Component-Based Synthesis by Solving Language Equations

Tiziano Villa[1]
joint work with Nina Yevtushenko[2], Alex Petrenko[3],
Robert Brayton[4], Alan Mishchenko[4],
A. Sangiovanni-Vincentelli[4], and more ...

[1] Department of Computer Science, University of Verona, Italy
[2] Department of Radiophysics, Tomsk State University, Russia
[3] CRIM, Montreal, Canada
[4] Department of EECS, UC Berkeley, USA

EPFL Workshop on Logic Synthesis and Verification,
December 10-11, 2015

Outline

The problem of synthesizing an unknown component

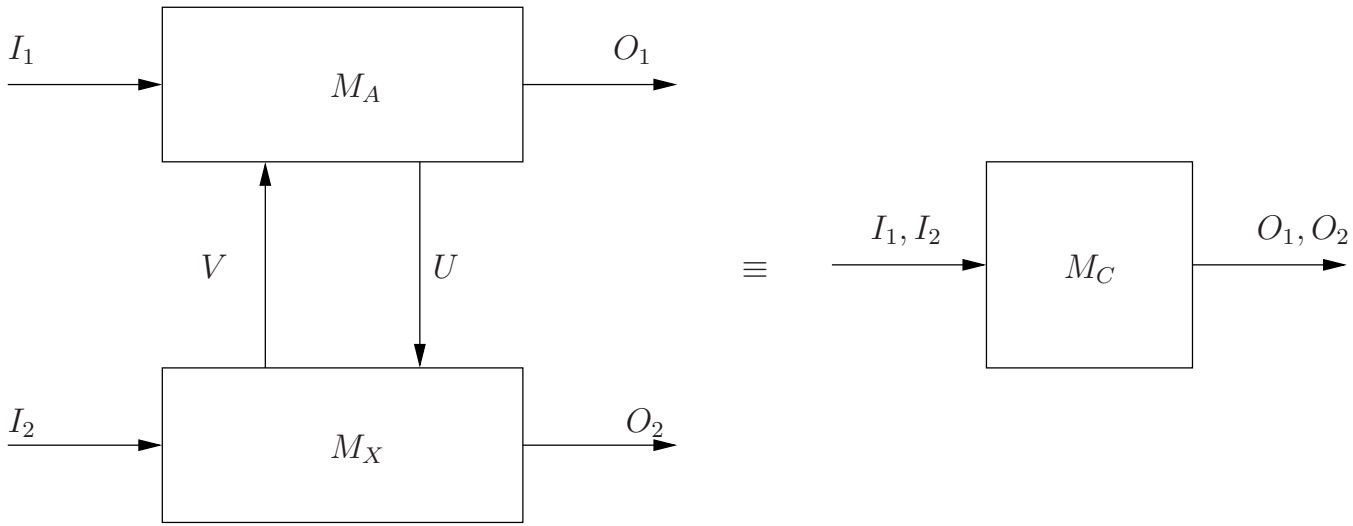
Problem: finding the unknown component

Design a component that combined with a known part of a system (called the **context** or **plant**) conforms to a given **specification**.

The problem of synthesizing an unknown component

Problem: finding the unknown component

Design a component that combined with a known part of a system (called the **context** or **plant**) conforms to a given **specification**.



How to formalize the problem

How to model the system and its components

Associate languages to systems and components: traces of events over alphabets of internal and external signals.

How to formalize the problem

How to model the system and its components

Associate languages to systems and components: traces of events over alphabets of internal and external signals.

How to model composition

- Interleaving parallel composition (a.k.a.: parallel composition, synchronous parallel composition, asynchronous composition)
- Synchronous composition

How to formalize the problem

How to model the system and its components

Associate languages to systems and components: traces of events over alphabets of internal and external signals.

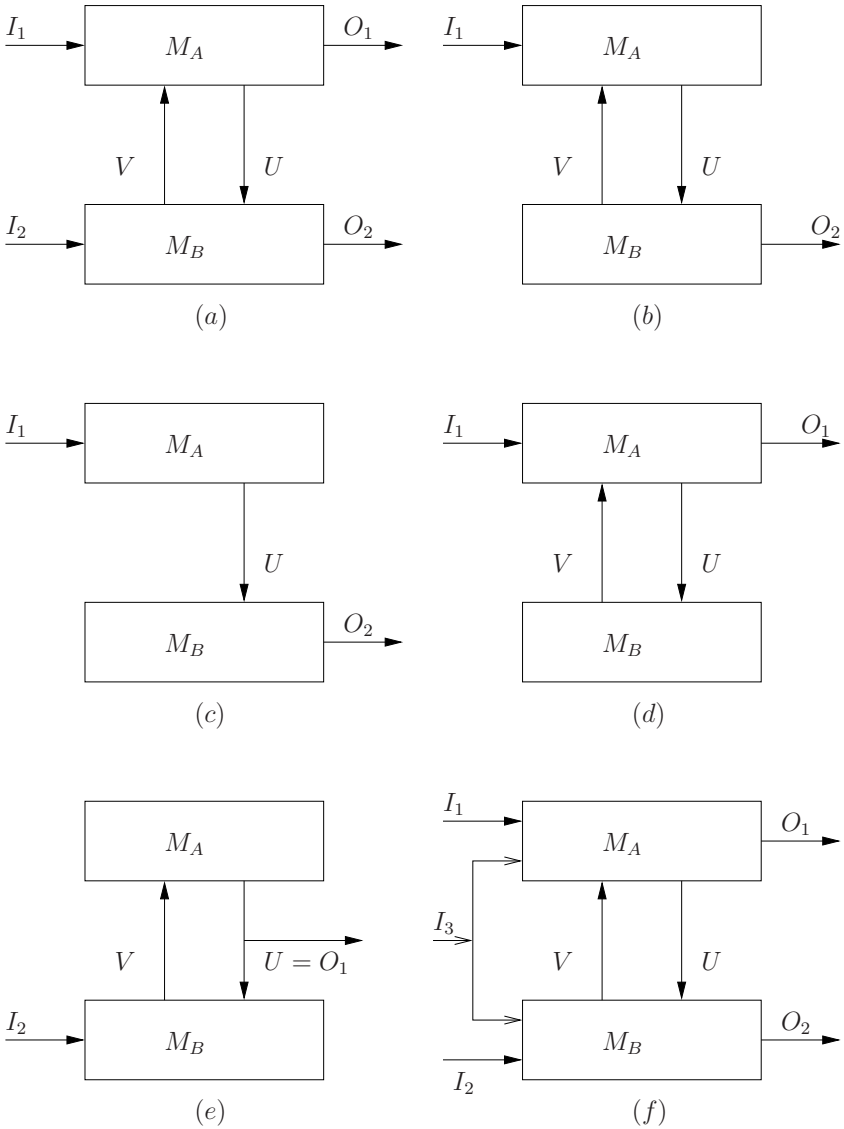
How to model composition

- Interleaving parallel composition (a.k.a.: parallel composition, synchronous parallel composition, asynchronous composition)
- Synchronous composition

How to model conformance

- Language containment
- Simulation/Bisimulation relation

Composition topologies



(a) General topology
(b) 2-way cascade topology
(c) 1-way cascade topology
(d) Rectification topology
(e) Supervisory control topology
(f) Variant of general topology

The unknown component problem

Problem: finding the unknown component

Design a component that combined with a known part of a system (called the **context** or **plant**) conforms to a given **specification**.

The unknown component problem

Problem: finding the unknown component

Design a component that combined with a known part of a system (called the **context** or **plant**) conforms to a given **specification**.

Solution: solving an equation over languages

Reduce the problem to solving abstract equations over languages under synchronous and interleaving parallel composition.

Sequential synthesis

Synthesis and resynthesis of a hardware component

- Kim-Newborn
- Yevtushenko et al.
- H.-Y. Wang-Brayton
- Watanabe-Brayton

Sequential synthesis

Synthesis and resynthesis of a hardware component

- Kim-Newborn
- Yevtushenko et al.
- H.-Y. Wang-Brayton
- Watanabe-Brayton

WS1S

- Büchi
- Thatcher-Wright
- A. Aziz-Brayton et al.

Synthesis of discrete controllers

Supervisory control

- Wonham-Ramadge
- Overkamp
- Kumar et al.

Synthesis of discrete controllers

Supervisory control

- Wonham-Ramadge
- Overkamp
- Kumar et al.

Model matching of finite state machines

- Khatri-Brayton-Sangiovanni Vincentelli et al.
- Di Benedetto-Sangiovanni Vincentelli et al.
- Lafortune et al.

Synthesis of automata and process algebras

Submodule specification for communication protocols

- Bochmann et al.
- Petrenko-Yevtushenko
- Haghverdi-Ural
- Drissi

Synthesis of automata and process algebras

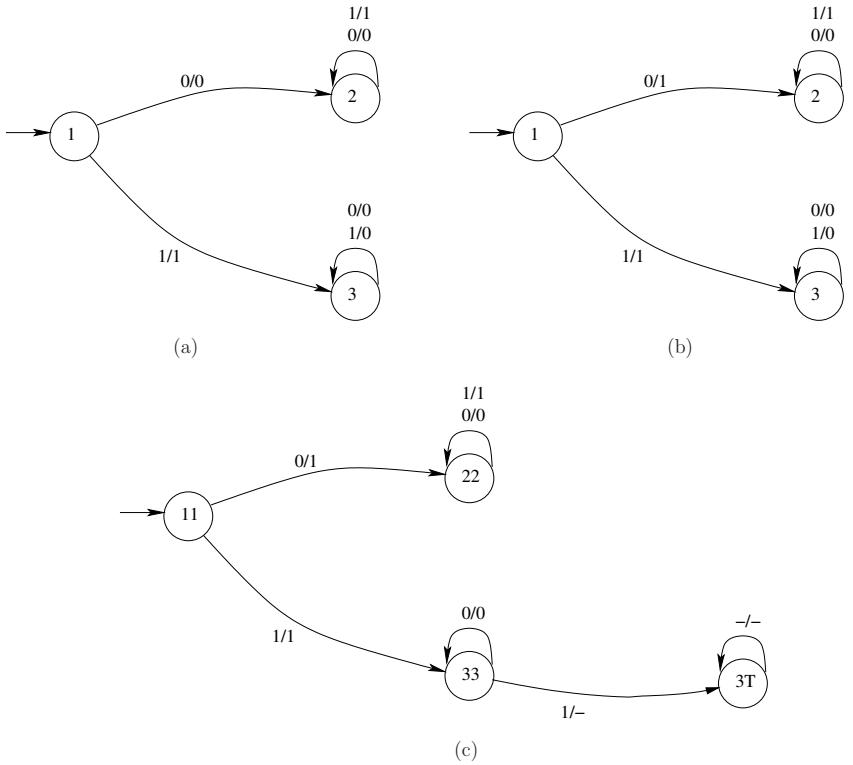
Submodule specification for communication protocols

- Bochmann et al.
- Petrenko-Yevtushenko
- Haghverdi-Ural
- Drissi

Process algebra

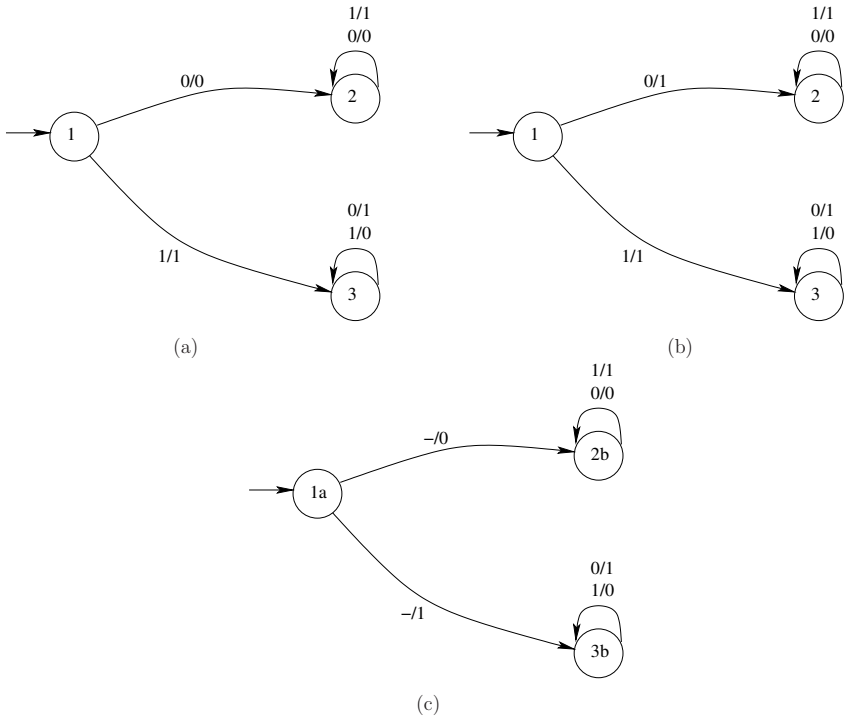
- Qin-Lewis
- Negulescu

Series topology: input don't care sequences



(a) Head FSM; (b) Tail FSM; (c) Flexibility at tail FSM
Solved by Kim-Newborn

Series topology: output don't care sequences



(a) Head FSM; (b) Tail FSM; (c) Flexibility at head FSM
Solved by Yevtushenko

WS1S and regular languages

- Weak Second-Order Logic of 1 Successor (WS1S) is a logic with the same expressive power as regular languages.
- It is possible to represent in WS1S any regular language L by encoding with a formula in WS1S the finite automaton A that recognizes the regular language L.

Theorem (Thatcher-Wright, 1968)

$L \subseteq (\{0,1\}^k)^\star$ is regular iff there exists a WS1S formula ϕ with $X_1, \ldots, X_k$ as free variables and $\mathcal{L}(\phi) = L$.

- The WS1S formalism allows to write down formulas that express the permissible behaviours at a node of a network of FSMs.

Synthesis with WS1S

1-Way Cascade (a) - case (c)

$$\phi^{M_A{}^*}(I_1, U) = (\forall O_2)[\phi^{M_B}(U, O_2) \rightarrow \phi^{M_C}(I_1, O_2)].$$

The machine M_A^* is the one produced by the construction of Kim and Newborn.

1-Way Cascade (b) - case (c)

$$\phi^{M_B{}^*}(U, O_2) = (\forall I_1)[\phi^{M_A}(I_1, U) \rightarrow \phi^{M_C}(I_1, O_2)].$$

Supervisory Control - case (e)

$$\phi^{M_B{}^*}(I_2, O_1, V) = \phi^{M_A}(V, O_1) \rightarrow \phi^{M_C}(I_2, O_1).$$

2-Way Cascade (a) - case (b)

$$\phi^{M_A{}^*}(I_1, V, U) = (\forall O_2)[\phi^{M_B}(U, V, O_2) \rightarrow \phi^{M_C}(I_1, O_2)].$$

2-Way Cascade (b) - case (b)

$$\phi^{M_B{}^*}(U, V, O_2) = (\forall I_1)[\phi^{M_A}(I_1, V, U) \rightarrow \phi^{M_C}(I_1, O_2)].$$

Rectification (a) - case (d)

$$\phi^{M_B{}^*}(U, V) = (\forall I_1, O_1)[\phi^{M_A}(I_1, V, U, O_1) \rightarrow \phi^{M_C}(I_1, O_1)].$$

Rectification (b) - case (d)

$$\phi^{M_A{}^*}(I_1, V, U, O_1) = \phi^{M_B}(U, V) \rightarrow \phi^{M_C}(I_1, O_1).$$

Synchronous and interleaving parallel composition

Synchronous composition

Synchronous composition ($\bullet$) corresponds to instantaneous communication of systems.

Synchronous and interleaving parallel composition

Synchronous composition

Synchronous composition ($\bullet$) corresponds to instantaneous communication of systems.

Interleaving parallel composition

Interleaving parallel composition ($\diamond$) corresponds to asynchronous communication allowing arbitrary delay between communication events. A slow environment is assumed, i.e., no external input is applied to the composition until it produces an external output to the previous external input.

Synchronous operators

Projection

Given a language L over alphabet $X \times V$, consider the homomorphism $p : X \times V \to V^\star$ defined as $p((x, v)) = v$, then the language

$$L_{\downarrow V} = \{p(\alpha) | \alpha \in L\}$$

over alphabet V is the **projection** of language L to alphabet V, or V-projection of L. By definition of substitution $p(\epsilon) = \epsilon$.
The projection of a language L over alphabet $X \times V$ to the alphabet V is the set of words obtained from those in L by replacing the symbols (x, v) by their second component v.

Synchronous operators - 2

Lifting

Given a language L over alphabet X and another alphabet V, consider the substitution $I : X \to 2^{(X \times V)^*}$ defined as $I(x) = \{(x, v) | v \in V\}$, then the language

$$L_{\uparrow V} = \{I(\alpha) | \alpha \in L\}$$

over alphabet $X \times V$ is the **lifting** of language L to alphabet V, or V-lifting of L. By definition of substitution $I(\epsilon) = \{\epsilon\}$.
The lifting of language L over alphabet X to the alphabet V (V disjoint from X) is the set of words obtained from those in L by replacing the symbols $x \in X$ by the pairs (x, v) for all $v \in V$.

Synchronous operators - 3

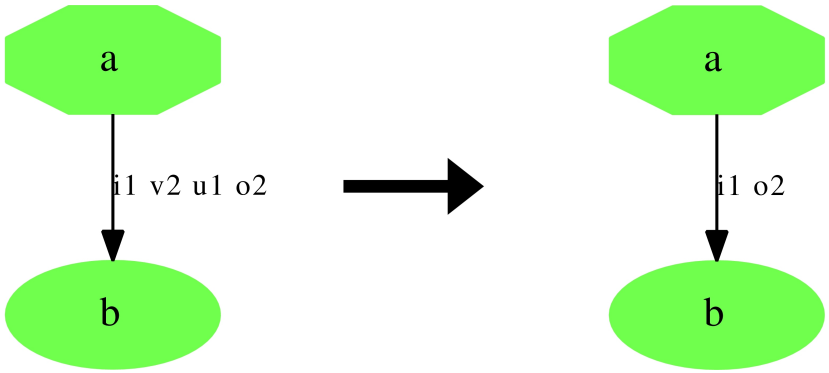


Figure: $AUT_{\downarrow(IO)}$

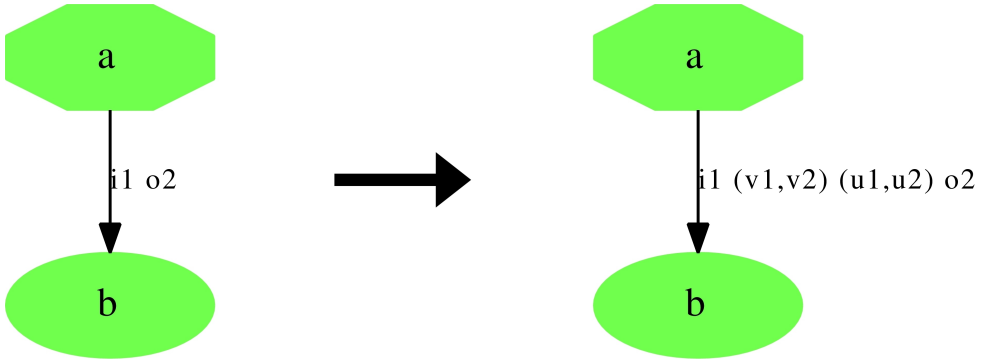


Figure: $AUT_{\uparrow(VU)}$

Synchronous language composition

Synchronous composition

Given alphabets I, U, O, language L_1 over $I \times U$ and language L_2 over $U \times O$, the **synchronous composition** of languages L_1 and L_2 is the language defined over $I \times O$

$$L_1 \bullet L_2 = [(L_1)_{\uparrow O} \cap (L_2)_{\uparrow I}]_{\downarrow I \times O}.$$

Given two systems A and B with associated languages $L(A)$ and $L(B)$ communicating internally by channel U and externally by channels I and O, their synchronous composition is defined by the operator of synchronous language composition.

Interleaving parallel operators

Restriction

Given a language L over alphabet $X \cup V$, consider the homomorphism $r : X \cup V \to V^\star$ defined as

$$r(y) = \begin{cases} y \text{ if } y \in V \\ \epsilon \text{ if } y \in X \backslash V \end{cases}$$ then the language

$$L_{\Downarrow V} = \{r(\alpha) | \alpha \in L\}$$

over alphabet V is the **restriction** of language L to alphabet V, or V-restriction of L, i.e., words in $L_{\Downarrow V}$ are obtained from those in L by deleting all the symbols in X that are not in V. By definition of substitution $r(\epsilon) = \epsilon$.

The restriction of language L over alphabet $X \cup V$ to the alphabet V is the set of words obtained from those in L by deleting the symbols in X that are not in V.

Interleaving parallel operators - 2

Expansion

Given a language L over alphabet X and an alphabet V disjoint from X, consider the mapping $e : X \to 2^{(X \cup V)^*}$ defined as $e(x) = \{\alpha x \beta | \alpha, \beta \in (V \backslash X)^*\}$, then the language

$$L_{\Uparrow V} = \{e(\alpha) | \alpha \in L\}$$

over alphabet $X \cup V$ is the **expansion** of language L to alphabet V, or V-expansion of L, i.e., words in $L_{\Uparrow V}$ are obtained from those in L by inserting anywhere in them words from $(V \backslash X)^*$. Notice that e is not a substitution and that $e(\epsilon) = \{\alpha | \alpha \in (V \backslash X)^*\}$. The expansion of language L over alphabet X to the alphabet V (V disjoint from X) is the set of words obtained from those in L by inserting anywhere in them any word from V^*.

Interleving parallel operators - 3

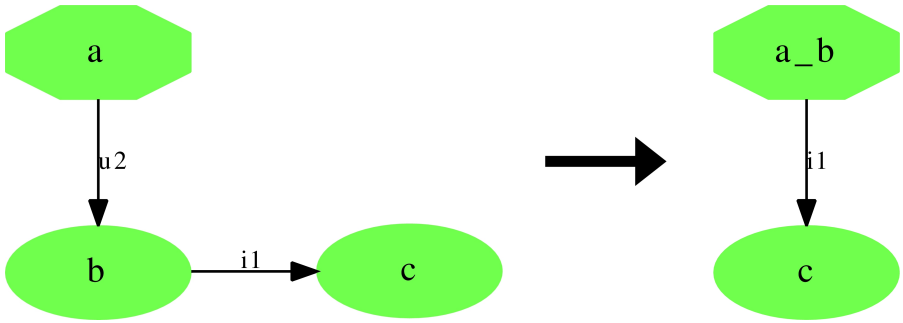


Figure: $AUT_{\Downarrow(IO)}$

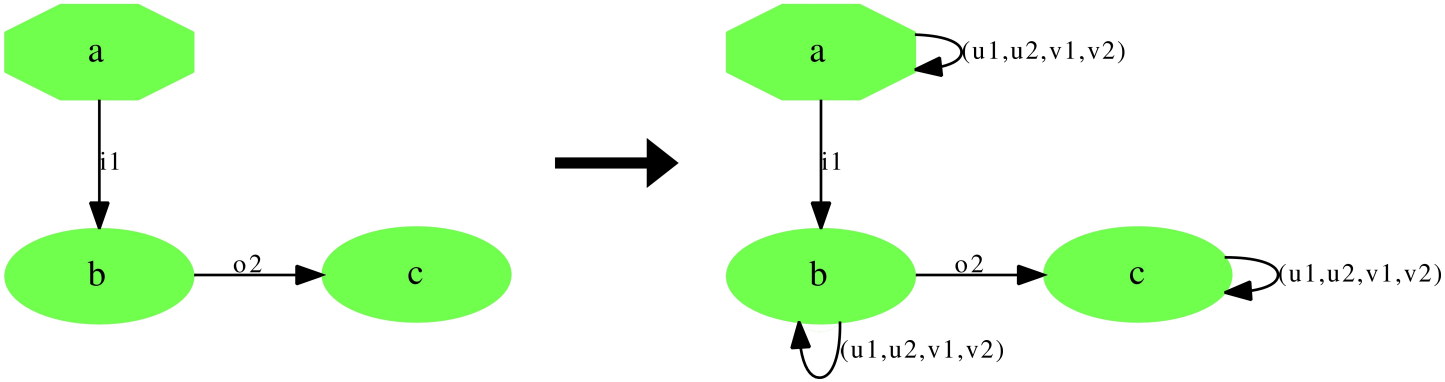


Figure: $AUT_{\Uparrow(UV)}$

Interleaving parallel language composition

Interleaving parallel composition

Given alphabets I, U, O, language L_1 over $I \cup U$ and language L_2 over $U \cup O$, the **interleaving parallel composition** of languages L_1 and L_2 is the language defined over $I \cup O$

$$L_1 \diamond L_2 = [(L_1)_{\Uparrow O} \cap (L_2)_{\Uparrow I}]_{\Downarrow I \times O}.$$

Given two systems A and B with associated languages $L(A)$ and $L(B)$ communicating internally by channel U and externally by channels I and O, their interleaving parallel composition is defined by the operator of interleaving parallel language composition.

Abstract composition operators

Abstract composition

Given the disjoint alphabets I, U, O, a language L_1 over alphabet $I \circ U$ and a language L_2 over alphabet $U \circ O$, let the following symbols represent operators defined over alphabets and languages:

- $\circ$ denotes an **operator between alphabets**;
- $\odot$ denotes a **composition operator between languages**, defined as

$$L_1 \odot_{I \circ O} L_2 = [(L_1)_{\top O} \cap (L_2)_{\top I}]_{\bot I \circ O}$$

where $\top$ and $\bot$ denote language substitutions operators in suffix notation.
E.g., $\top$ is $\uparrow$, $\bot$ is $\downarrow$, and so $\odot$ denotes the synchronous composition operator $\bullet$, and $\circ$ denotes $\times$ (or $\top$ is $\Uparrow$, $\bot$ is $\Downarrow$, and so $\odot$ denotes the parallel composition operator $\diamond$ and $\circ$ denotes $\cup$).

Abstract equations over languages

Abstract equations

Given the disjoint alphabets I, U, O, a language A over alphabet $I \circ U$ and a language C over alphabet $I \circ O$, we define the **language inequation**

$$A_{I \circ U} \odot_{I \circ O} X_{U \circ O} \subseteq C_{I \circ O}, \text{ or } A \odot_{I \circ O} X \subseteq C,$$

and the **language equation**

$$A_{I \circ U} \odot_{I \circ O} X_{U \circ O} = C_{I \circ O}, \text{ or } A \odot_{I \circ O} X = C,$$

with respect to the unkown language X over alphabet $U \circ O$.

Solutions of abstract equations over languages

Abstract equations

Given the disjoint alphabets I, U, O, a language A over alphabet $I \circ U$ and a language C over alphabet $I \circ O$, language B over alphabet $U \circ O$ is called a **solution** of the inequation $A \odot_{I \circ O} X \subseteq C$ iff $A \odot_{I \circ O} B \subseteq C$.

A solution is called the **largest solution** if it contains any other solution.

$B = \emptyset$ is the trivial solution.

Closed-form solution of abstract inequations over languages

Theorem

If the substitution operators $\top$ and $\bot$ are such that

- $H1$: Given disjoint alphabets Z, Y and language L over Z, $(L_{\top Y})_{\bot Z} = L$
- $H2$: Given disjoint alphabets Z, Y and languages L_1, L_2 over $Y \circ Z$, if $L_1 = (L_{1\bot Z})_{\top Y}$ or $L_2 = (L_{2\bot Z})_{\top Y}$ then $(L_1 \cap L_2)_{\bot Z} = L_{1\ \bot Z} \cap L_{2\ \bot Z}$
- $H3$: Given disjoint alphabets Z, Y and language L over $Y \circ Z$, $L_{\bot Z} = \emptyset \Leftrightarrow L = \emptyset$

then the **largest language solution** of the inequation $A \odot X \subseteq C$ is

$$S = \overline{A \odot \overline{C}}.$$

S contains every language that is a solution (also the empty one). Any language contained in S is a solution of the language inequation.

Closed-form solution of abstract equations over languages

Corollary

If $S \odot \overline{A \odot \overline{C}} = C$, then S is the largest solution of the language equation $A \odot X = C$. A subset of S may not be a solution of the language equation.

If $S \odot \overline{A \odot \overline{C}} \subset C$, then the language equation is unsolvable and the language $D = S \odot \overline{A \odot \overline{C}}$ is the largest subset of C such that the language equation $A \odot X = D$ is solvable.

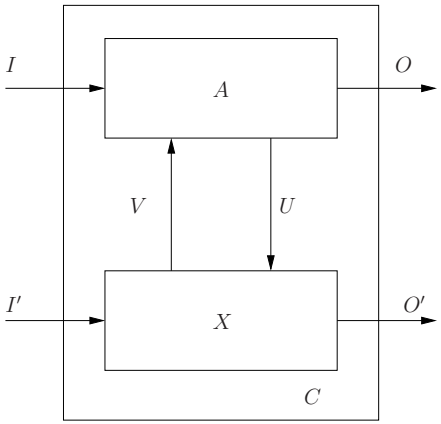
Concrete equations over languages ($\diamond$ composition)

Given the alphabets I, I', V, U, O, O', a language A over alphabet $I \cup V \cup U \cup O$, and a language C over alphabet $I \cup I' \cup O \cup O'$, consider the language inequality (equation)

$$A \diamond X \subseteq C \qquad (A \diamond X = C)$$

The language B over alphabet $I' \cup U \cup V \cup O'$ is called a **solution** of the inequality (equation) $A \diamond X \subseteq C$ $(A \diamond X = C)$ iff $A \diamond B \subseteq C$ $(A \diamond B = C)$. The largest language solution is computed by

$$X = \overline{A \diamond \overline{C}}.$$



Specializing the solutions

Restricted solutions

One may restrict the solutions to those that satisfy a required property (prefix-closed, progressive, compositionally progressive etc.). Algorithms are available for regular languages.

Specializing the solutions

Restricted solutions

One may restrict the solutions to those that satisfy a required property (prefix-closed, progressive, compositionally progressive etc.). Algorithms are available for regular languages.

Optimal solutions

One may extract a particular solution that optimizes a given cost function (e.g., the number of states of an automaton representing it, size of logical implementation etc.). It is mostly an open problem.

Domains of application

This approach can be applied to many fields:

- conversion between mismatching protocols;
- supervisory control with full controllability and observability;
- testing modules of a discrete event system;
- design of winning stategies for logic games;
- logic circuit optimization;
- etc.

How to solve effectively the language equations

Computing the closed-form solution

Automata are operational counterpars of languages. The closed-form largest solution can be computed if the operations appearing in it can be perfomed on the related automata. Special attention is required by the complementation operator.

How to solve effectively the language equations

Computing the closed-form solution

Automata are operational counterpars of languages. The closed-form largest solution can be computed if the operations appearing in it can be perfomed on the related automata. Special attention is required by the complementation operator.

Classes of languages equations

Most of the work done refers to regular languages to which correspond finite automata and finite state machines.
Language equations for special classes of Büchi automata and Petri nets have been studied too.

Representations of regular languages

Finite automata

Finite automata are the most natural representation for regular languages.

Representations of regular languages

Finite automata

Finite automata are the most natural representation for regular languages.

FSMs

Many problems derived from practical industrial applications are described as FSMs.

Representations of regular languages

Finite automata

Finite automata are the most natural representation for regular languages.

FSMs

Many problems derived from practical industrial applications are described as FSMs.

Conversion FSM $\Longleftrightarrow$ automaton

Using a specialized procedure it's possible to convert a FSM into an automaton and vice versa

Conversion FSM $\rightarrow$ automaton wrt parallel composition

For a language over $I \cup O$, the automaton is obtained from the original FSM, by replacing each edge (i, s, s', o) by the pair of edges $(i, s, (s, i))$ and $(o, (s, i), s')$, where (s, i) is a new node (non-accepting state). All original states are made accepting.

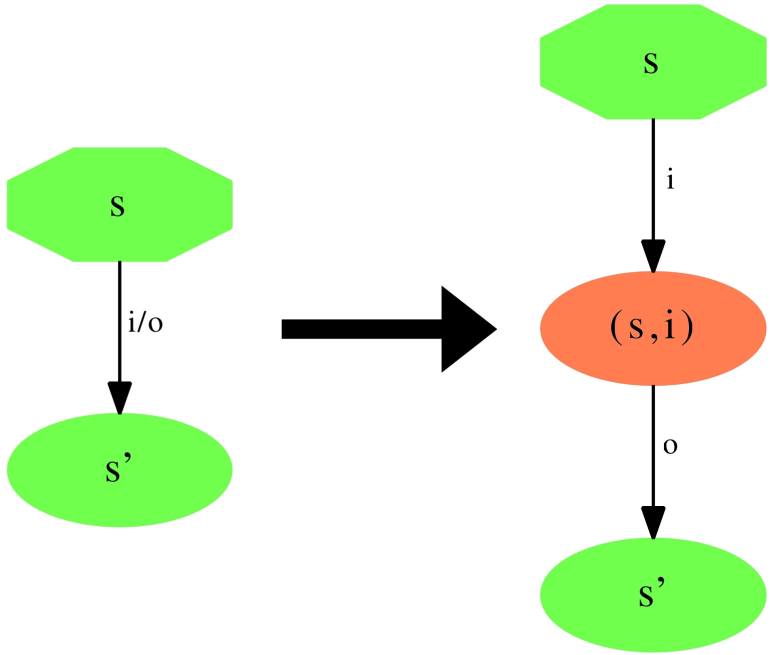


Figure: Transformation from FSM to automaton.

Berkeley Automata and Language Manipulation (BALM)

BALM

BALM was developed at U.C. Berkeley as a branch of the MVSIS project at the *Center for Electronic System Design*, Brayton's group, main architect Alan Mishchenko.

Features of BALM

The BALM package aims at providing an experimental environment for efficient manipulation of finite automata in various application domains. With BALM it is possible to perform classic operations over automata and efficiently solve synchronous equations.

Berkeley Automata and Language Manipulation (BALM)

Main features

The package can perform classical operations like minimization, complementation, determinization, product, etc. on automata described in .aut format.

Graphical representation

BALM can show results converting the .aut files into .ps using graphviz libraries. The initial states are represented with octagons, while the other states are elliptical. Colours are used to define the acceptance/non acceptance of a state, respectively green and red.

BALM–II

BALM

BALM–II is an extension of BALM developed at the University of Verona.

Features of BALM–II

BALM–II is able to solve also parallel equations. In order to solve them, some operators like expansion and restriction were introduced, together with the procedure to derive an automaton starting from a FSM, and to derive an FSM from an automaton solution.

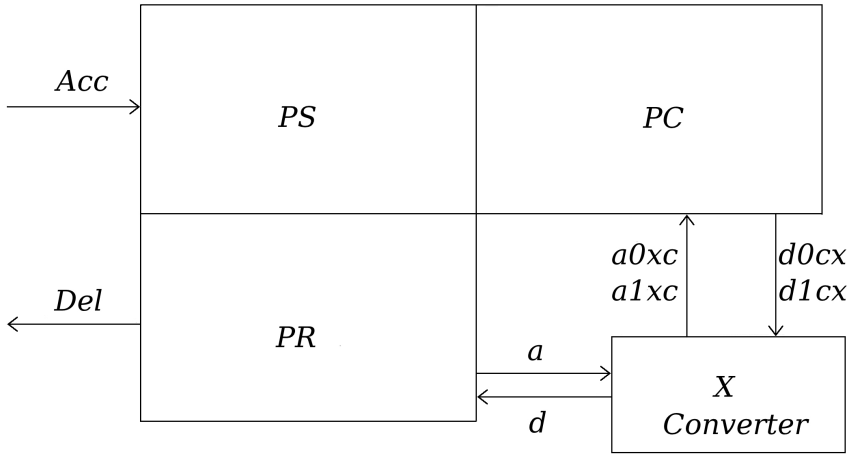
Some commands in BALM / BALM–II

```
complement
product

expansion
restriction

support
chan_sync

read_para_fsm
write_para_fsm
```

In BALM–II there is a topological notion of channel that is a connection between terminals; we may associate one or more discrete variables to a given channel. To operate on automata, they must be defined over the same variables with the same order: this is achieved by using `support` and `chan_sync`.

Solution of an equation over finite automata

As an example we consider the problem, taken from Kumar et al. (DEDS, 1997), of designing a protocol converter to interface an *alternating-bit* (AB) sender and a *non-sequenced* (NS) receiver.



The component of the system are an AB protocol sender (PS), an AB protocol channel (PC) and a NS protocol receiver (PR).

Automata representation of the components

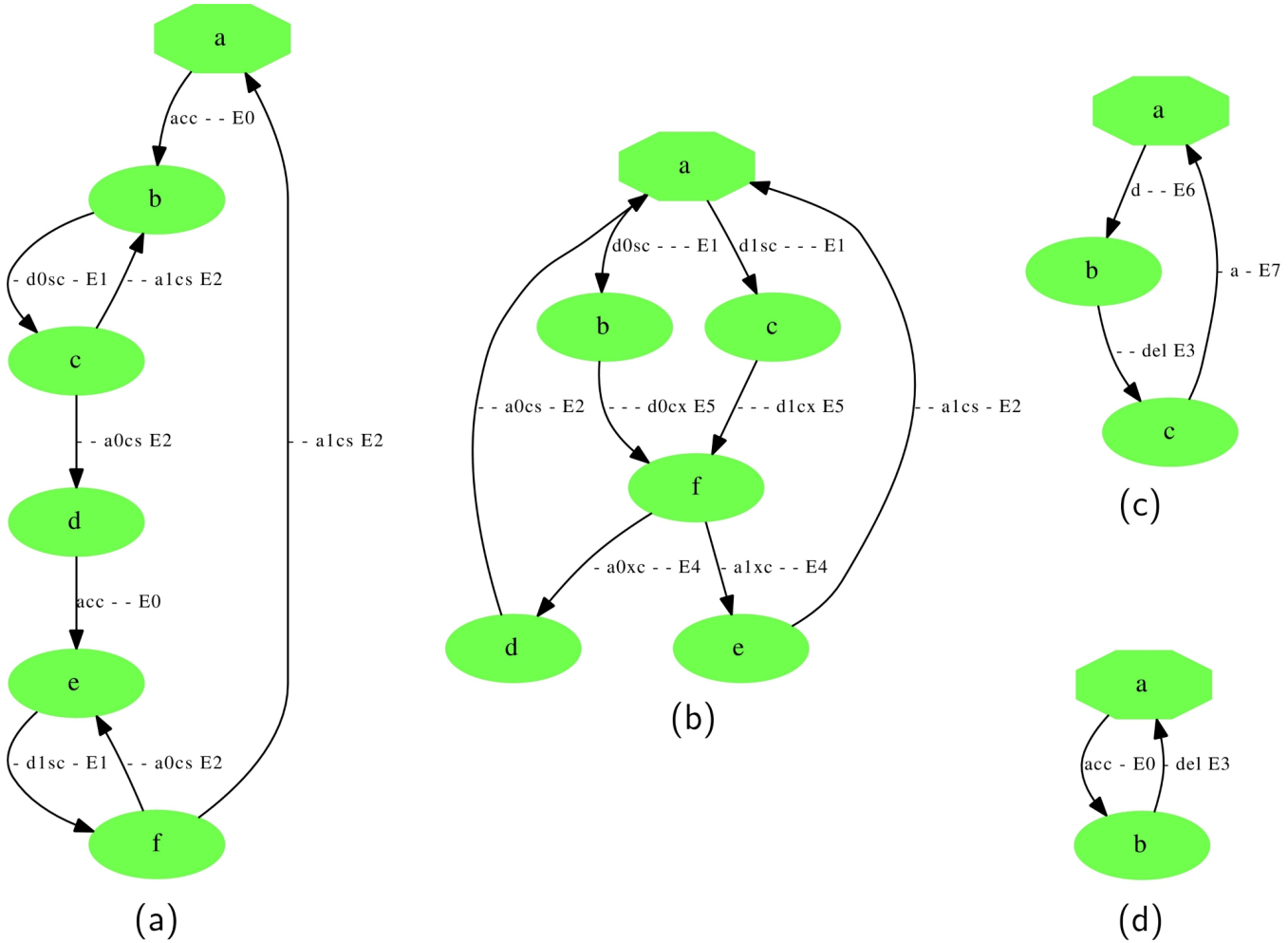
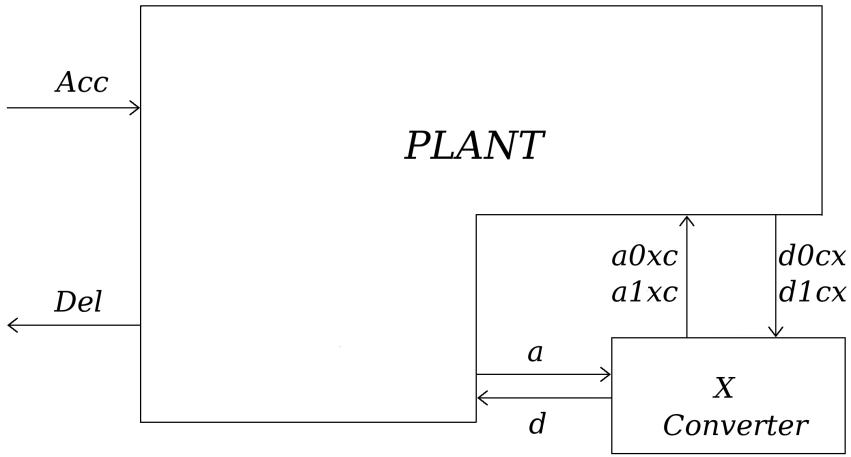


Figure: (a) AB sender; (b) AB channel; (c) NS receiver; (d) Specification.

Solution of the example

In order to use the procedure described above, it is necessary first to compute the composition of the components that represent the plant.



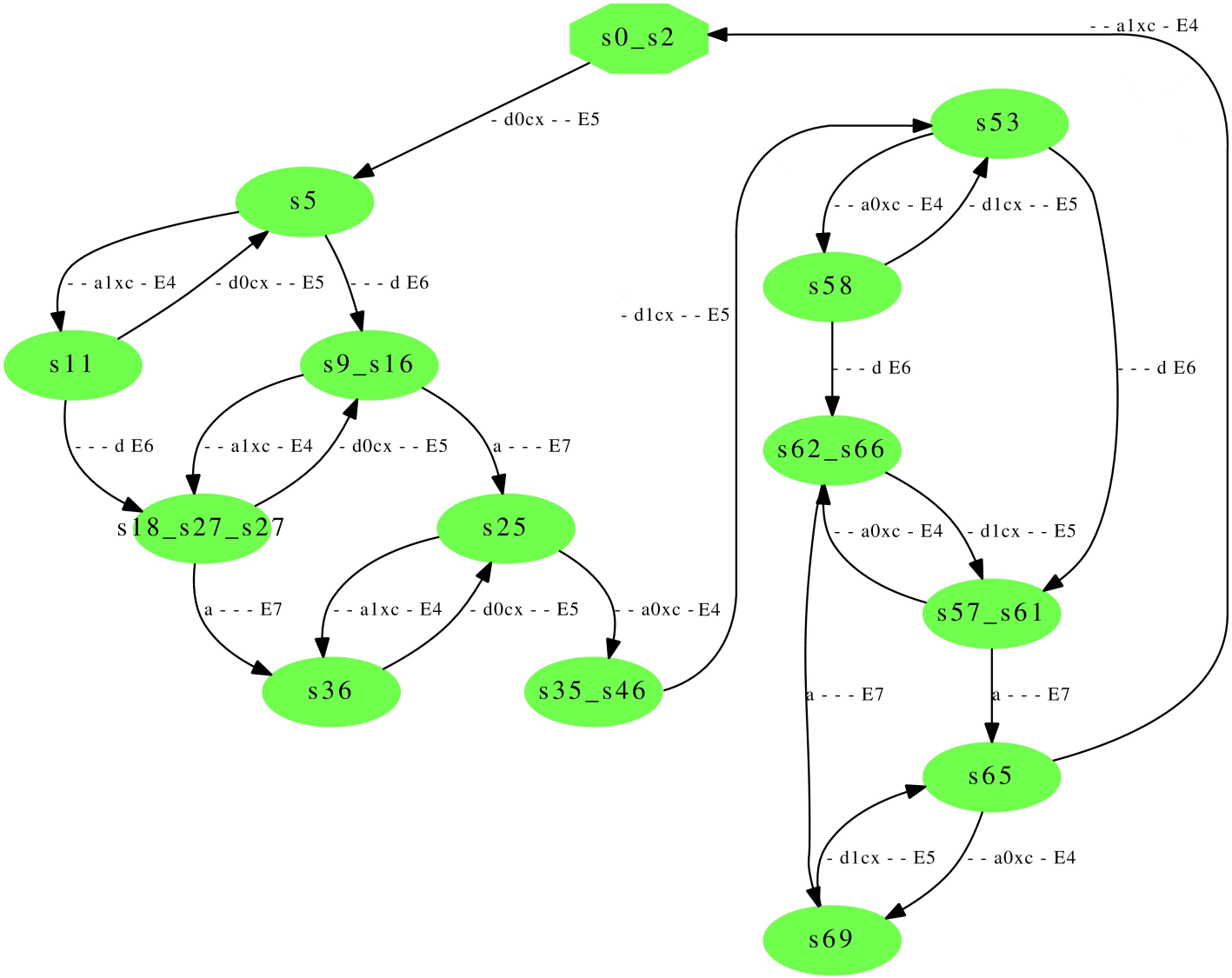
The solution is found by computing the formula:

$$X = \overline{PS \diamond PC \diamond PR \diamond \overline{S}}$$

Solving procedure in BALM–II

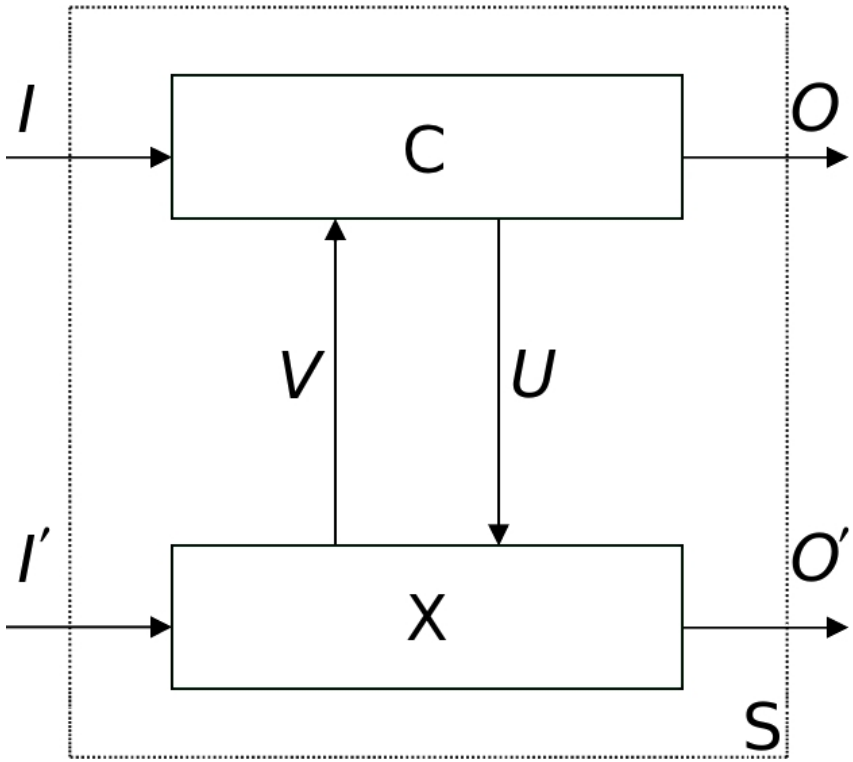
```
complement spec.aut spec_comp.aut
expansion E4,E5,E6,E7 spec_comp.aut spec_comp_exp.aut
product fixed.aut spec_comp_exp.aut product.aut
support I,O,AX,DX,D,A,E(8) product.aut product_supp.aut
restriction E4,E5,E6,E7 product_supp.aut product_res.aut
support A,DX,AX,D,E(8) product_res.aut product_supp.aut
complement product_supp.aut x.aut
```

Graphical representation of the largest automaton solution



Solution of an equation over FSMs

As a second example we consider a problem over FSMs taken from El-Fakih et al. (TCS, 2006). This is the topology:



Plant and specification as FSMs

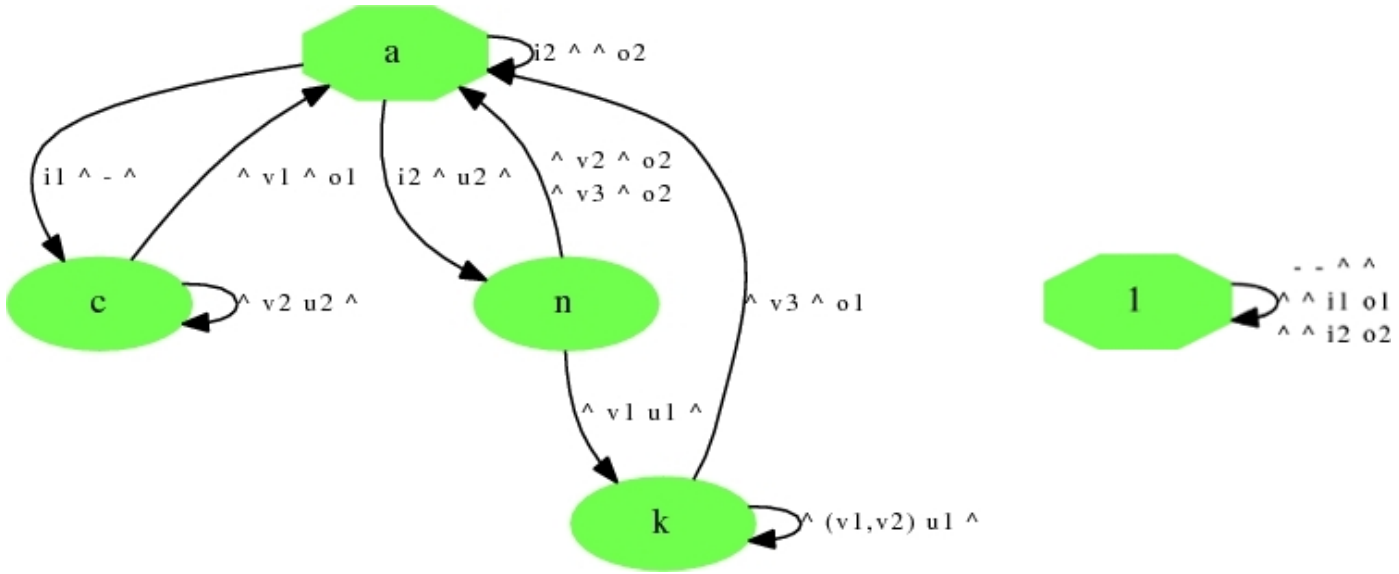
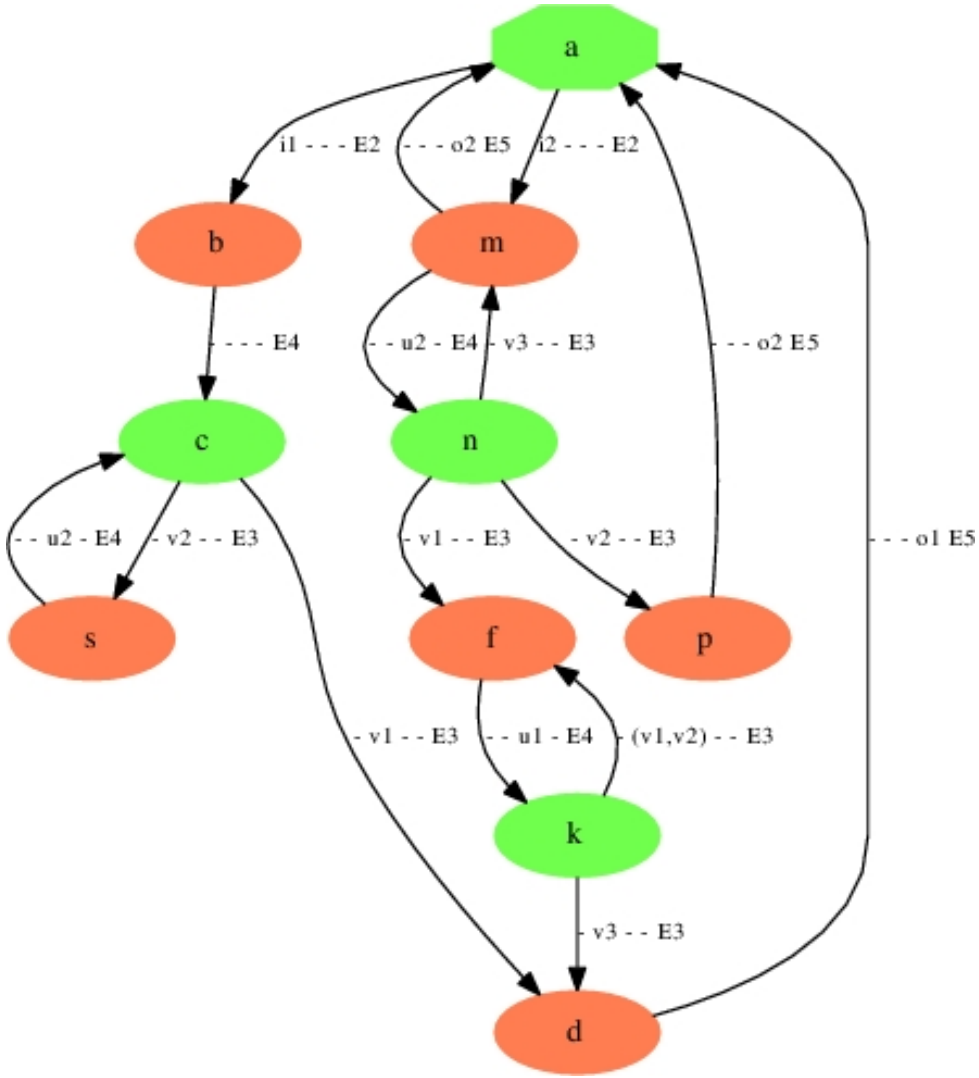
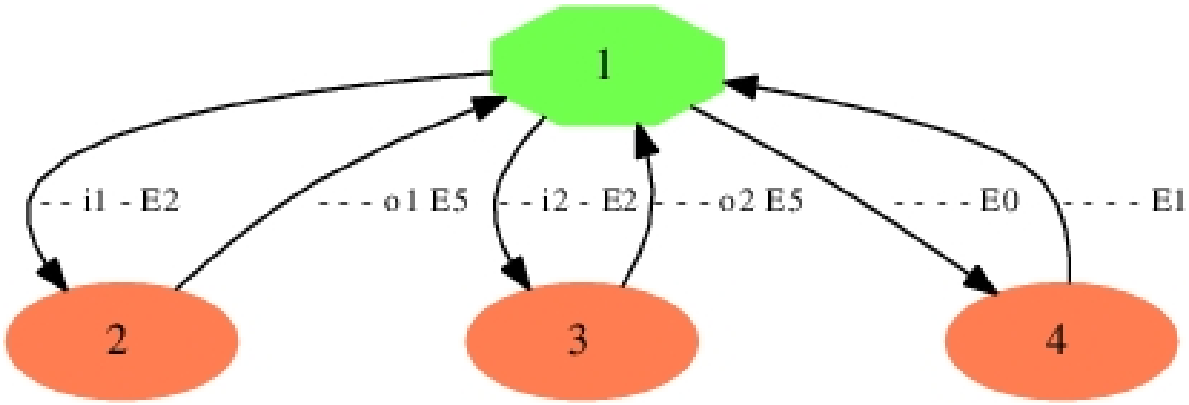


Figure: Plant FSM (left) and specification FSM (right)

Automaton representing the FSM language of the plant



Automaton representing the FSM language of the spec.



Notice that the specification has input variables $I = \{i1, i2\}$, $X = \{x\}$ and output variables $O = \{o1, o2\}$, $Y = \{y\}$.
As an FSM the specification has a single state and three self-loops under labels: $i1/o1$, $i2/o2$, x/y.

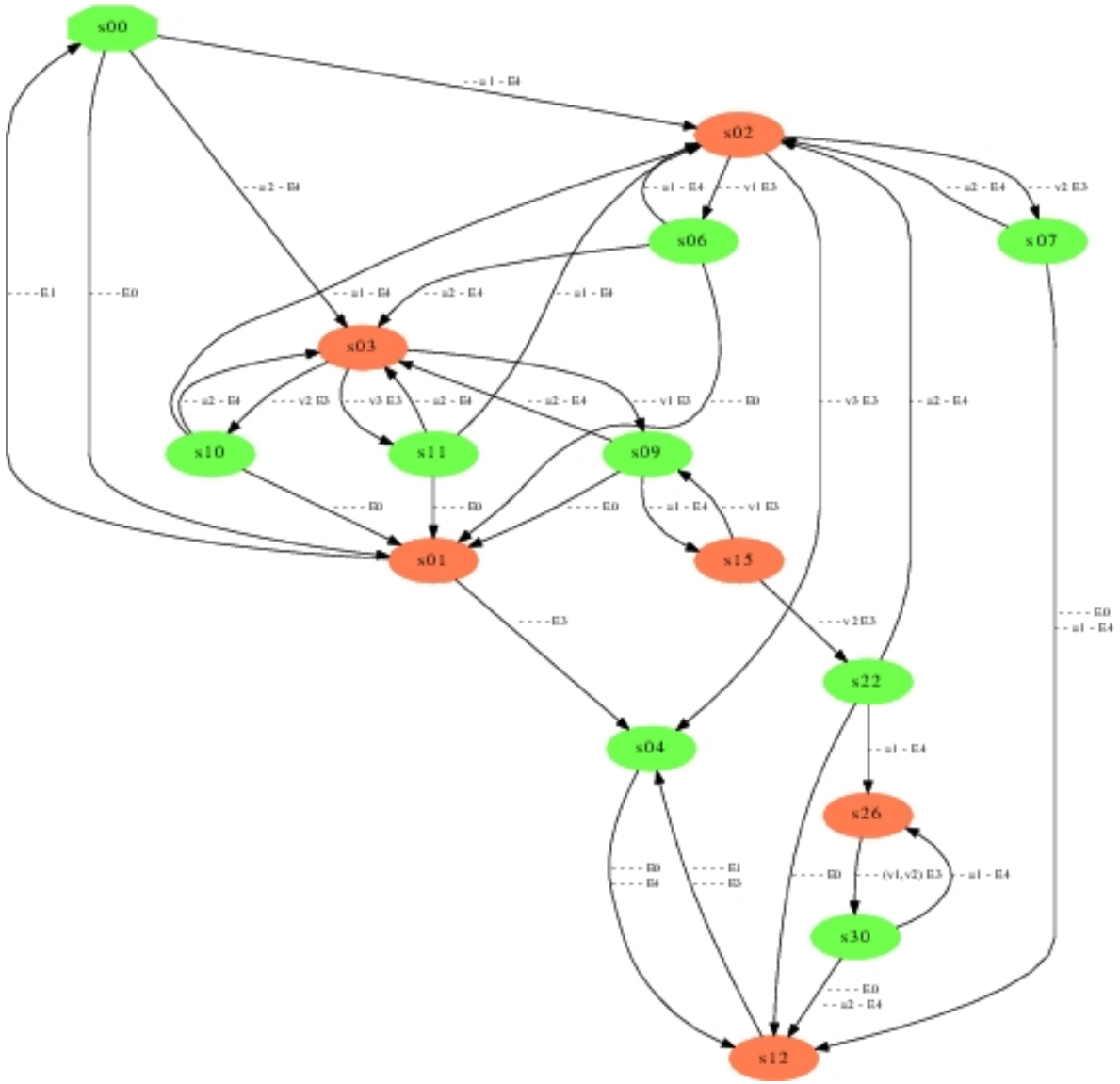
Solving procedure in BALM–II

The procedure to compute the largest FSM solution is:

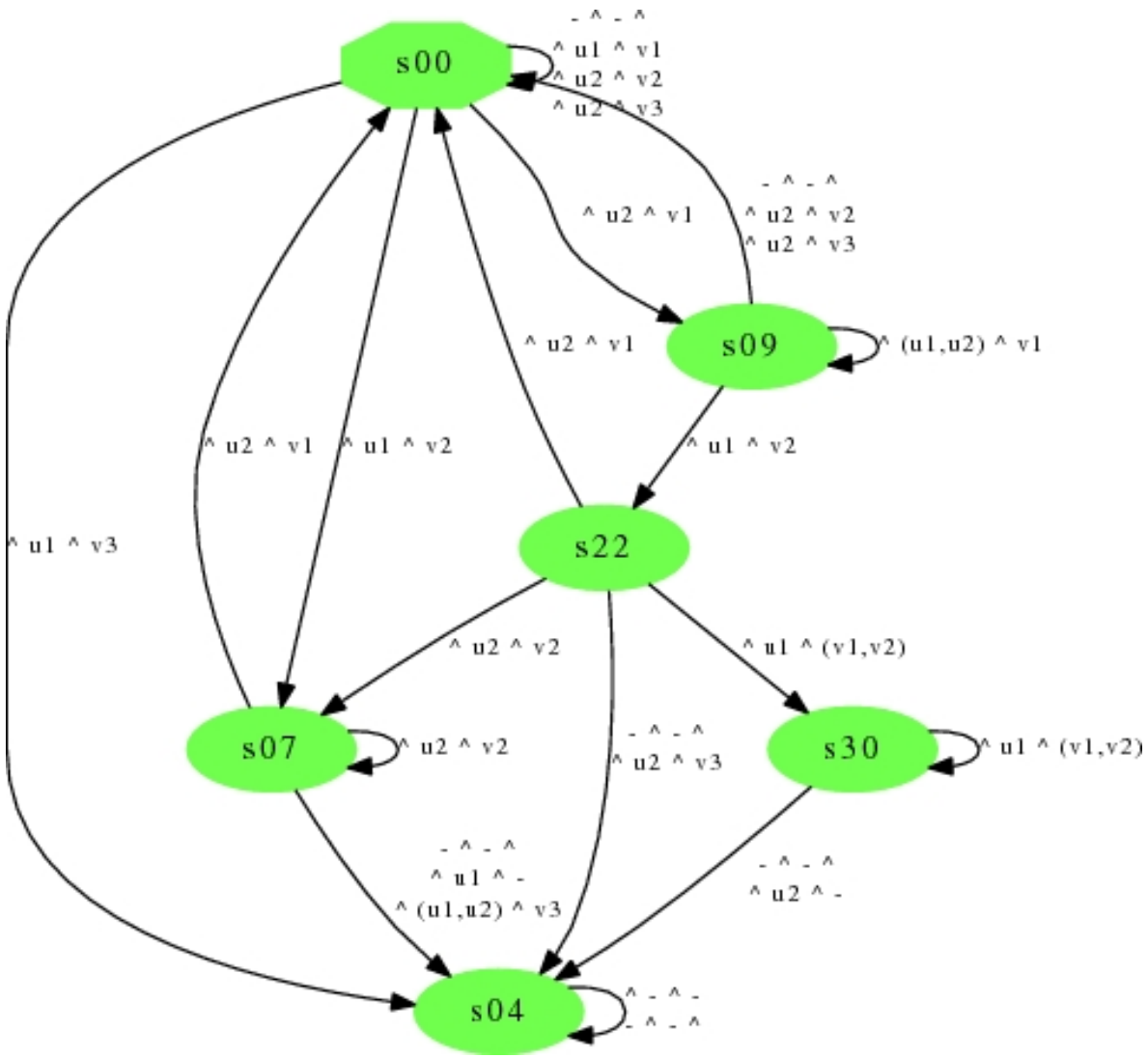
$$X = \overline{C \diamond \overline{S}}$$

```
complement spec_sync.aut spec_sync_comp.aut
product spec_sync_comp.aut ioStar.aut spec_sync_io.aut
expansion E3,E4 spec_sync_io.aut spec_sync_io_exp.aut
expansion E0,E1 context_sync.aut context_sync_exp.aut
product context_sync_exp.aut spec_sync_io_exp.aut product.aut
restriction E0,E1,E3,E4 product.aut product_res.aut
complement product_res.aut product_comp.aut
product product_comp.aut uvStar.aut x.aut
write_para_fsm x|u|y|v|E E0|E1,E4|E3 x.aut.aut x_fsm.aut
```

Automaton representing the largest FSM solution



Largest FSM solution



Conclusions and future work

Results

- Developed a general frame for unifying the synthesis of components given a specification and a context.
- BALM–II was tested successfully on many examples, some taken from problems of synthesis of protocol converters.

Conclusions and future work

Results

- Developed a general frame for unifying the synthesis of components given a specification and a context.
- BALM–II was tested successfully on many examples, some taken from problems of synthesis of protocol converters.

Future work

- Investigate the problem of extracting an "optimal solution" for synthesis/resynthesis of sequential logic.
- Extend the software to solve supervisory control problems with full or partial controllability and full or partial observability.

References

Tiziano Villa, Nina Yevtushenko, Robert Brayton, Alan Mishchenko, Alexandre Petrenko, Alberto Sangiovanni-Vincentelli
The Unknown Component Problem: Theory and Applications
Springer Verlag, 2012

Tiziano Villa, Alexandre Petrenko, Nina Yevtushenko, Alan Mishchenko, Robert Brayton
Component-Based Design by Solving Language Equations
Proc. of IEEE, Nov. 2015

Giovanni Castagnetti, Matteo Piccolo, Tiziano Villa
BALM-II (Software and user's manual)
http://esd.scienze.univr.it/index.php/it/balm-ii.html

Giovanni Castagnetti, Matteo Piccolo, Tiziano Villa, Nina Yevtushenko, Alan Mishchenko, Robert Brayton
BALM–II (Solving Parallel Equations with BALM-II)
EECS Dept., UC Berkeley, Tech. Rep. UCB/EECS-2012-181, July 2012

Ratnesh Kumar, et al.
A Discrete Event Systems Approach for Protocol Conversion
Discrete Event Dynamic Systems, 1997

Khaled El-Fakih, Nina Yevtushenko, et al.
Progressive Solutions to a Parallel Automata Equation
Theoretical Computer Science, 2006

Roberto Passerone
Interface Specification and Converter Synthesis
Embedded Systems Handbook, Aug 2005, 23:1-20