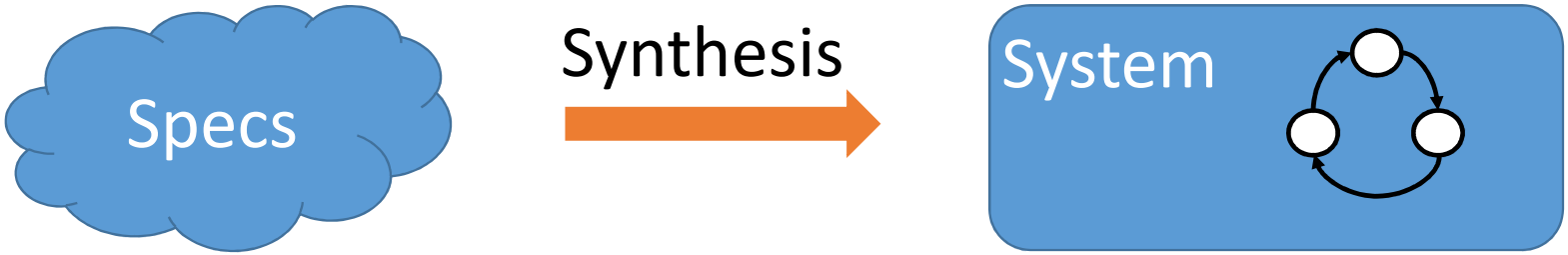
Modular Synthesis of Pushdown Systems

Salvatore La Torre, DI, Univ. Salerno

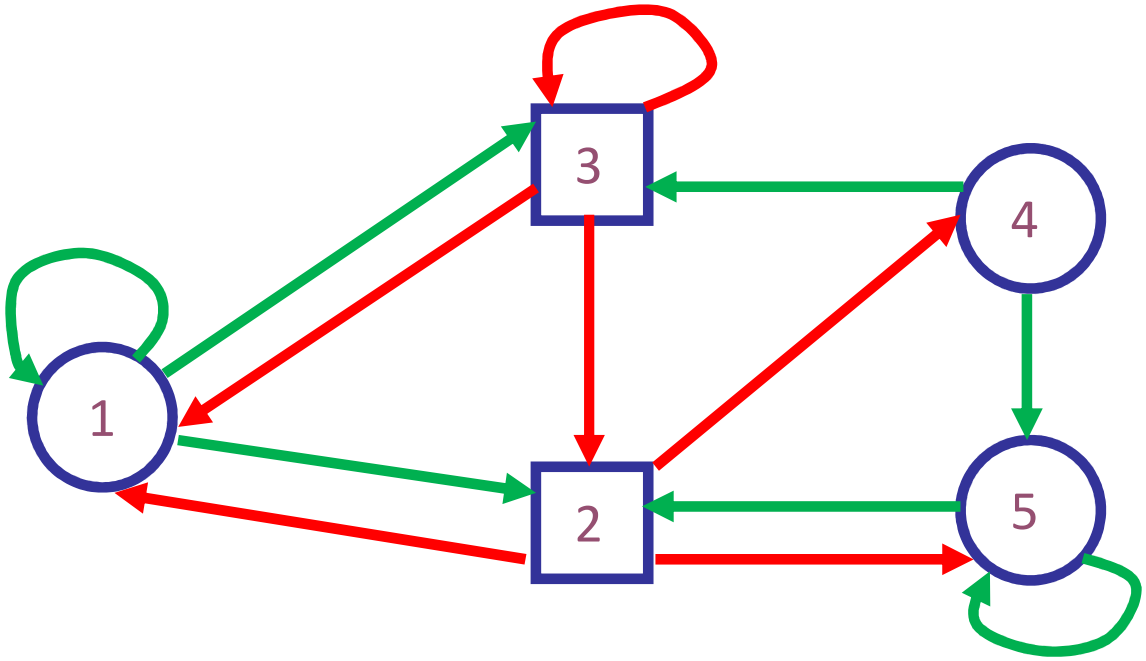
Synthesis Problem



- We wish to synthesize a system that is correct w.r.t. a specification
- Different formulations in the literature
- Several classes of systems and specifications

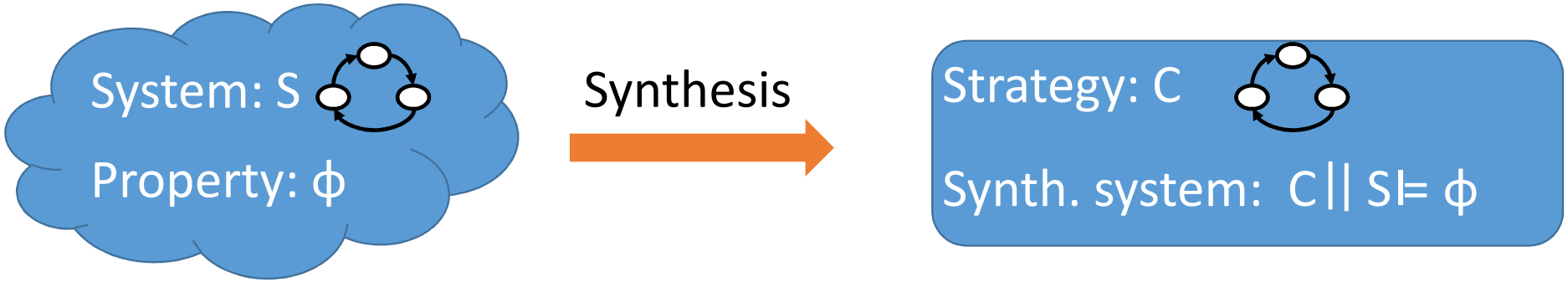
Game-graph formulation

- Open System (Arena): graph with controllable and uncontrollable actions

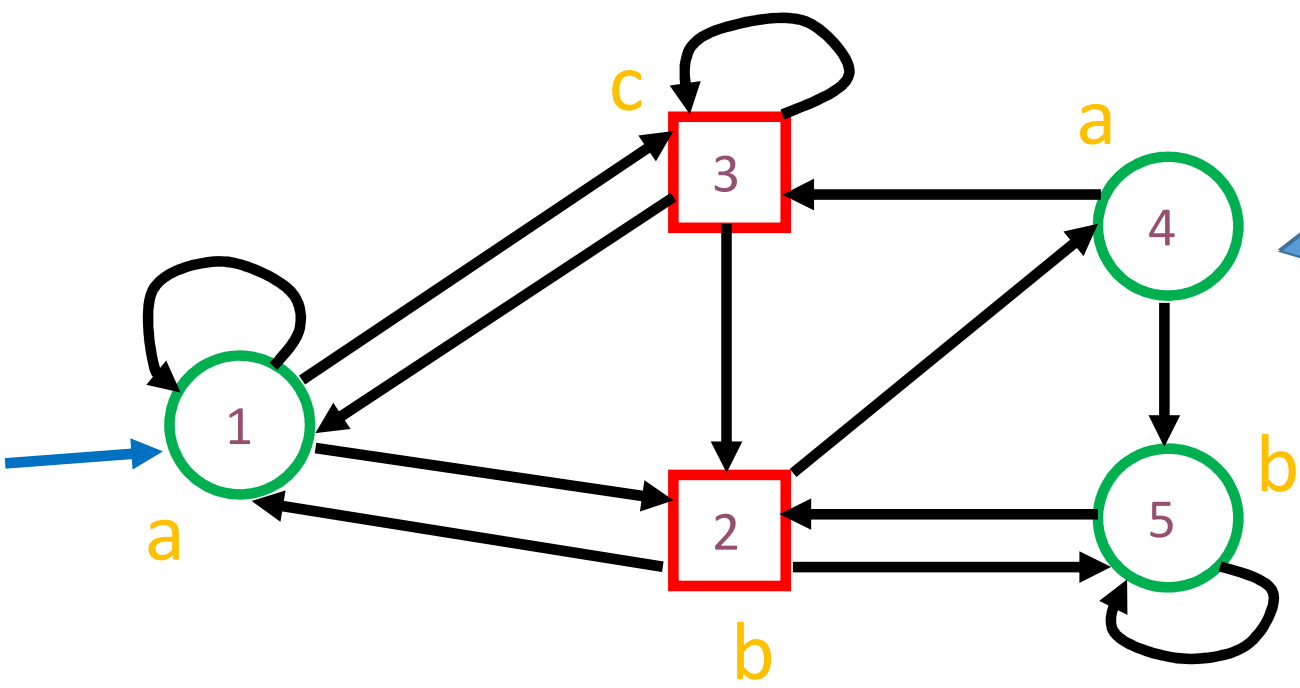


Game graph formulation

- Open System (Arena): graph with controllable and uncontrollable actions
- We wish to synthesize a Strategy that
  - selects controllable actions
  - enforces specs on the System



Game graph formulation

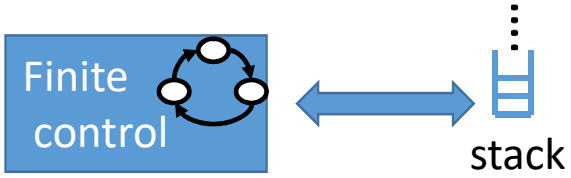


(correct) Synthesized System

- Property: every a is eventually followed by a b

Pushdown systems (PDS)

- Accurate model of control flow of programs with recursive function calls
- Classical formulation (machine view):



- Semantics is given by an infinite graph (which may not be the unwinding of any finite graph):
  - Nodes are the configurations (control state + stack content)
- Well studied in the literature (formal languages, model-checking, static analysis, games/synthesis, parsing, ….)

Pushdown games

- The arena is the underlying infinite graph
  - Controllable/uncontrollable actions are according to splitting of control states
- Relevant for synthesis of programs with recursive function calls
- Deciding pushdown games (i.e., is there a winning strategy?) is
  - undecidable for context-free specs (trivial)
  - EXPTIME-COMPLETE for parity winning conditions [Walukiewicz, Info&Comp, 2001]
  - 3EXPTIME-COMPLETE for LTL/CARET specifications [Loeding-Madhusudan-Serre, FSTTCS'04]
- Focus of the talk: modular pushdown games

Some motivation…

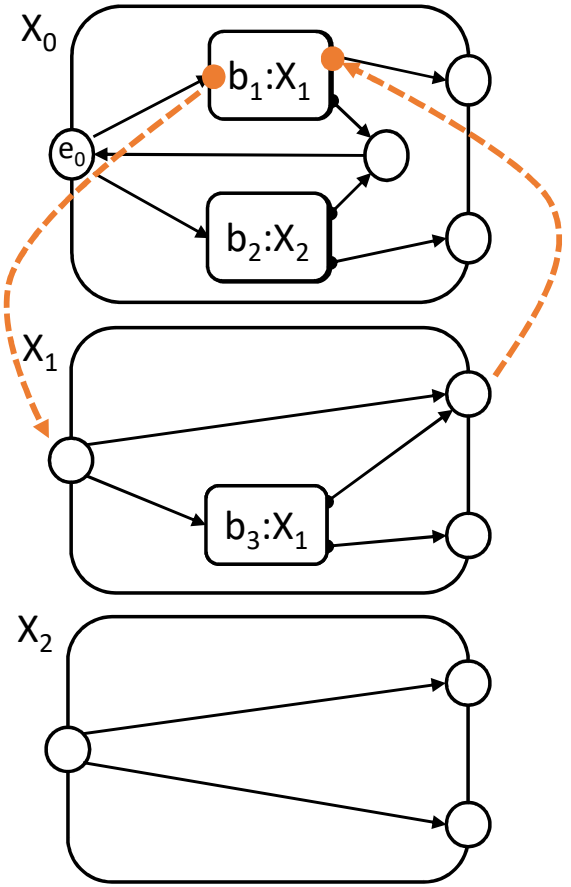
- Real (complex) systems are structured in modules
  - simplification and standardization of designs
  - simpler updates
  - simpler debugging
  - efficient and scalable development
  - possible reuse of preexisting modules
  - …
- Programs are composed of functions
- Modular synthesis:
  - Synthesis of systems composed of modules

Modular view of PDS

- Recursive state machine (RSM) [Alur et al., TOPLAS 05]
  - Set of finite state machines composed via the call-return paradigm
  - Each module has entry and exit points
  - Standard nodes
  - Boxes are mapped to modules:

    calls → entries

    returns → exits



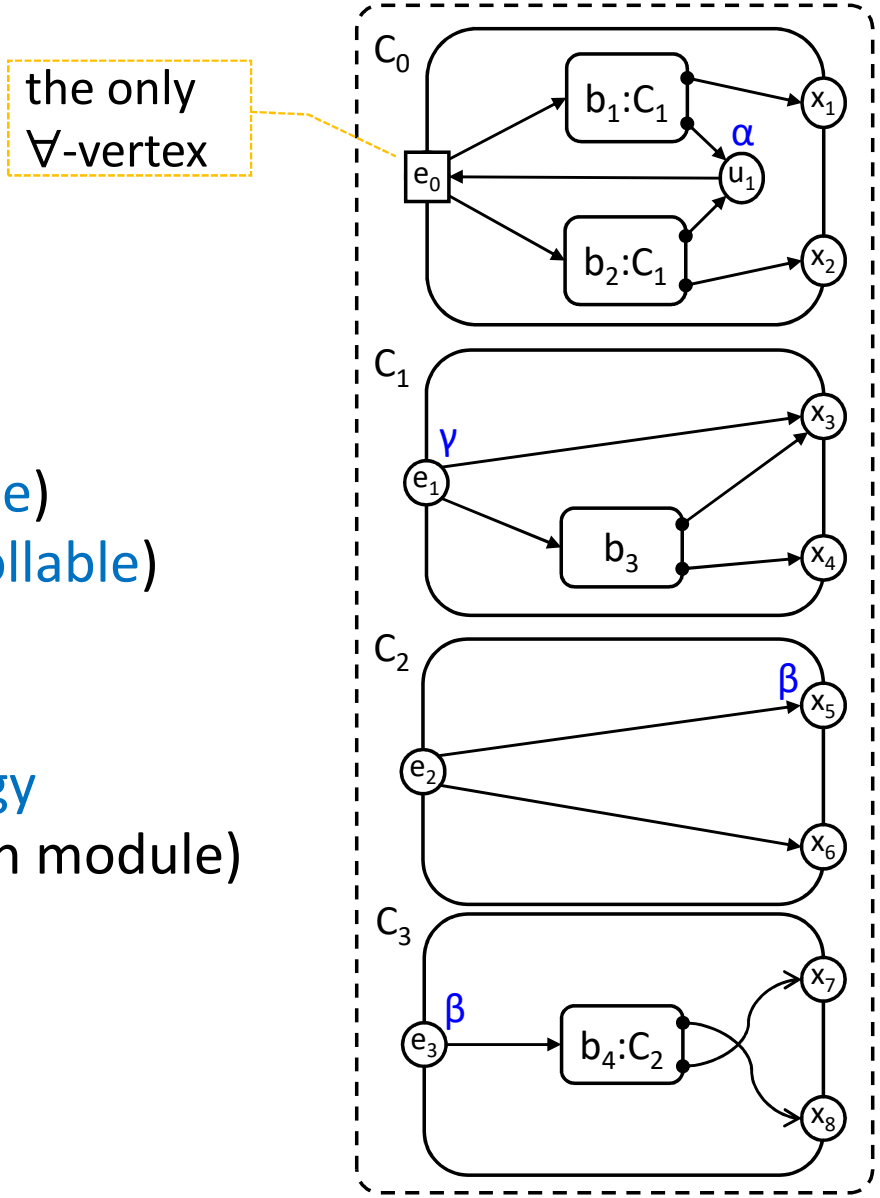
Modular PD Arena: Open component Library

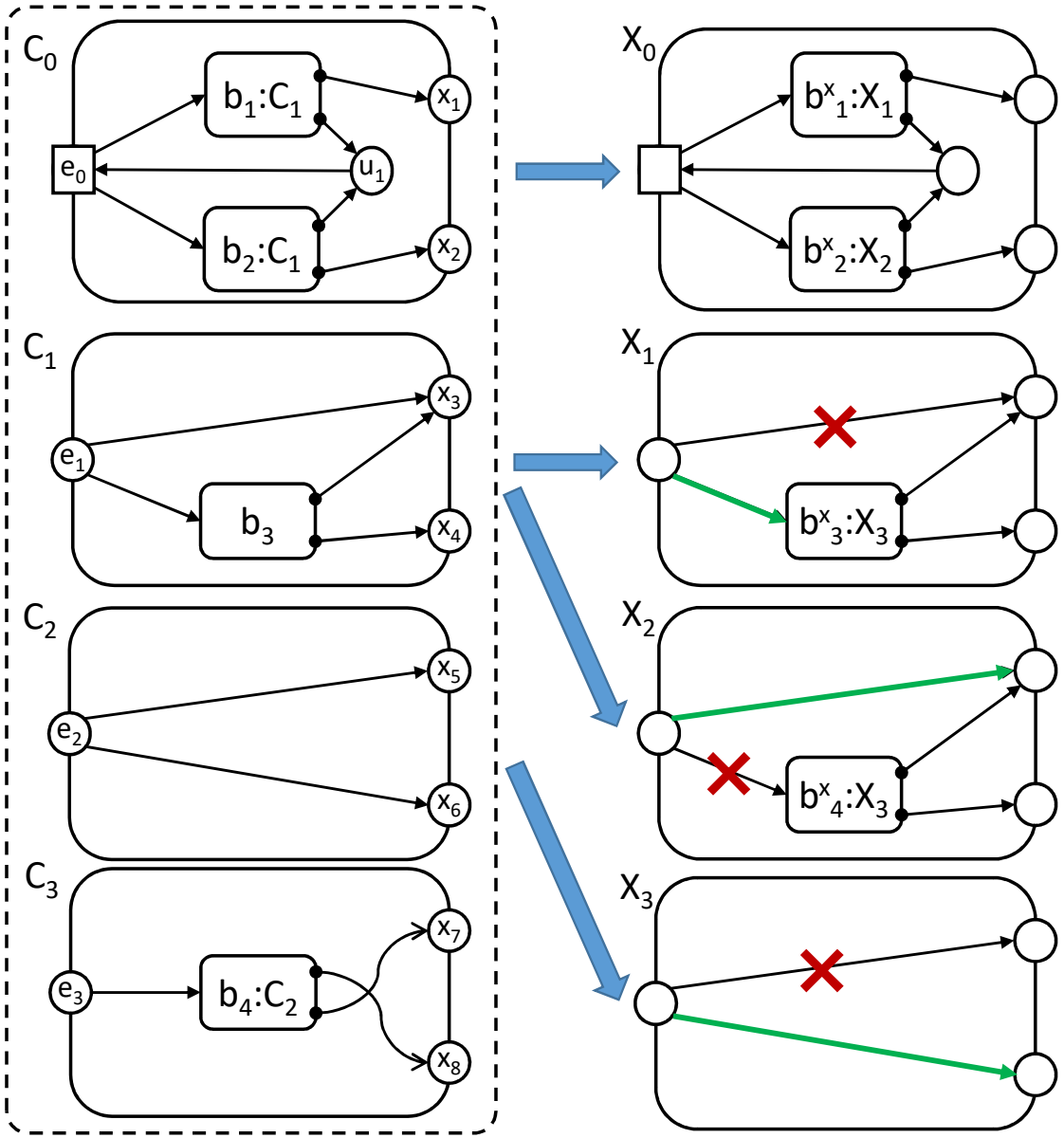
- Same as RSM except for
  - Boxes (possibly unmapped)
  - Vertices can be
    - ∃-vertex (outgoing edges are controllable)
    - ∀-vertex (outgoing edges are not controllable)

- Instance of a Library is RSM where
  - each module is component + local strategy

  (strategy is composed of a function for each module)
  - all boxes gets mapped

    (selection of modules to call)





Library instance

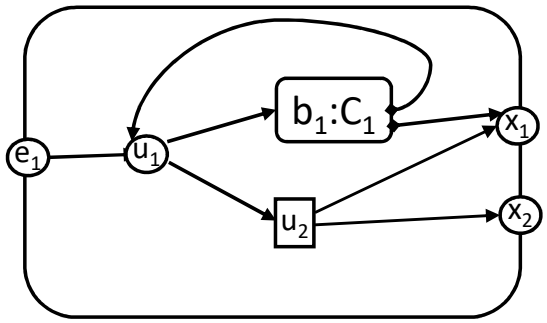
- each module corresponds to a library component (multiple copies allowed)
- box-mapping agrees with library box-mapping
- moves from ∃-vertex are selected by a local strategy

not required be memoryless

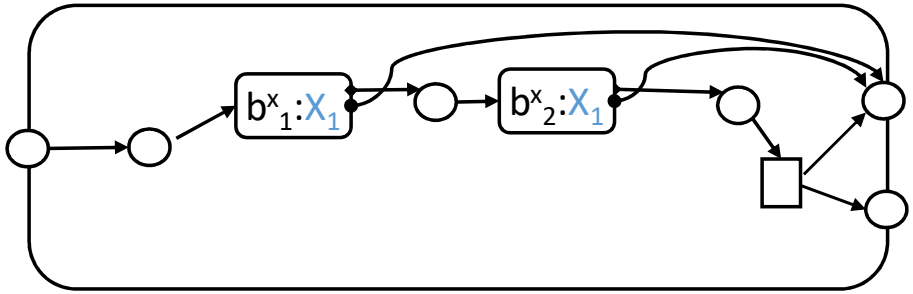
Library instance

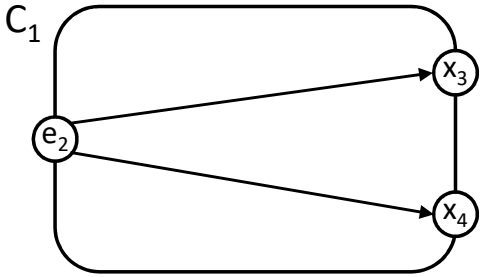
- Box copies obtained by unwinding loops (non-memoryless strategy) must be mapped to same module

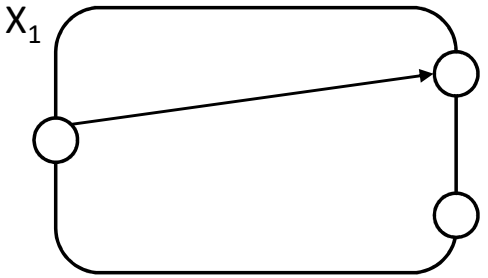
Arena



Instance

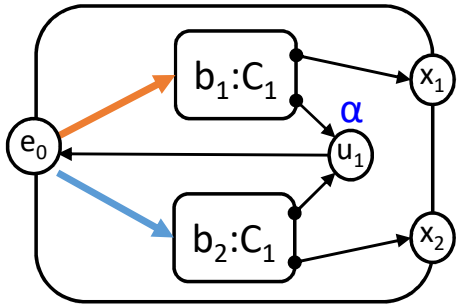


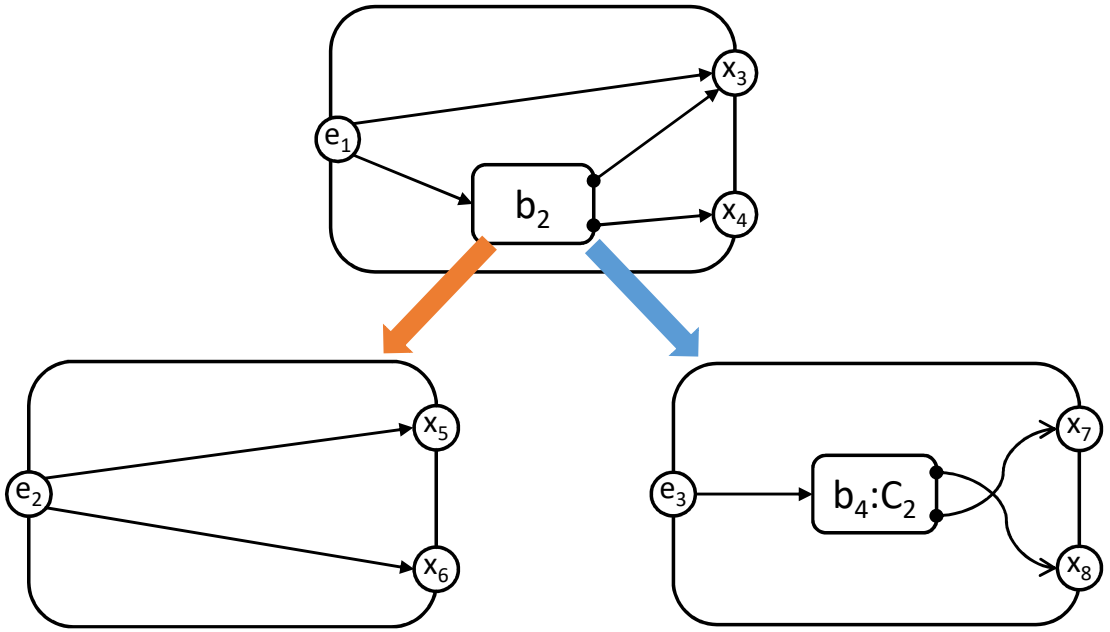




Note

- Controllable actions are of two kinds:
  1. intra-module actions (as in game graphs)
  2. inter-module actions (module call selection)



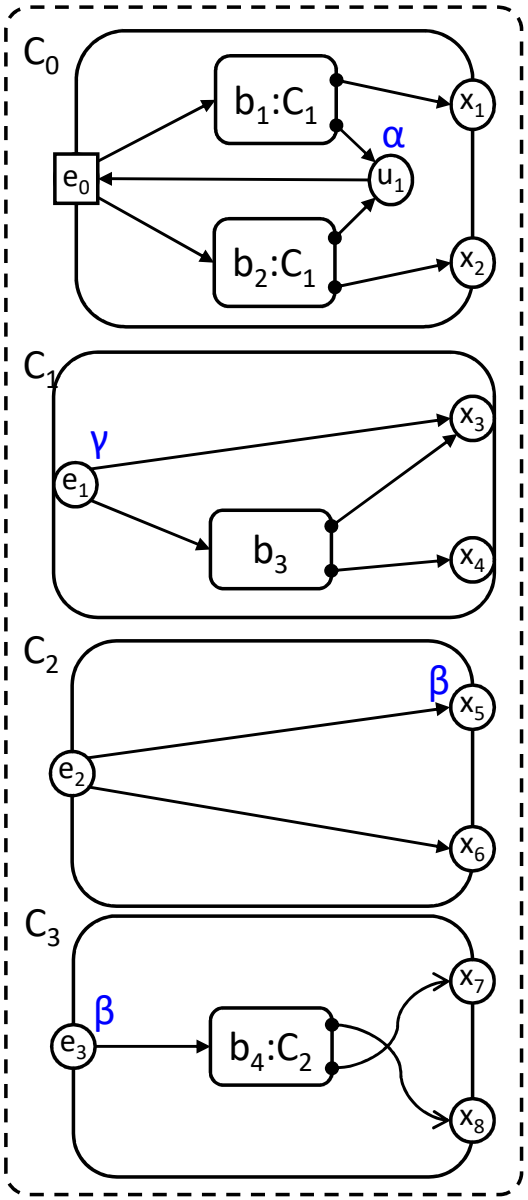


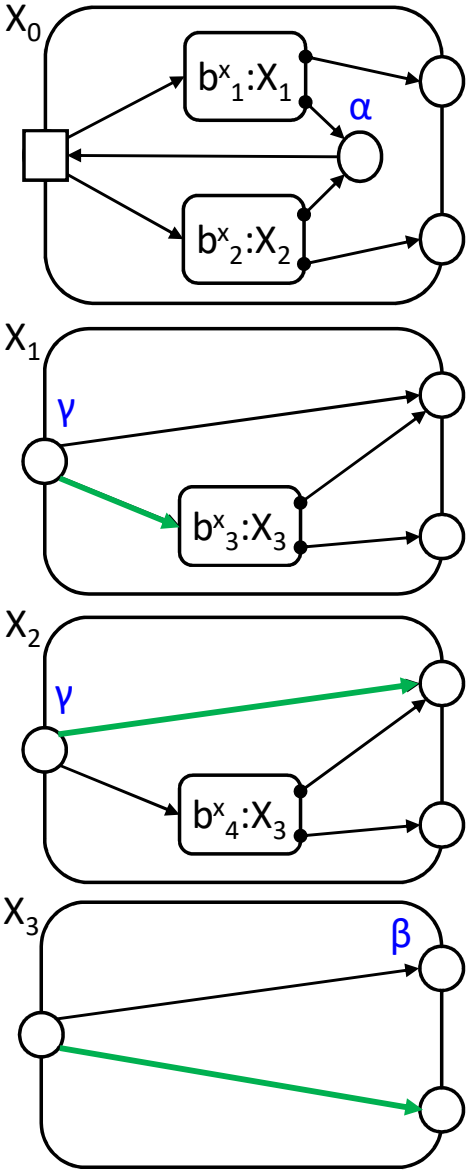
Strategy vs. Modular strategy

- A strategy has a total view of the play

- A modular strategy is
  - local to each module:
    controllable actions are selected only on the basis of the local play
    - it does not account for moves taken into other modules

  - not persistent through different module activations:
    each module activation has its own local play
    - it is oblivious of the moves taken in previous activations





Specs

- Language of traces
- Requirement:
  traces of resulting system must be included in Spec
- Ex. $(\gamma\alpha)^{\omega}$

Modular Pushdown Synthesis (MPS)

- Given a modular PD arena and a specification,

  is there a winning **modular** strategy?

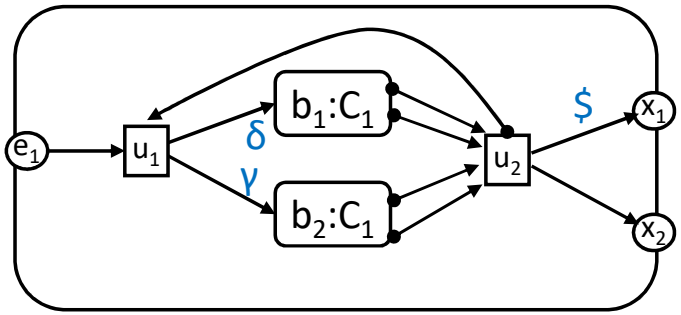
- Specification:
  - Reachability
  - Safety
  - Visibly PDS
  - Buchi automata
  - Linear temporal logic: LTL, CARET

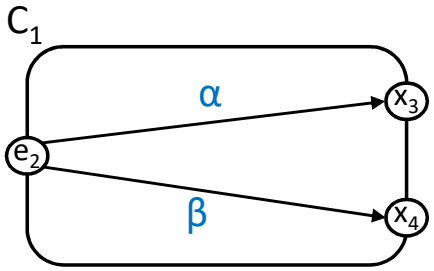
Rest of the talk

- Persistent modular strategies: undecidability
- Decidability of Safety-MPS
- Other Specs and MPS variations
- Conclusions and future work

Persistent modular strategies

- Modular strategies now can refer to local plays of previous module activations
- Ex.





- Local strategy for C1 could alternate between α and β in consecutive calls to same module
- L is language of w s.t.
  - w is in ((δαγα)+(δβγβ))*$
  - or projection of w over {δ,γ,$} is NOT in (δγ)*$
- Strategy is winning iff modules called from b1 and b2 generate the same sequence in (α+β)*

Undecidability of persistent-MPS

- Post Correspondence Problem (PCP):
  - Given n pairs of words $(u_1,v_1),\ldots,(u_n,v_n)$
    is there a sequence of indices $i_1 \ldots i_m$ s.t. $u_{i_1} \ldots u_{i_m} = v_{i_1} \ldots v_{i_m}$?
- We can design an arena as in the previous example
  - one module is in charge of generating a sequence U using the u's the other a sequence V using the v's
- Enforcing U=V can be done as before (by synchronizing the two modules on each produced symbol)
- Additionally, we also need to ensure that the two modules use the same sequence of indices
  - this can be done advancing one module up to the next word and then the other, and so on…

Observation

- Locality and persistency of modular strategy make PD games very powerful

- Persistent-MPS have the same flavour of concurrent games with partial information
  - the two modules indeed act as two processes and the main module as a scheduler

- Complexity results of several concurrent games:

G.L. Peterson and J.H. Reif, “Multiple-Person Alternation”, FOCS’79

Rest of the talk

- Persistent modular strategies: undecidability
- Decidability of Safety-MPS
- Other Specs and MPS variations
- Conclusions and future work

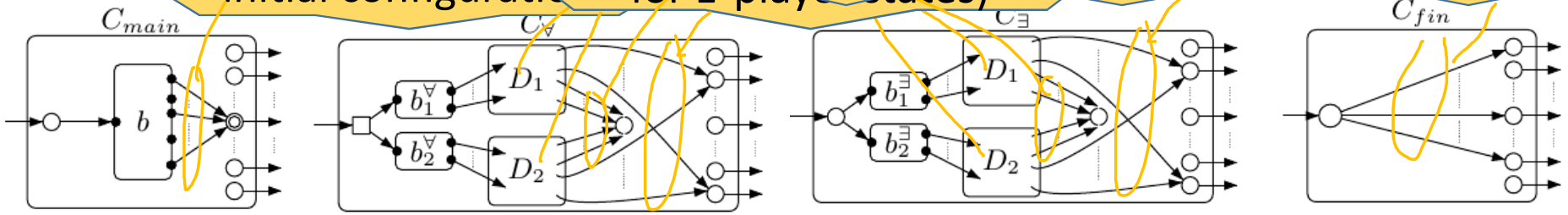
Safety-MPS

- Specification is given as a deterministic finite state automaton
- Strategies are modular (local + not persistent)
- Deciding Safety-MPS is EXPTIME-Complete
  - Lower bound:
    direct reduction from alternating linear-space Turing machines
    (reachability specs suffice)
  - Upper bound: reduction to emptiness of Büchi tree automata
    - Component/Library tree
    - Pre-post requirements
    - Overall construction

Exptime-hardness (gist)

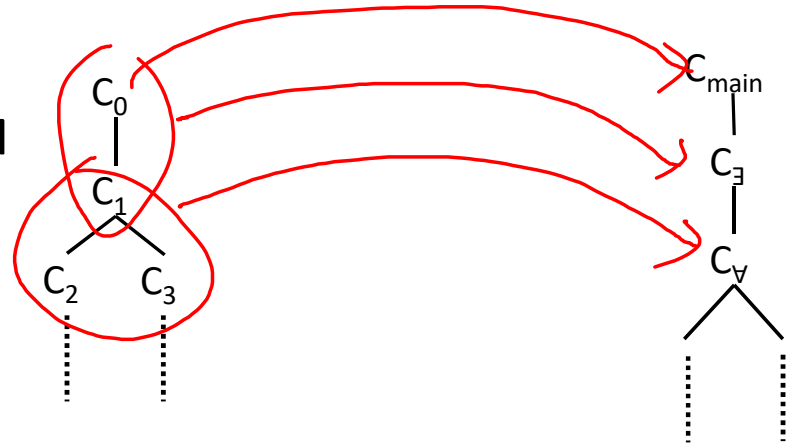
- Assume TM with 2 transit
- Fo

edges to targe
initial configurati
corre
for ∃ play
connect return to
nding exit (
other returns
go to sink
v-player states)
entry connected
to final states



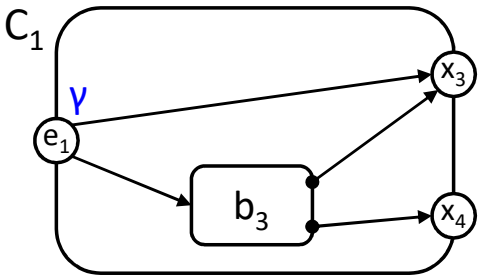
- Config. ecoding: exit 1-to-1 to cell index + content

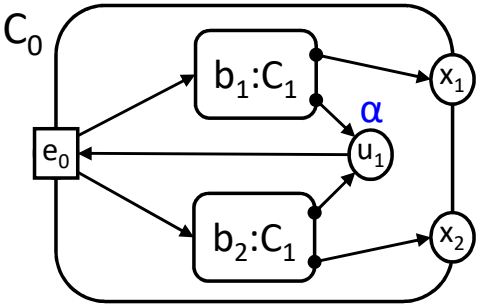
alternating TM computation



Box-to-instance mapping (types)

Component tree

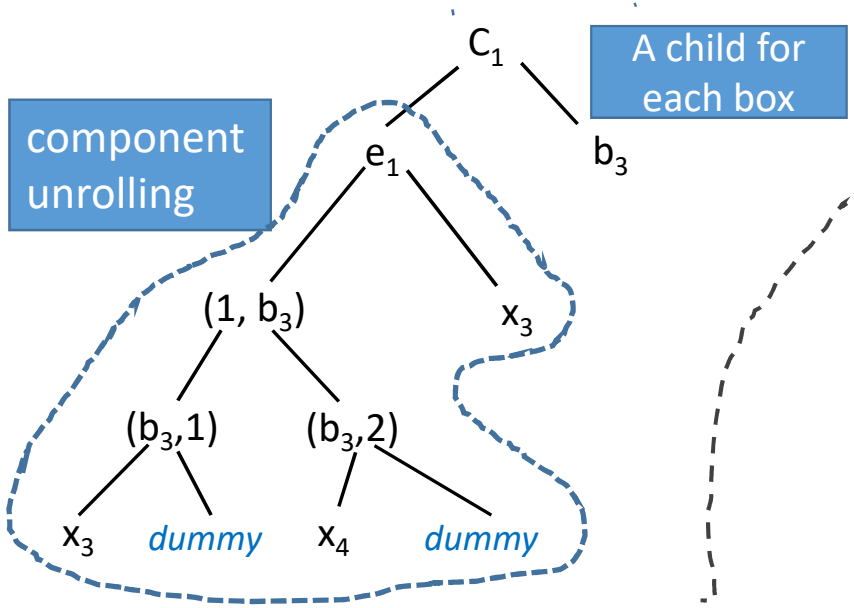


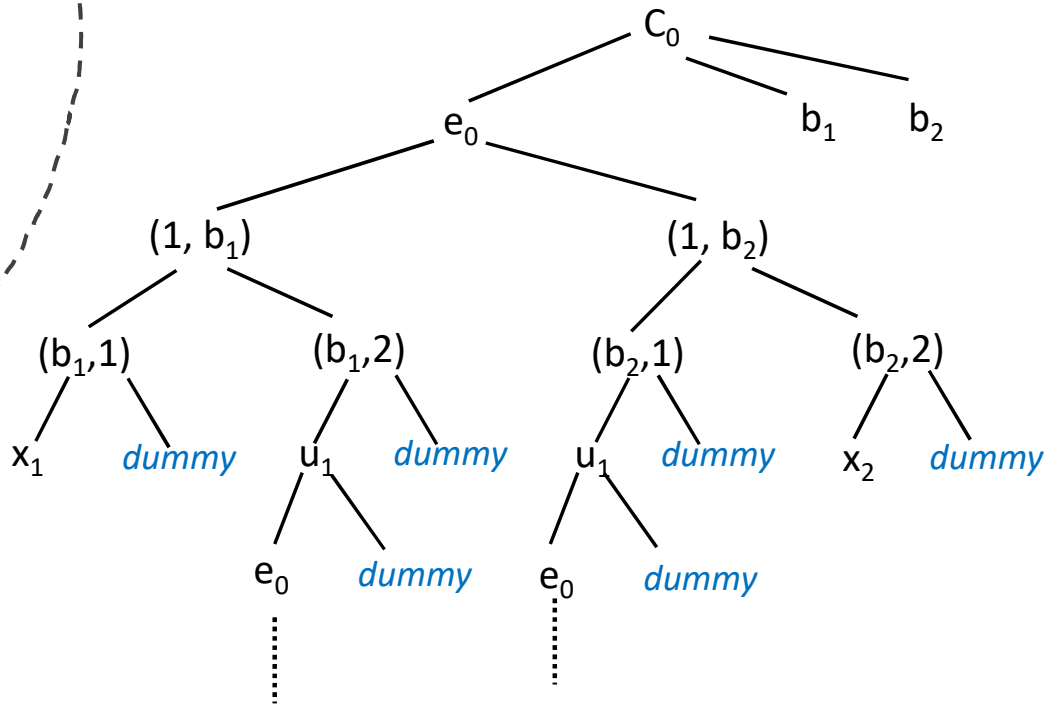


Root label: component id

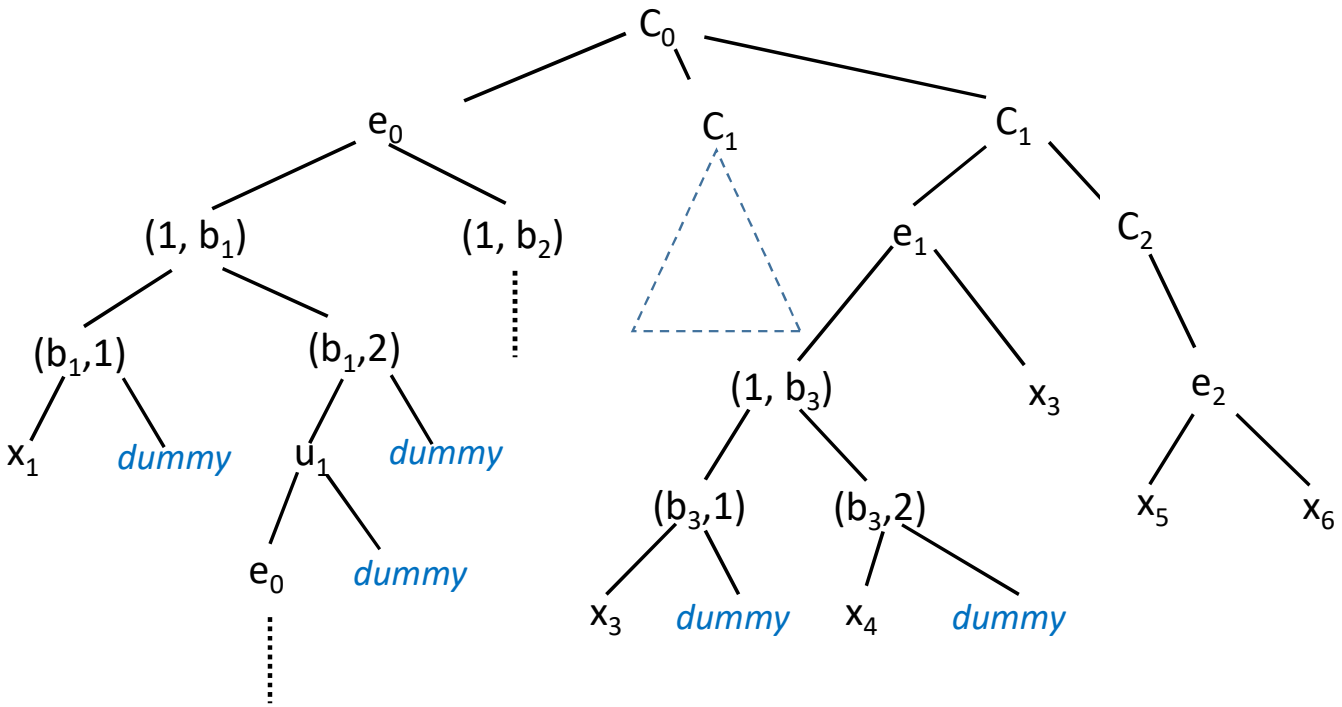
A child for each box

component unrolling

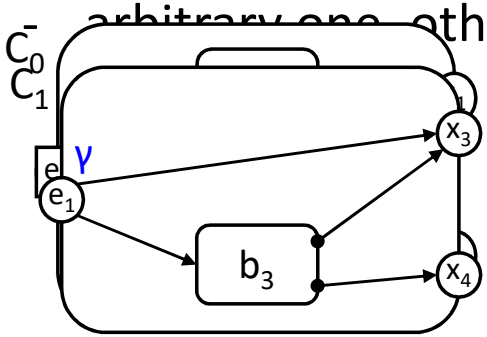




Library tree (ω-concatenation of component trees)



- Start with the main component tree
- Iteratively replace each box with component tree
  - specific one, if box mapped
  - arbitrary one, otherwise



The set of library trees is regular

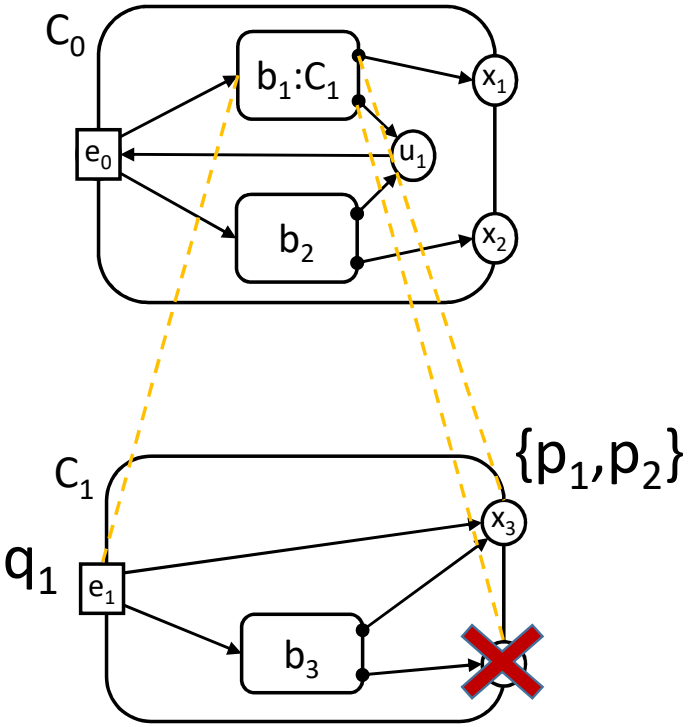
Determining the satisfying modular controller

- Component tree gives the module skeleton
  - Local strategy is needed to get the synthesized module
- Library tree
  - keeps the skeleton of each module and
  - guesses a full box-mapping (module invocations are fixed by library tree)
- Wat's left?
  - Select a local controller
  - Ensure the fulfillment of specifications

Successor is nondeterministically guessed at each ∃-vertex

simulate spec automaton

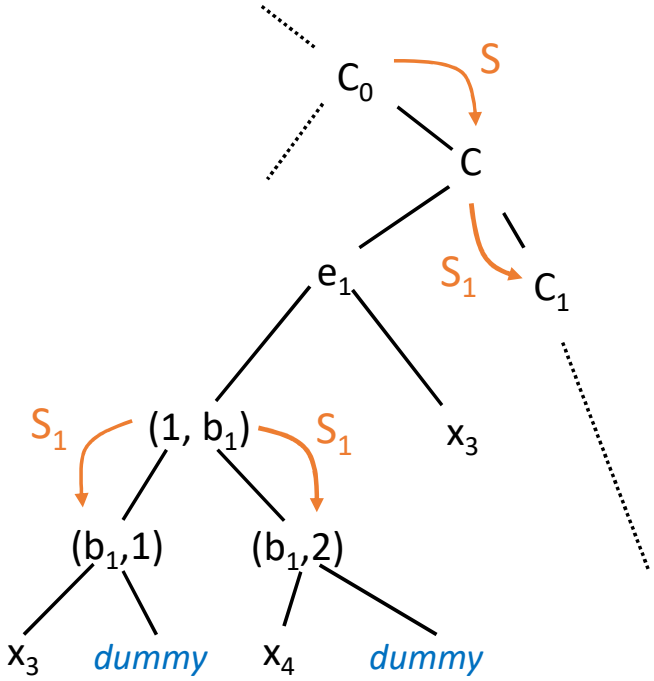
Simulating spec automaton on library tree



- Internal moves are simple:
  - just update the state according to FA transition on
    - selected successor (∃-vertex)
    - all successors (∀-vertex)
- Moves from a call to a matching return requires simulation on all paths in the recursively invoked modules
  - we use summaries

    Ex. $S=\{ (q_1,x_3,p_1), (q_1,x_3,p_2), (q_2,x_3,p_3) \}$

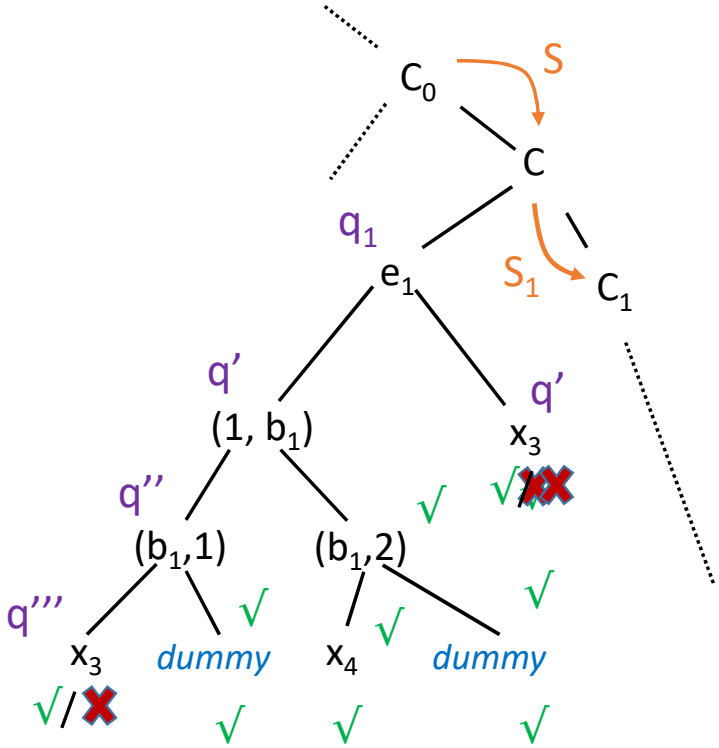
Summary (pre-post condition)



- Receive pre-post condition S from parent
- Check S on C and guess S_1
  (guess a summary for each module box)
- S_1 is
  - passed to C_1 to be checked
  - used in C to cross boxes mapped to C_1

  (do for each guessed summary)
- Assume-guarantee reasoning:
  - Assume S_1 (i.e., guessed summaries)
  - Guarantee S (i.e., received summary)

Checking summaries



- Example:
  - e_1 is a ∀-vertex
  - $S=\{ (q_1,x_3,p_1), (q_1,x_3,p_2), (q_2,x_3,p_3) \}$
  - $S_1= \{ (r_1,ex_1,s_1)\}$
    -- ex_2 is not reachable

if $q' \xrightarrow{\lambda(1,b_1)} q_1 \in \{p_1, p_2\}$, accept (S holds)

otherwise reject

S post-condition for q_1

S pre-condition

Apply guess S_1

Wrapping up

- Construction:
  - A_1 checks that input is a valid library tree (regular set of trees)
  - A_2 checks there exists a modular strategy (with possibly infinitely many modules) that fulfils specs and is structured on input
    (it assumes input is a library tree)

- Thus, $A_1 \cap A_2 \neq \emptyset$ iff a winning modular strategy exists
  (finiteness and synthesis is a consequence of Rabin theorem)

- Overall size is exponential, thus deciding Safety-MPS is in EXPTIME

Rest of the talk

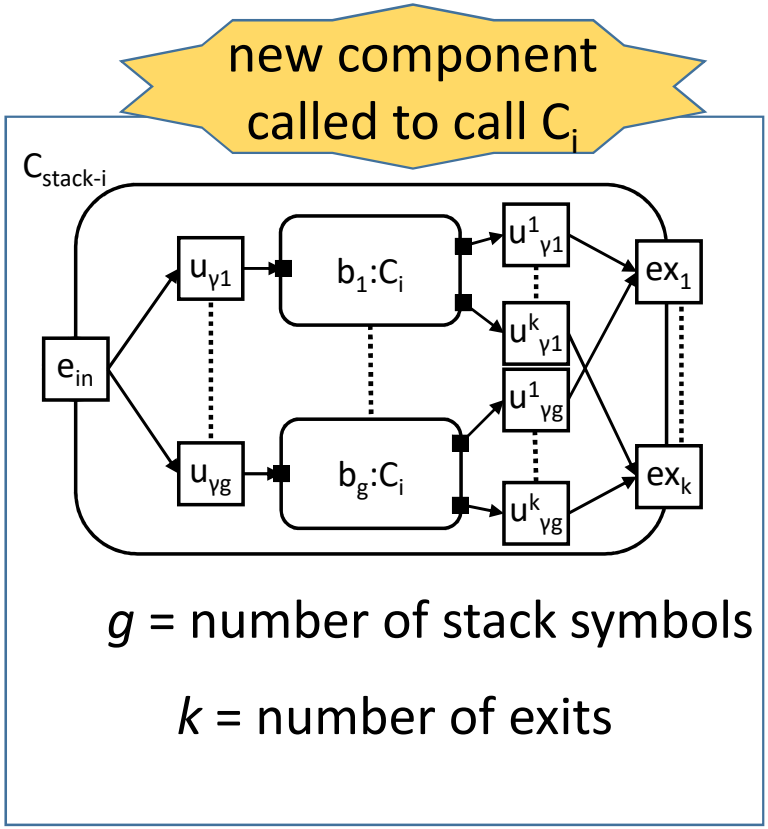
- Persistent modular strategies: undecidability
- Decidability of Safety-MPS
- Other Specs and MPS variations
- Conclusions and future work

Reachability-MPS

- Specification is given as set of vertices
- Deciding Reachability-MPS is EXPTIME-Complete
  - Lower bound already discussed
  - Upper bound: from Safety-MPS
- We can give a simpler (direct) fixed-point decision algorithm:
  - vertices are discovered backward starting from target
  - algorithm computes tuples $(u, E, \{\mu_b\}_{b \in B})$ meaning that:
    - there is a strategy leading from u that
    - reaches exits within E
    - by crossing boxes b jumping to its returns μ_b

MPS with Visibly Pushdown specs

- Specification is a deterministic visibly pushdown automaton
- Deciding VPA-MPS is EXPTIME-complete
  - lower bound: FA is a VPA
  - upper bound: reduction to Safety-MPS
- Main idea of reduction:
  - Incorporate the VPA stack in the arena
  - Each call to a module goes though a new component that store stack symbol
  - FA keeps the control state and the top-stack symbol of VPA



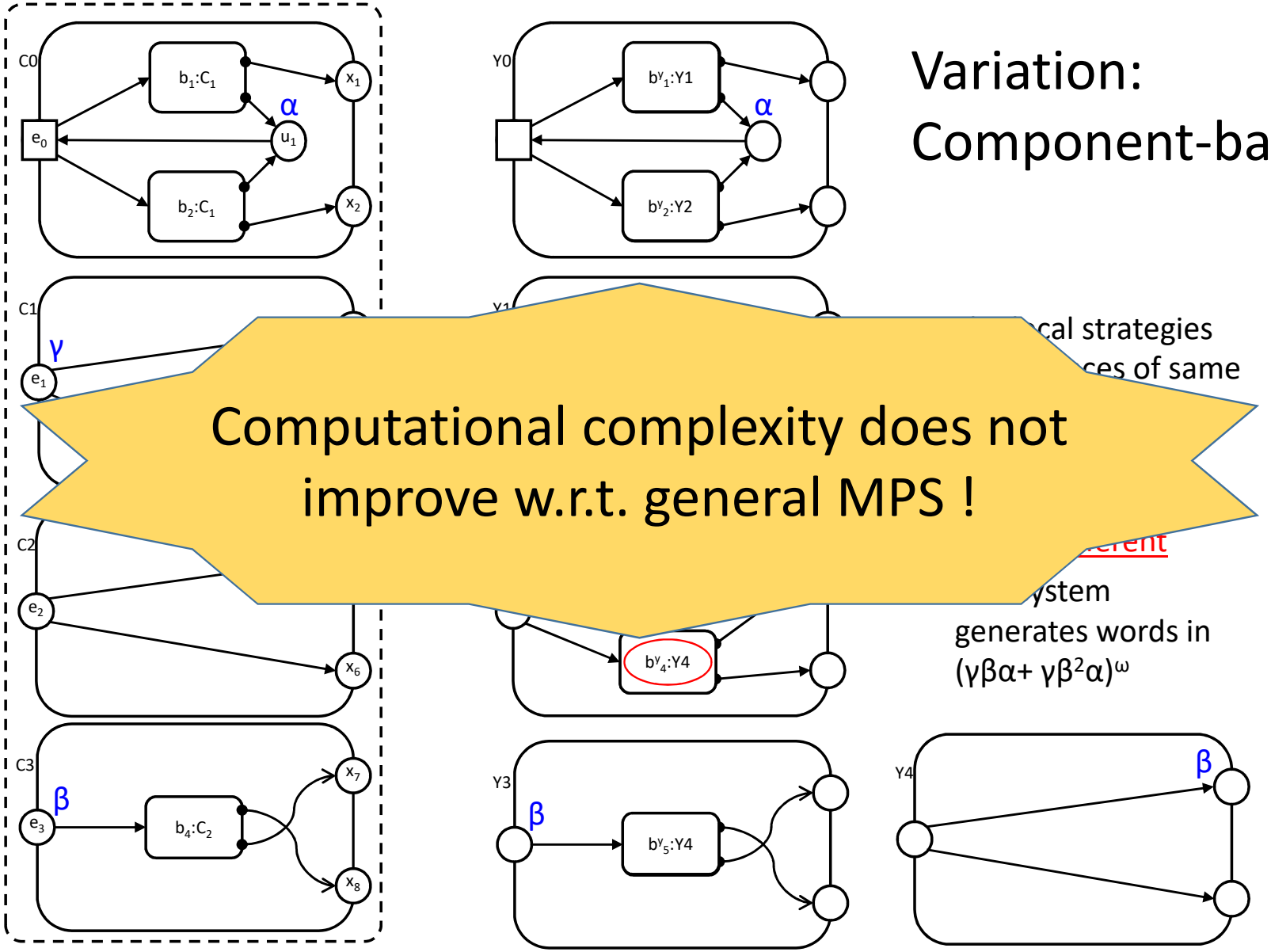
More specs

- MPS with nondeterministic FA/VPA specs is in 2EXPTIME
  - reduces to Safety/VPA-MPS through determinization

- MPS still EXPTIME-complete when specs are extended with
  - Büchi /co- Büchi acceptance --adapt acceptance set of tree automaton
  - universality of model (dual to nondeterminism)
    --use set of spec states in tree automaton

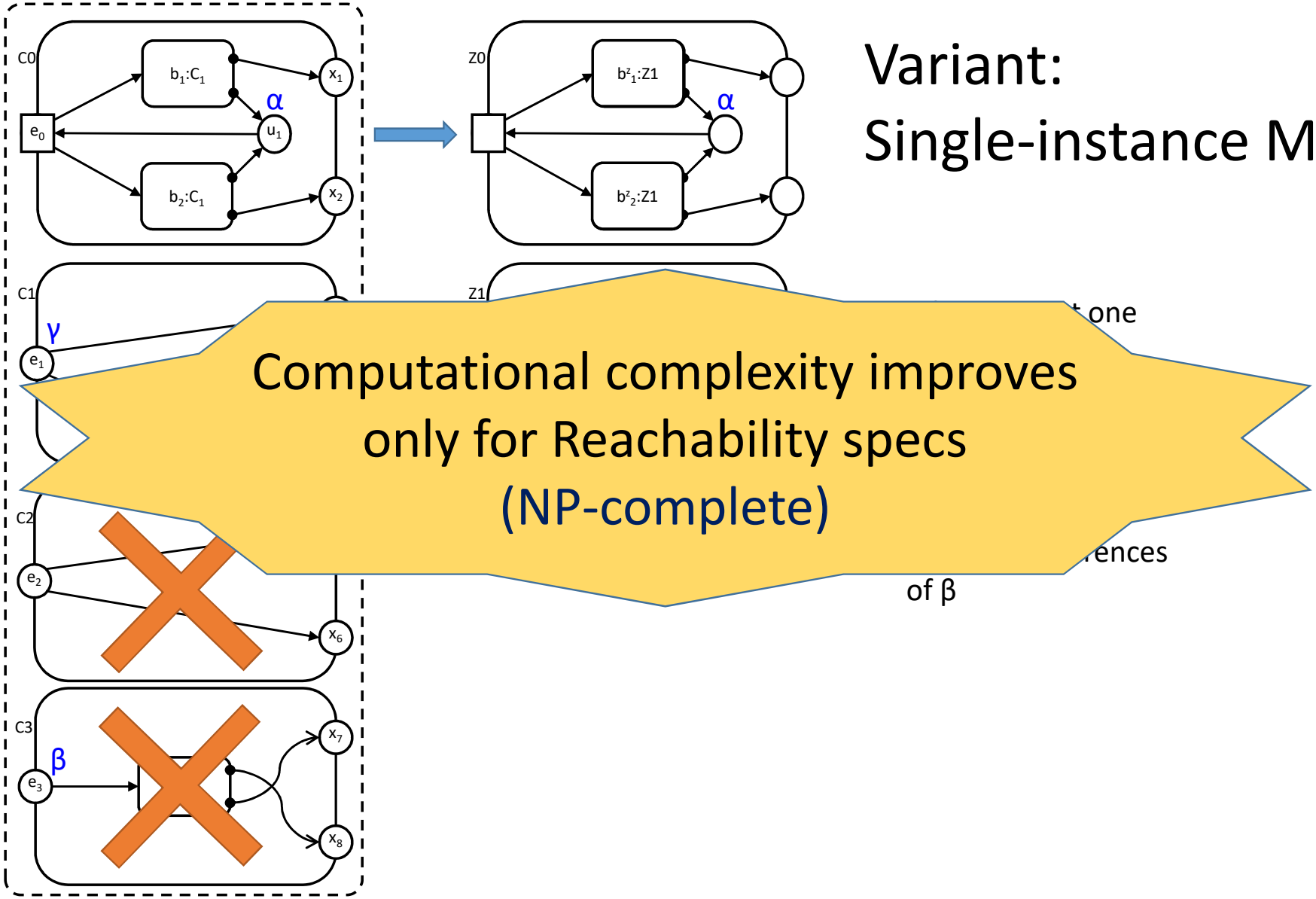
- MPS with temporal logic specs (LTL, CARET, NWTL) is 2EXPTIME-complete
  - Upper-bound: reduce to MPS with Büchi universal specs
  - Lower-bound: inherited from standard LTL games
    --for MPS hardness holds already for bool(PATH-LTL)

Variation: Component-based MPS



Computational complexity does not improve w.r.t. general MPS !

cal strategies
ces of same
erent
ystem
generates words in
$(\gamma\beta\alpha+ \gamma\beta^2\alpha)^{\omega}$



Variant: Single-instance MPS

Rest of the talk

- Persistent modular strategies: undecidability
- Decidability of Safety-MPS
- Other Specs and MPS variations
- Conclusions and future work

Conclusions

- General modular synthesis problem for pushdown systems
  - game both inside module and in the module composition
  - local controllers are oblivious of previous invocations

- A natural setting for automatic program synthesis:
  - capable of yielding solutions with control-flow structure of real programs (which are formed of functions)

- Restricting to modular solutions yields computational gain only for:
  - reachability with single-instance restriction (EXPTIME→NP)
  - temporal logic specs (3EXPTIME→2EXPTIME)

Talk is based on

1. I. De Crescenzo and S. La Torre, “A General Modular Synthesis Problem for Pushdown Systems”, VMCAI’16
2. I. De Crescenzo, “Synthesis of Pushdown Systems”, PhD Thesis, Università degli Studi di Salerno, 2016.
3. I. De Crescenzo and S. La Torre, “Modular Synthesis with Open Components”, RP’13

4. R. Alur, S. La Torre, P. Madhusudan, “Modular Strategies for Recursive Game Graphs”, Theor. Comp. Scie.’06 ---undec. of persistent MPS

5. I. De Crescenzo, S. La Torre, and Y. Velner, “Visibly Pushdown Modular Games”, Info&Comp’17 ---2EXPTIME-hardness simple TL specs

Modular Synthesis on Recursive Game Graphs

- Only controllable intra-module actions
  - Call-return structure is fixed

- RGG are recursive state machines where vertices are split into controllable and uncontrollable

- Special case of Single-instance MPS

- References:

  1. R. Alur, S. La Torre, P. Madhusudan, “Modular Strategies for Recursive Game Graphs”, TACAS’03-Theor. Comp. Scie.‘06
  2. R. Alur, S. La Torre, P. Madhusudan, “Modular Strategies for Infinite Games on Recursive Graphs”, CAV’03
  3. I. De Crescenzo, S. La Torre, and Y. Velner, “Visibly Pushdown Modular Games, GandALF’14-Info&Comp’17

Synthesis from Libraries of Components

- Only controllable inter-module actions
- Modules are already synthesized except for some function calls that are left unspecified (no intra-module game)
- Special case of Component-Based MPS

- Refrences:
  1. Y. Lustig, M.Y. Vardi, “Synthesis from Recursive-components Libraries”, GandALF’11
  2. Y. Lustig, M.Y. Vardi, “Synthesis from Component Libraries”, FOSSACS’09

Synthesis of Reactive programs

- Goal is to sinthesize a Boolean program (imperative program with only boolean type) with possibly recursive function calls
- Number of modules and variables of the program to synthesize are parameters of the problem
- Difference with MPS:
  - modules are guessed not derived from library open components
  - result is a program with variables
  - searches for solutions with a bound on the number of modules (similarly to MPS with the single-instance restriction)

- Refrence:
  1. P. Madhusudan, “Synthesizing Reactive programs”, CSL’11

Future directions: automatic program synthesis

- Natural domain of application for MPS is automatic program synthesis
  - Main drawback: high computational complexity

- Possible way to go: focusing on small parts of programs (few modules)
  - Ex. synthesize the main module given some library functions or
    synthesize a called module given the rest of the program

- Application of synthesis from libraries to synthesize simple modules (i.e., sequence of function calls) is explored in

  "Oracle-guided component-based program synthesis", S. Jha et al., ICSE'10

  - A simplified MPS setting could be used to yield more structured solutions

Future directions: program repair

- Program repair attempts to repair a faulty program with small fixes

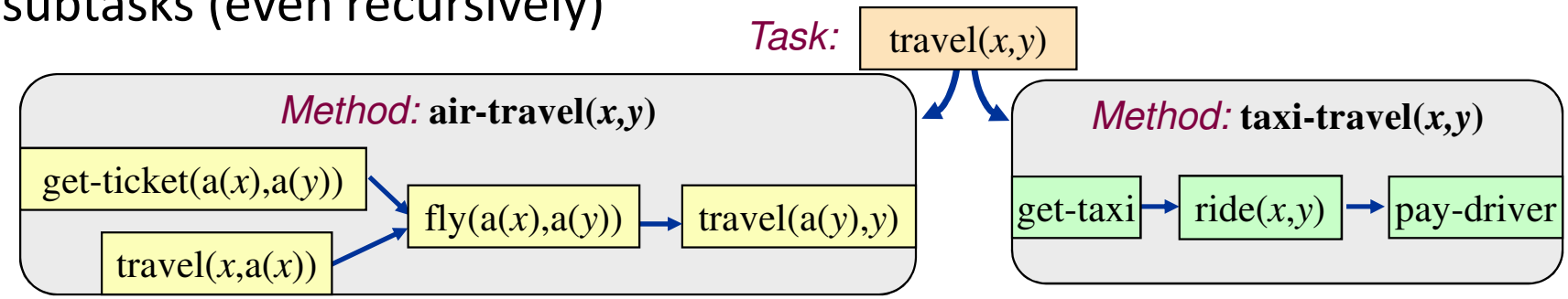
- Program repair with fault localization and correction by infinite game
  - attempts to fix a faulty function
  - game only involves intra-module actions

- “Finding and fixing faults”, B. Jobstmann et. al., J. Comput. Syst. Sci. ’12

- MPS setting could allow also to fix bugs by replacing a function call
  - the repair game can be extended to inter-module actions (replace a faulty function instead of fixing it)

Future directions: planning

- Hierarchic Task Networks model scenarios where tasks are composed of subtasks (even recursively)



- Planning problem: given a specification, determine a plan (a composition of tasks) such that its execution satisfies the specification

- MPS seems suitable to model such planning problem
  - Hiearchic Task Network can be modelled as an Open Component Library
  - The plan would be the synthesized RSM