W. M. Wonham, *Control of vector discrete–event systems. ii.– Controller synthesis*, IEEE Transactions on Automatic Control **39** (1994), no. 3, 512–531.

[9] J. L. Nielsen, *Buddy: Binary decision diagram package, release 2.0*, IT University of Copenhagen (ITU), 2001.

[10] P. J. Ramadge and W. M. Wonham, *Supervisory control of a class of discrete–event processes*, SIAM Journal of Control and Optimization **25** (1987), no. 1, 206–230.

[11] J. Reza, *Sintesis de controladores de procedimientos utilizando tecnicas de calculo simbolico (in spanish)*, Master's thesis, Cinvestav, 2002.

[12] A. Sanchez, G. Rotstein, N. Alsop, and S. Macchietto, *Synthesis and implementation of procedural controllers for event–driven operations*, AIChE Journal **45** (1999), no. 8, 1753–1775.

[13] J. N. Tzitziklis, *On the control of discrete–event dynamical systems*, Math. Control Signal Systems **2** (1989), 95–107.

[14] W. M. Wonham and P. J. Ramadge, *On the supremal controllable sublanguage of a given language*, SIAM Journal of Control and Optimization **25** (1987), no. 3, 637–659.

[15] Z. Zhang, *personal communication*, 2002.

[16] Z. Zhang and W. M. Wonham, *Synthesis and control of discrete-event systems*, ch. STCT: An efficient algorithm for supervisory control design, Kluwer Academic, 2002.

		Memory Usage, Mb			CPU Time, s			# nodes
# blocks	$\|Q\|$	SSPC	UKDES	Suprm	SSPC	UKDES	Suprm	SSPC
1	12	< 13	< 1	0.3	< 0.01	$<< 1$	0.2	62
3	1,728	< 13	24	7	1	< 0.01	2.2	510
4	20,736	< 13	300	108	0.01	375	19.1	878
5	248,832	13.5	N/D	N/A	0.05	N/A	N/A	1,382
10	6.19×10^{10}	16	N/A	N/A	0.05	N/A	N/A	5,366
20	3.83×10^{21}	19	N/A	N/A	0.45	N/A	N/A	21,134
40	1.46×10^{43}	26	N/A	N/A	3.5	N/A	N/A	82,800

Table 1: Memory usage in Mb, CPU time in seconds and number of nodes for asynchronous product

				Memory Usage, Mb			CPU Time, s			# nodes
# blocks	$\|Q_1\|$	$\|Q_2\|$	$\|Q_{SYNC}\|$	SSPC	UKDES	Suprm	SSPC	UKDES	Suprm	SSPC
1	4	2	4	< 13	< 1	0.06	< 0.01	$<< 1$	0.04	26
3	64	8	64	< 13	< 1	0.88	<0.01	< 1	0.3	481
5	1,024	32	1,024	13.5	17	5	0.02	2	1.2	3,156
7	16,384	128	16,384	14.5	288	68	0.2	650	43	16,581
8	65,536	256	65,536	17	N/A	N/A	0.8	N/A	N/A	36,765
10	1.04×10^6	1024	1,048,576	35	N/A	N/A	12	N/A	N/A	174,673
12	1.67×10^7	4096	1.67×10^7	130	N/A	N/A	86	N/A	N/A	806,402
14	2.68×10^8	16384	2.68×10^8	400	N/A	N/A	736	N/A	N/A	3,653,044

Table 2: Memory usage in Mb, CPU time in seconds and number of nodes for exact synchronous product.

				CPU Time, s			Memory Usage, Mb			# nodes
# blocks	$\|Q_P\|$	$\|Q_E\|$	$\|Q_{SUP}\|$	SSPC	UKDES	Suprm	SSPC	UKDES	Suprm	SSPC
1	8	4	10	< 0.01	< 0.01	0.07	< 13	< 1	0.3	76
2	64	16	100	0.02	1	0.9	< 13	6.6	2.8	665
3	512	64	1,000	0.2	3,302	1,524	13.5	88	98	3,830
4	4,096	256	10,000	1.7	N/A	N/A	14.5	N/A	N/A	19,347
5	32,768	1,024	100,000	12	N/A	N/A	21.5	N/A	N/A	92,445
6	262,144	4,096	1,000,000	112	N/A	N/A	55	N/A	N/A	428,275
7	2,097,152	16,384	10,000,000	933	N/A	N/A	227	N/A	N/A	1,944,427

Table 3: Memory usage in Mb, CPU time in seconds and number of nodes for the calculation of the supremal controllable sublanguage.