Synthesis of Hierarchical Systems from Libraries

Benjamin Aminof
Hebrew Univerisity

Fabio Mogavero
Università di Napoli "Federico II"

Aniello Murano
Università di Napoli "Federico II"

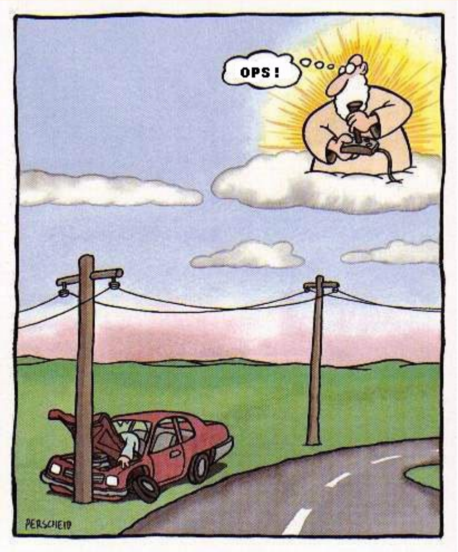
Paris – France
September 3, 2011

Preface

- Model Checking: Given a finite system and its desired behavior
  - System → labeled state-transition graph M
  - Specification → a temporal logic formula ψ
  - The system has the desired behavior → $M \models \psi$
- Synthesis: Given ψ, M is built in such a way $M \models \psi$
  - The correctness of the system is given by construction !

Open systems

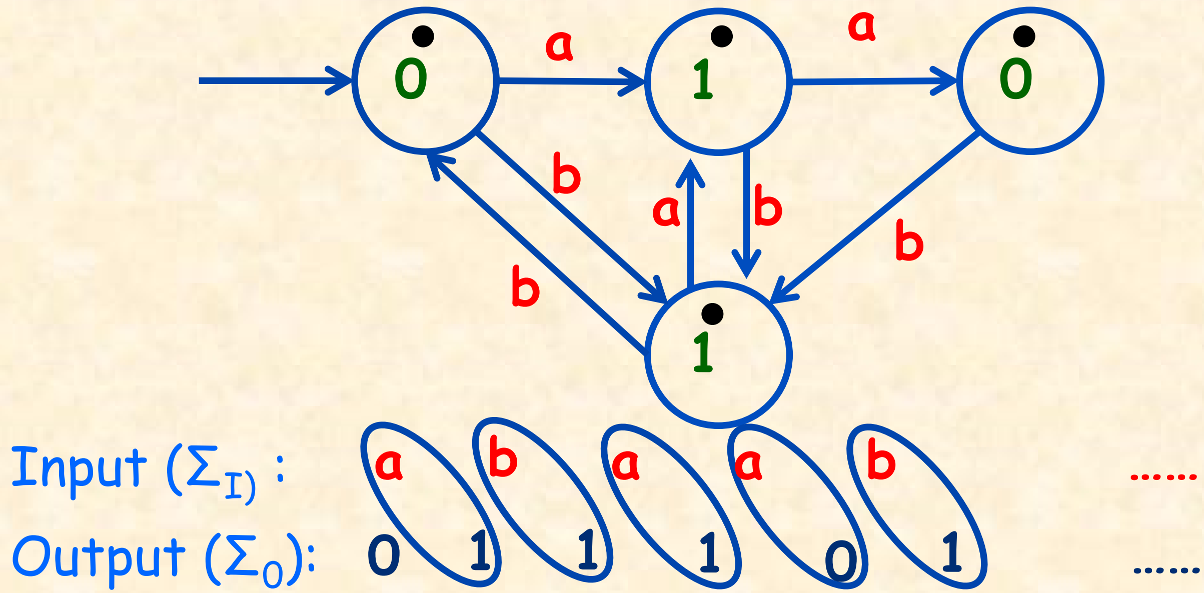
- In system design, we distinguish between
- Closed Systems → the system behavior depends on its own
- Open System → the system interacts with the environment and its behavior strongly depends on this interaction.



- In the open case, we synthesize a system that satisfies the specification no matter how the environment behaves

Modeling open systems: Trasducers

- The interaction system-environment can be modeled by means of input and output signals



- A computation is a infinite word of input and output symbols
- All computations can be collected in a **Computation Tree** (a full Σ_O labeled Σ_I tree)

The Pnueli-Rosner Algorithm for LTL

Pnueli & Rosner'89

- An Input alphabet Σ_I
- An Output alphabet Σ_O
- A specification φ in LTL over $\Sigma_I \times \Sigma_O$
- Construct a Machine M such that that for every input string the induced input-output stream (computation) satisfies φ
- Technically, we build A_T that runs on computation trees. The system corresponds to the witness of the non-emptiness of A_T
- Complexity: **2Exptime-complete**
- For **CTL** and **μ-calculus** it is **Exptime-complete**

Synthesis in real-life systems

- The Pnueli-Rosner algorithm starts from scratch and produces a "flat" system.
- Real-life software and hardware systems are created
  1. using preexisting libraries
  2. not "flat" since repeated sub-systems are described only once.
- We propose an algorithm for the synthesis of a hierarchical system from a library of hierarchical components
- Extends "Synthesis from Component Libraries" ([Lustig-Vardi'09])
- The algorithm we propose works for several specification logics.
- In terms of complexities, it never works worst than the classical approach.

Outline of the talk

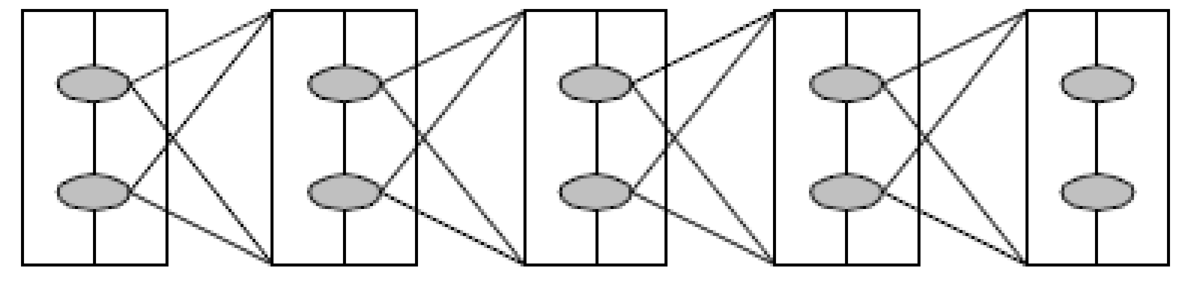
- ✓ Introduction
- ❑ Hierarchical transducers
- ❑ Connectivity trees
- ❑ Solving the synthesis problem via tree automata emptiness
- ❑ Modularity
- ❑ Imperfect information

Hierarchical Transducers

A Hierarchical transducer **M** $= \langle \Sigma_I , \Sigma_O, (M_1,\ldots, M_k)\rangle$ is such that

- Σ_I and Σ_O are input and output alphabets, respectively
- $(M_1,\ldots, M_k)$ are k deterministic sub-transducers modelling k sub-procedures
- Each sub-transducer can call others hierarchically
- Sub-transducers are represented through labelled graphs

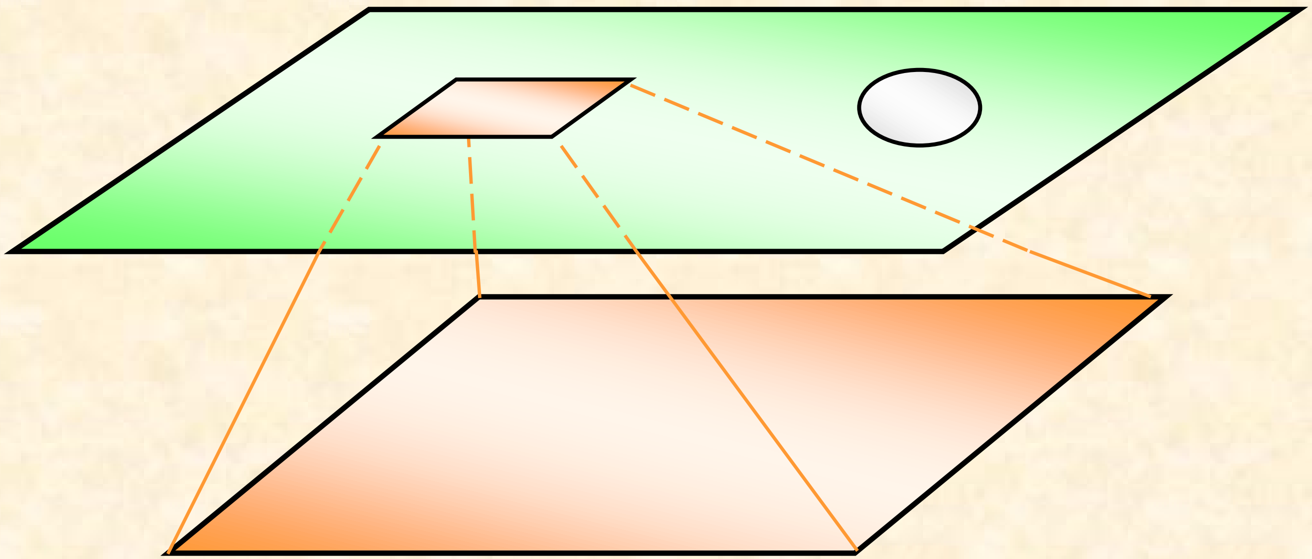
- **Intuition:** A pushdown system with a **bounded stack**.



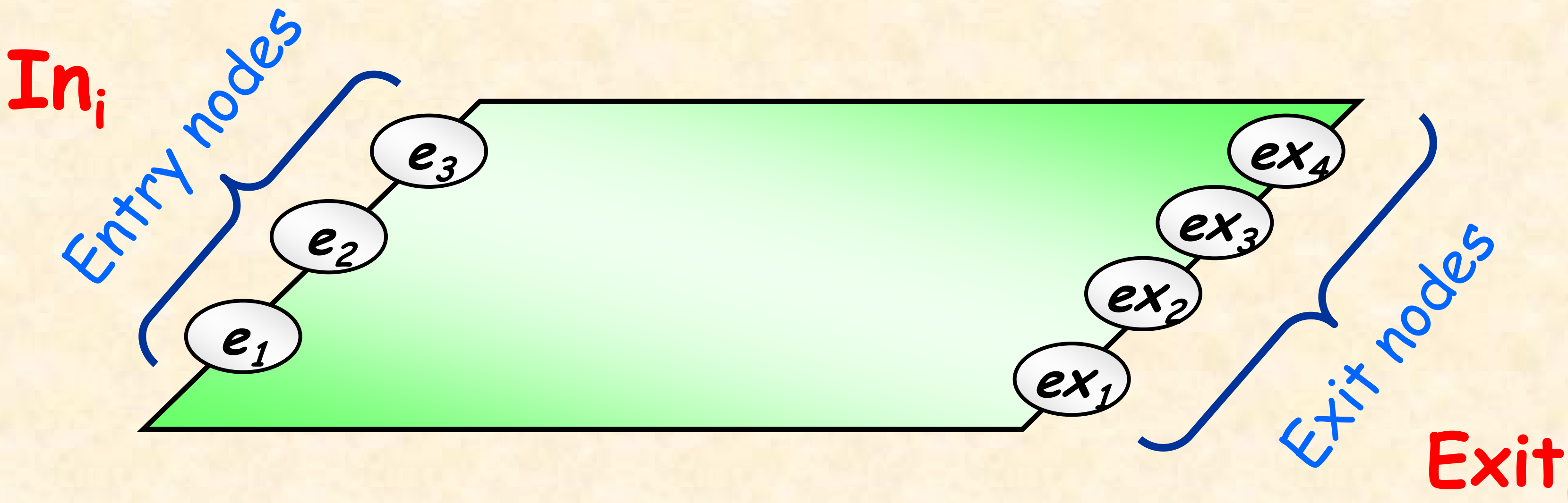
Vertices of M_i

Each Sub-transducer M_i can have two kind of vertices:

- Nodes (internal state): Q_i
- Boxes (procedure-call): B_i
  - Note that different boxes may call the same procedure, all under the hierarchical order.



Entry and Exit Nodes of M_i

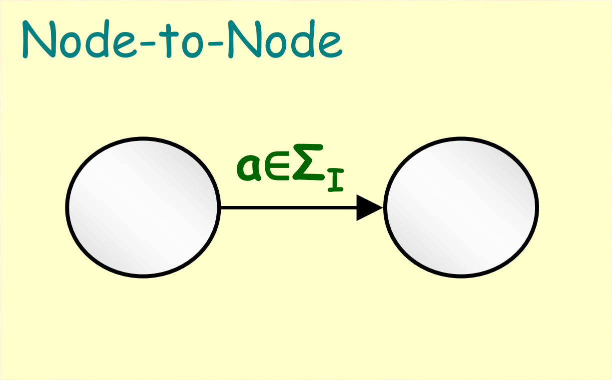


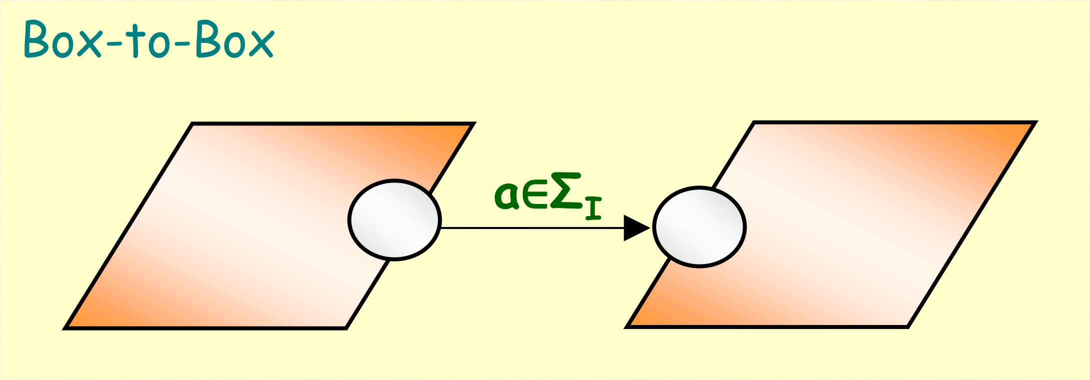
parameters

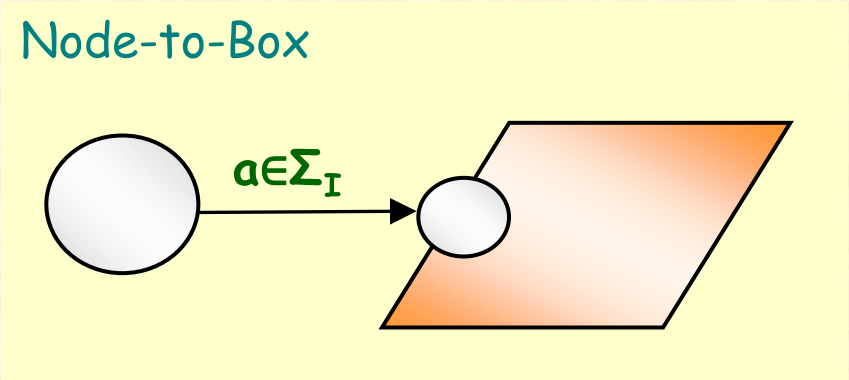
return values

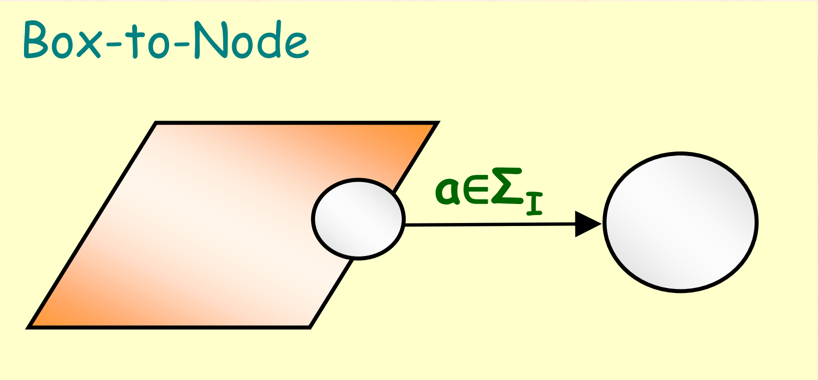
Edges in M_i

δ_i









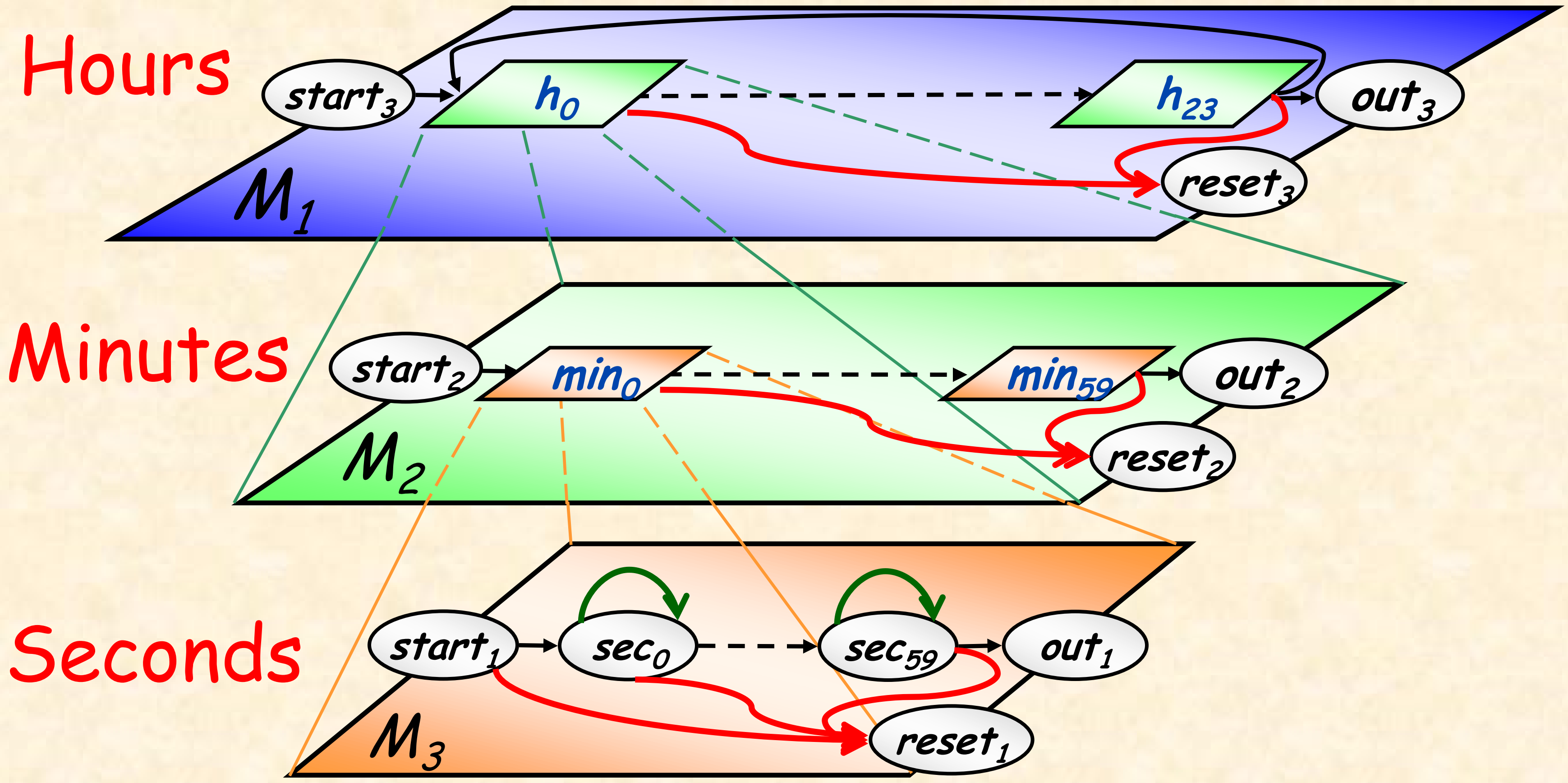
Labelling nodes in M_i

$lab_i : Q_i \rightarrow \Sigma_0$

- We associates to nodes output simbols

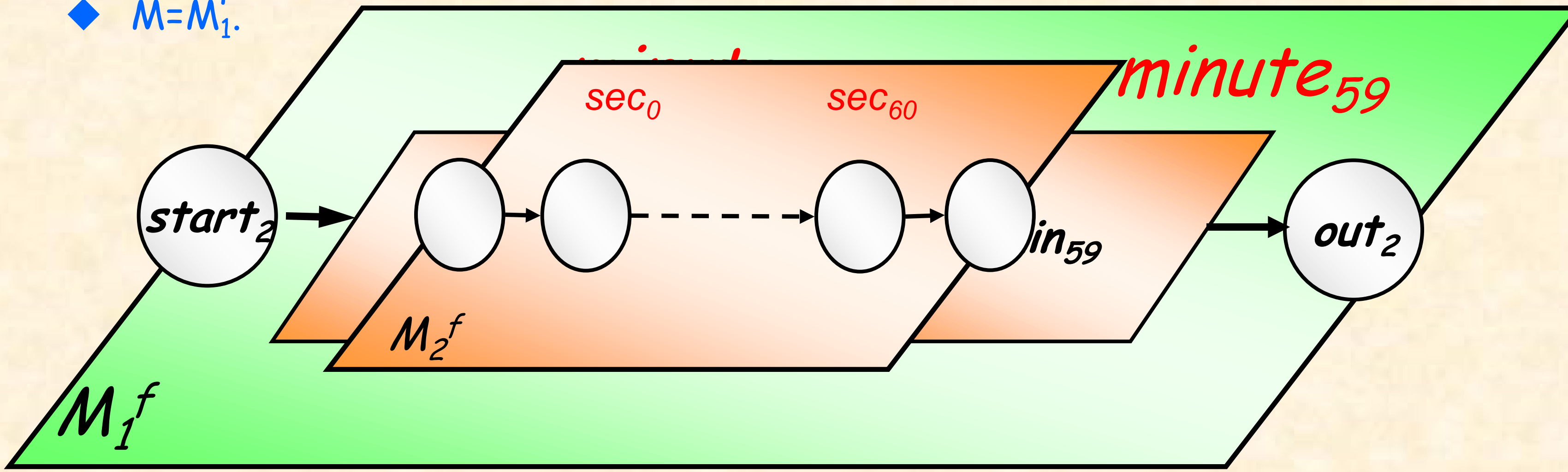
Example: Digital Cronometer as an Hierarchical Transducer

Σ_I = {pause, reset, next}



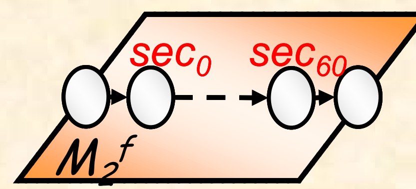
Flat Model

- A hierarchical Transducer M can be "flattened" in an ordinary Transducer M^f: recursively substitute each box with the sub-transducer it refers to
  - $M^f_n = M_n$ (there are no boxes in M_n)
  - In M^f_i each state is $q=(b_1,\dots,b_m, u)$. **The labeling $Lab^f(q) = Lab(u)$.**
  - $M^f = M^f_1$.



The flat model in the example has 24·60·60=86400 states!!!

In the worst case, the fattening can cause an exponetial blow-up: since different boxes can be associated with the same structure, a state can appear in different contexts.



Hierarchical satisfiability

- Let φ be a formula built using as atomic propositions $\Sigma_I \times \Sigma_O$
- Let $M = \langle \Sigma_I, \Sigma_O, \langle M_1, \ldots, M_n \rangle \rangle$ be a hierarchical transducer
- Let T_M the computation tree of M^f
- $M \models \varphi$ iff the computation tree T_M satisfies φ.

Hierarchical Synthesis from a hierarchical Library

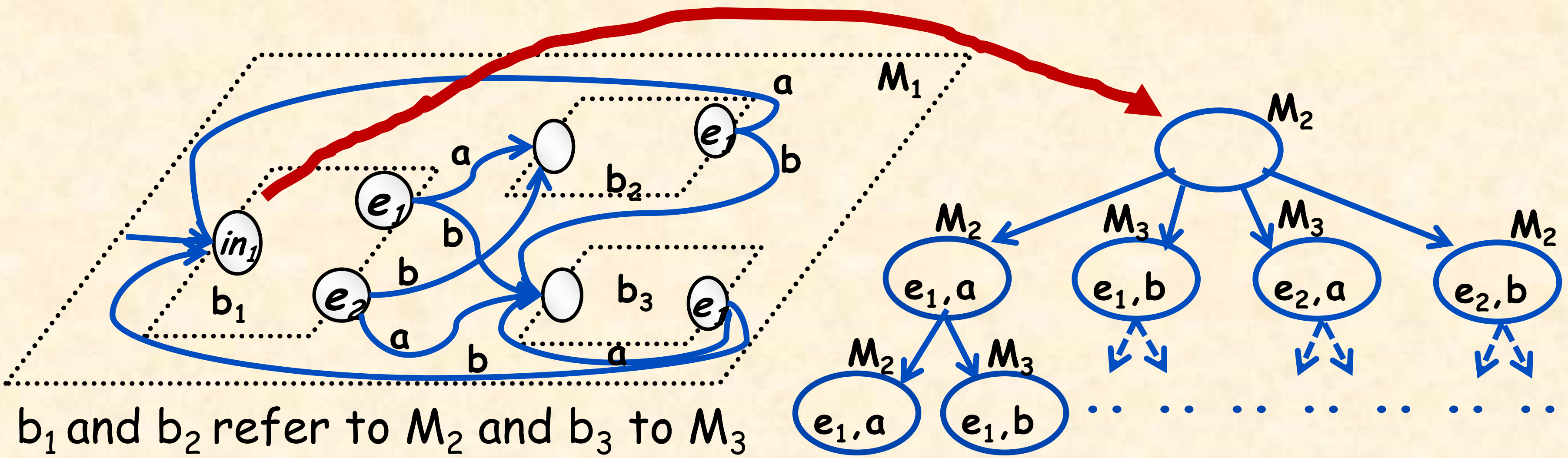
- The algorithm mimics the **bottom-up** approach in real-life programming
- It starts with an initial library L_0 of hierarchical (atomic) transducers.
- Then, it proceeds synthesizing the system in rounds.
- At each round $i > 0$:
  - a specification φ_i of M_i is provided
  - M_i is synthesized using transducers in L_{i-1} as sub-transducers.
  - $L_i = L_{i-1} \cup M_i$ (some existent components can be also removed).
- The hierarchical transducer synthesized in the last round is the output of the overall algorithm.
- The single-round synthesis is the main challenge of the algorithm !!!

Recall the Pnueli-Rosner Idea

- In the Pnueli-Rosener algorithm, we build an automaton that runs on computation trees. The system is the witness of the non-emptiness.
- In the hierarchical synthesis, can we use a better (smaller) tree structure ?
- Remember that the **atomic objects** we use in the hierarchical setting are **hierarchical structures** rather than atomic propositions.
- We use **connectivity trees**!
- *A* connectivity tree describes how to connect sub-transducers from a library in order to create a new hierarchical transducer.

Connectivity tree C_T

- C_T is an L-labeled {exits × Σ_I)-tree and represents the unwinding of the top structure M_1 of the transducer $M = \langle \Sigma_I, \Sigma_O, \langle M_1, \ldots, M_n \rangle \rangle$



b_1 and b_2 refer to M_2 and b_3 to M_3

- Each node in C_T corresponds to a sub-transducer a box in M refers to.
  (Regular states can be treated as atomic transducers)
- The label of a son $x \cdot (e,\sigma)$ specifies the destination of the transition from the exit e of b associated to x when reading σ.

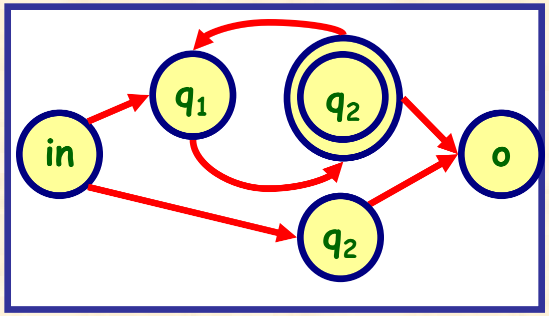
Solving the one-step hierarchical synthesis

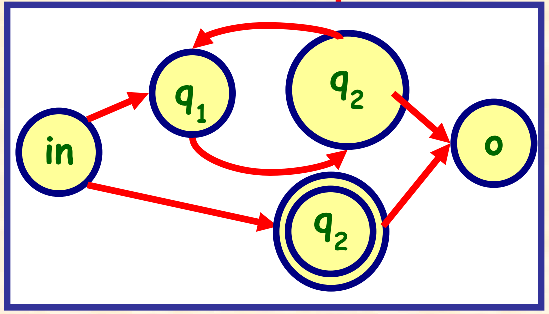
- Given L and φ, we build an APT A_T that runs on connectivity trees.
- A_T accepts C_T iff C_T represents a transducer M that satisfies φ.
- The basic idea: on reading C_T, on each node, A_T simulate the computation tree automaton on the corresponding sub-transducer.
  - This is done without consuming input (in the connectivity tree) until we reach an exit.
  - Then, we consult the children of the current node in C_T that specifies to which sub-transducer the simulation should proceed.
  - **Caution(!!):** In A_T we cannot embed the computation tree automaton otherwise we embed all flat sub-transducers coming from L.

A_T and the Overall Complexity

- We embed in A_T only a **summary** of the computation tree automaton w.r.t. each structure.
- For example, for a Buchi Automaton, it is enough to report if it accepts locally or meets an accepting state on the way to an exit.
- The summary function can be obtained by solving **local** games.
- Starting with a μ-calculus (LTL) formula φ, A_T can be built in time polynomial in |L|, exponential in |Exits|, and exp. (resp., 2exp.) in |φ|
- By applying classical results for the emptiness of an A_T we get that

The hierarchical synthesis problem from a library is EXPTIME-complete for μ-calculus and 2EXPTIME-complete for LTL.





Modularity

- The automata-theoretic approach easily allows to enforce some modularity
- In particular, we can inject regular modularity properties
  - Limitiation on the number of pure states before having a call to a sub-procedure
  - Limitation on maximal number of times a routine is called

Imperfect Information

- The extra work required to A_T is to only consider trees that are consistent with the obervability
- This can be easily done by applying classical construction from synthesis with imperfect information

Conclusion

- Synthesis of an hierarchical system form a library of hierarchical sub-structures
- We extend the Pnueli-Rosner idea by working directly on hierarchical components, without flattening them.
- The overall complexity matches the complexity of classical synthesis.

References

- Benjamin Aminof, Fabio Mogavero, Aniello Murano: Synthesis of Hierarchical Systems. FACS 2011: 42-60, 2011.
- Rajeev Alur, Mihalis Yannakakis: Model checking of hierarchical state machines. ACM Trans. Program. Lang. Syst. 23(3): 273-303, 2001
- Salvatore La Torre, Margherita Napoli, Mimmo Parente, Gennaro Parlato: Verification of scope-dependent hierarchical state machines. Information and Computation 206(9-10): 1161-1177, 2008
- Benjamin Aminof, Orna Kupferman, Aniello Murano: Improved Model Checking of Hierarchical Systems. VMCAI 2010: 61-77, 2010
- Aniello Murano, Margherita Napoli, Mimmo Parente: Program Complexity in Hierarchical Module Checking. LPAR 2008: 318-332, 2008
- Benjamin Aminof, Orna Kupferman, Aniello Murano: Improved model checking of hierarchical systems. Information and Computation, 2012. To appear
- Laura Bozzelli, Aniello Murano, Adriano Peron: Pushdown module checking. Formal Methods in System Design 36(1): 65-95, 2010
- Benjamin Aminof, Aniello Murano, Moshe Y. Vardi: Pushdown Module Checking with Imperfect Information. CONCUR 2007: 460-475, 2007
- Benjamin Aminof, Axel Legay, Aniello Murano, Olivier Serre: μ-calculus Pushdown Module Checking with Imperfect State Information. IFIP TCS 2008: 333-348