Bounded Synthesis

Presentation: Steffen Metzger

Same problem as before

- Given some specification (e.g. in LTL)
- Automatically synthesize a program that acts according to the specification
- We just saw this is possible in some cases

Decidability

- But, distributed synthesis in general **undecidable**

Idea: Use bounds to iteratively approach the problem,
allowing larger and larger solutions
and finally generate a minimal solution
(if a solution exists at all).

Other Advantages

- Deal with real-world restrictions on implementations
- Obtain a smaller solution space as output by concentrating on a small (realizable) subset of solutions
- Can we still do it for distributed architectures?

Yes, we can!

Bounded Synthesis Overview

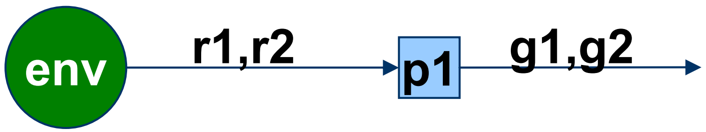
- Get specification as **universal co-Büchi automaton** (e.g. LTL→Büchi→co-Büchi)
- The acceptance of an implementation can be characterized by **existence of some special annotation**
- Finally **synthesize by solving a constraint system** representing the properties of that annotation (e.g. in **SMT**)

Example

- Consider a pedestrian crossing
- Environment can issue 2 different access requests
  - pedestrian pushing the pedestrian light button
  - car arriving on contact line
- Find an implementation that guarantees they are not granted access simultaneously

Architectures

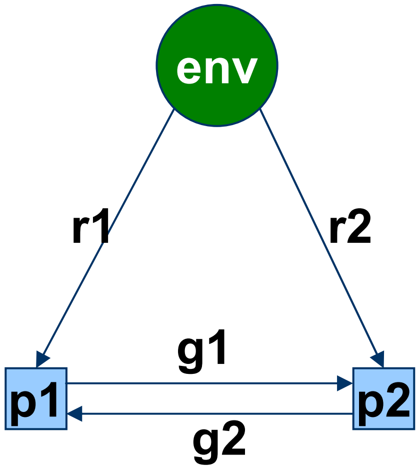
- V = set of boolean system variables
- $\{I_p \subseteq V \mid p \in P\}$ a family of Input variable sets
- $\{O_p \subseteq V \mid p \in P\}$ a family of Output variable sets



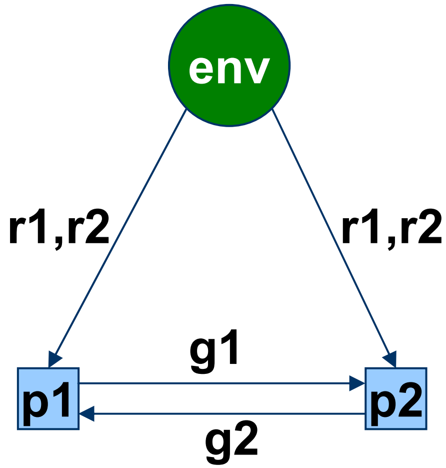
single process

- here: $O_{env}=\{r_1,r_2\}$

Distributed Architectures



2-process arbiter



2-process arbiter

fully informed

$O_{env} \subseteq I_p$

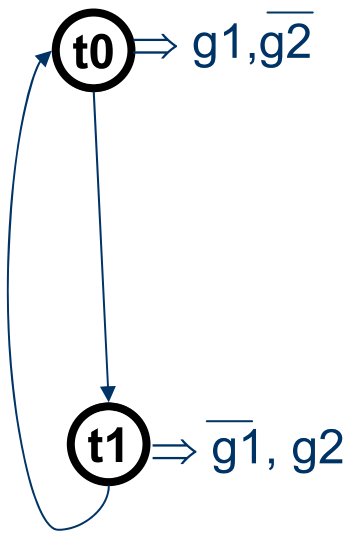
Implementations

- represented as **transition system**
- **each process** represented by one **independent transition system**
- **Merging** of all process systems **gives** a **composed system** for overall **properties/specification check**

transition systems

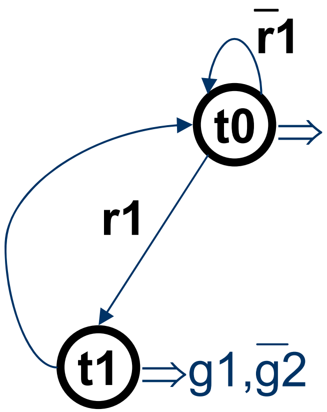
- Given directions Υ and labels Σ, a Σ-labeled Υ-transition system is a tuple $\mathrm{I} = (T, t_0, \tau, o)$ where:
  - T is a set of states
  - t_0 is an initial state
  - $\tau: T \times \Upsilon \rightarrow T$ is a transition function
  - $o: T \rightarrow \Sigma$ is a labeling function

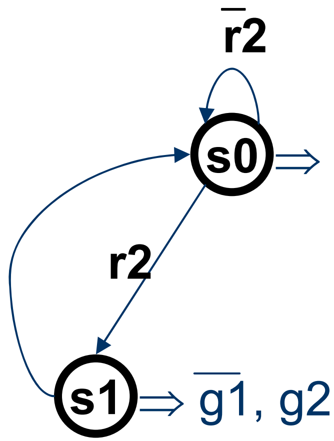
Single Process Example



2 Process Example

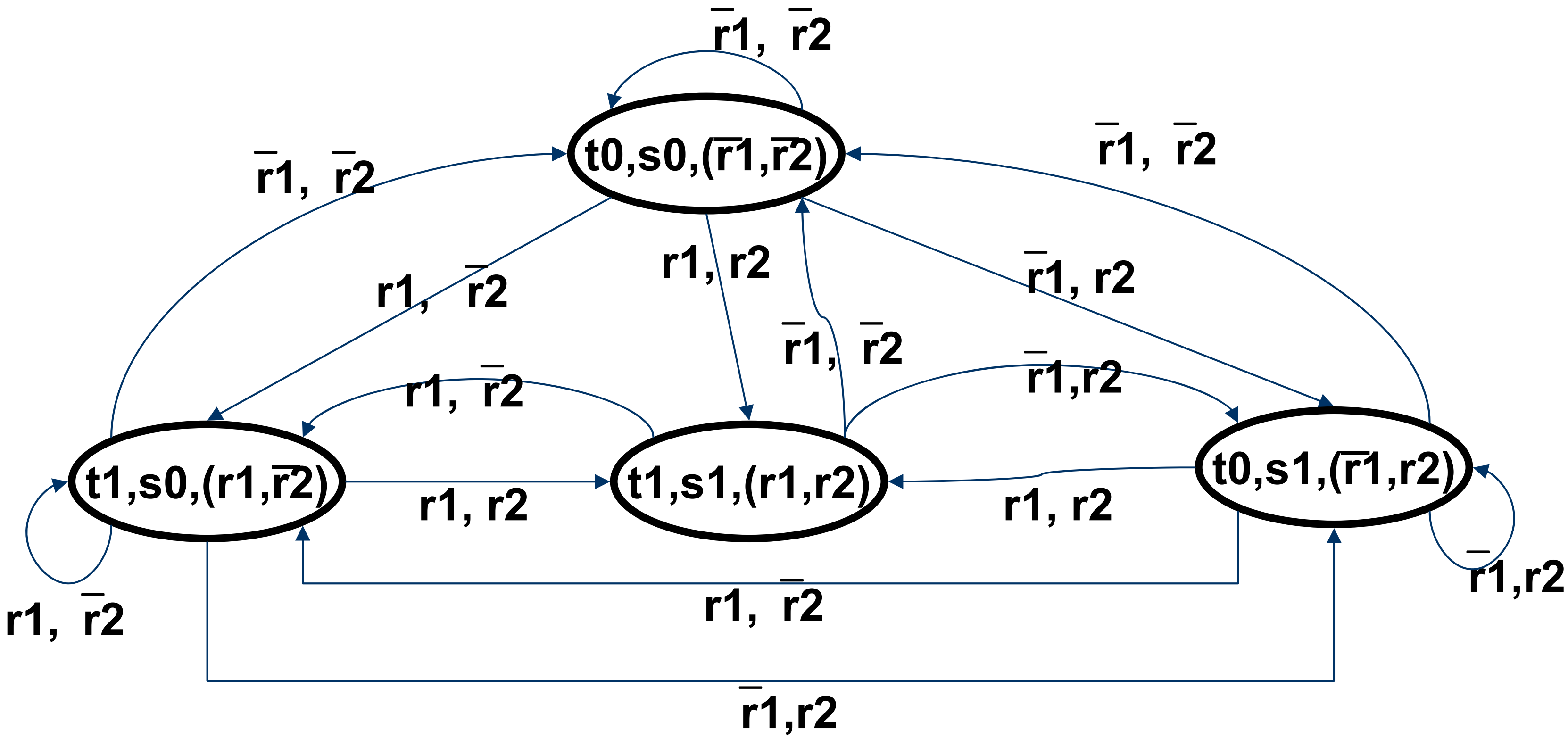
- $O_{env} = \{r1, r2\}$





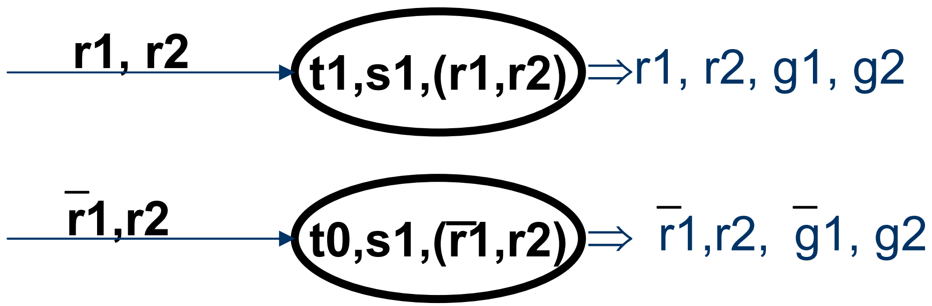
2 Process Composition Example

- $O_{env} = \{r1, r2\}$, root direction $\overline{r1}$, $\overline{r2}$



Input-preserving

- Labels composed of process labels, but also contain current input from Env, e.g.



input preserving

- We are only considering input-preserving transition systems in the following

Specification

- A specification φ is (finite-state) *realizable* in an architecture with processes P iff it exists a family of (finite-state) implementations $\{T_p \mid p \in P\}$ such that their composition T_A satisfies φ

Bounded realizable

- Given
  - architecture with processes P
  - family of bounds $\{b_p \in \mathbb{N} \mid p \in P\}$ for processes
  - bound b_A for the whole system T_A
- A specification φ is *bounded realizable* if there exists a family $\{T_p \mid p \in P\}$ such that:
  - T_p has at most b_p states for all $p \in P$
  - T_A satisfies φ
  - T_A has at most b_A states

co-Büchi automaton

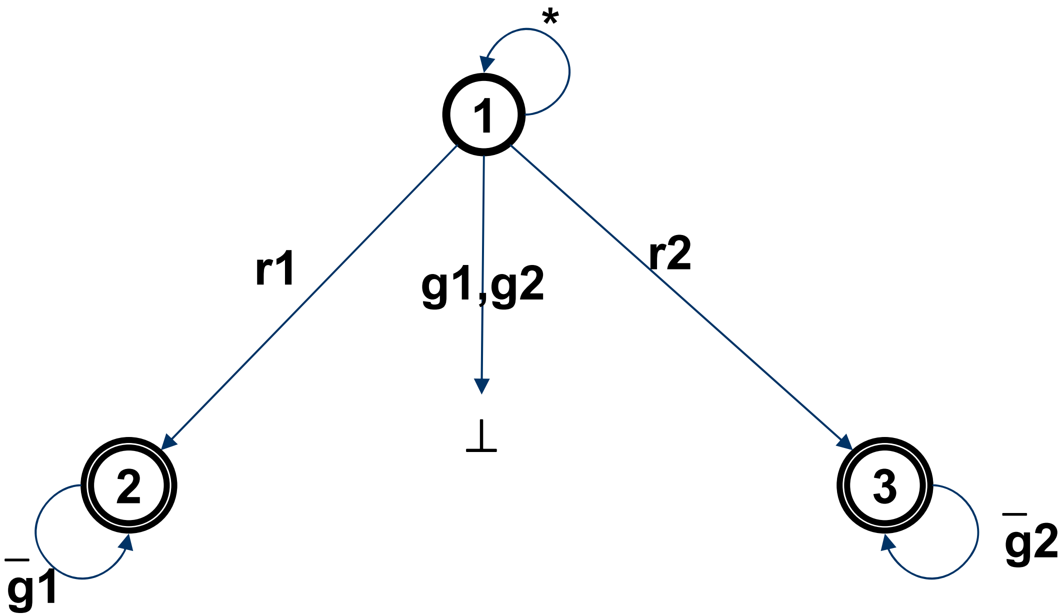
- A co-Büchi automaton B is given by a tuple $(\Sigma, \Upsilon, Q, q_0, \delta, F)$ where:
  - Σ denotes a finite set of labels
  - Υ denotes a finite set of directions
  - Q denotes a finite set of states
  - q_0 denotes an initial state
  - $\delta: Q \times \Sigma \rightarrow \mathcal{B}^+(Q \times \Upsilon)$ denotes a transition function
  - $F \subseteq Q$ denote rejecting states

Universal co-Büchi automaton

- A co-Büchi automaton $B = (\Sigma, \Upsilon, Q, q_0, \delta, F)$ is called ***universal*** iff for all states *q* and input letters *in*, **δ(*q,in*) is a conjunction**

- A **run *R*** in a co-Büchi automaton $B = (\Sigma, \Upsilon, Q, q_0, \delta, F)$ is **accepted** iff **rejecting states** ($r \in F$) appear only **finitely often in R**

Specification as universal co-Büchi



Run graph

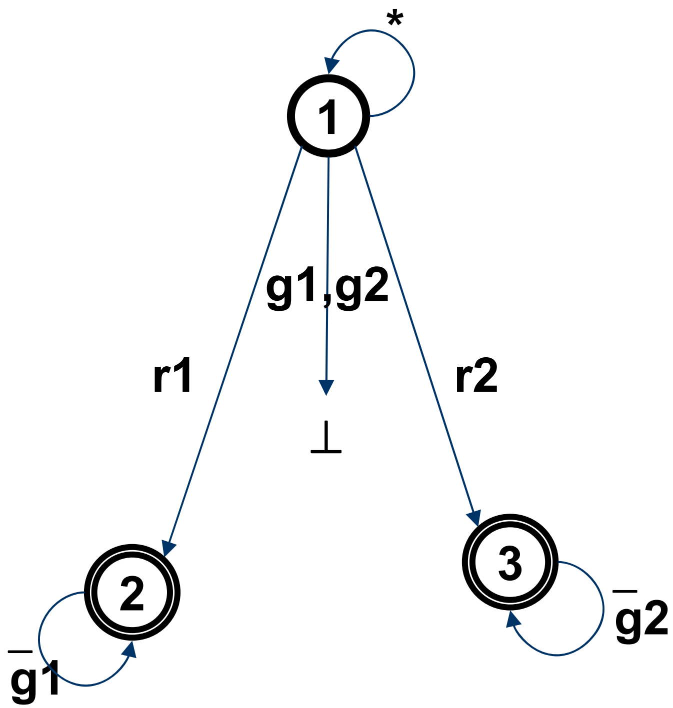
- A run graph of a co-Büchi automaton B on a transition system I is a minimal directed graph G=(V,E) that models all possible runs of B on I

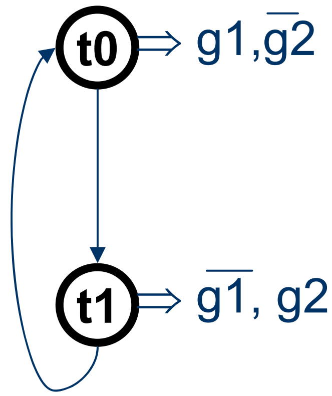
- A run graph is accepting iff

  in every infinite path,

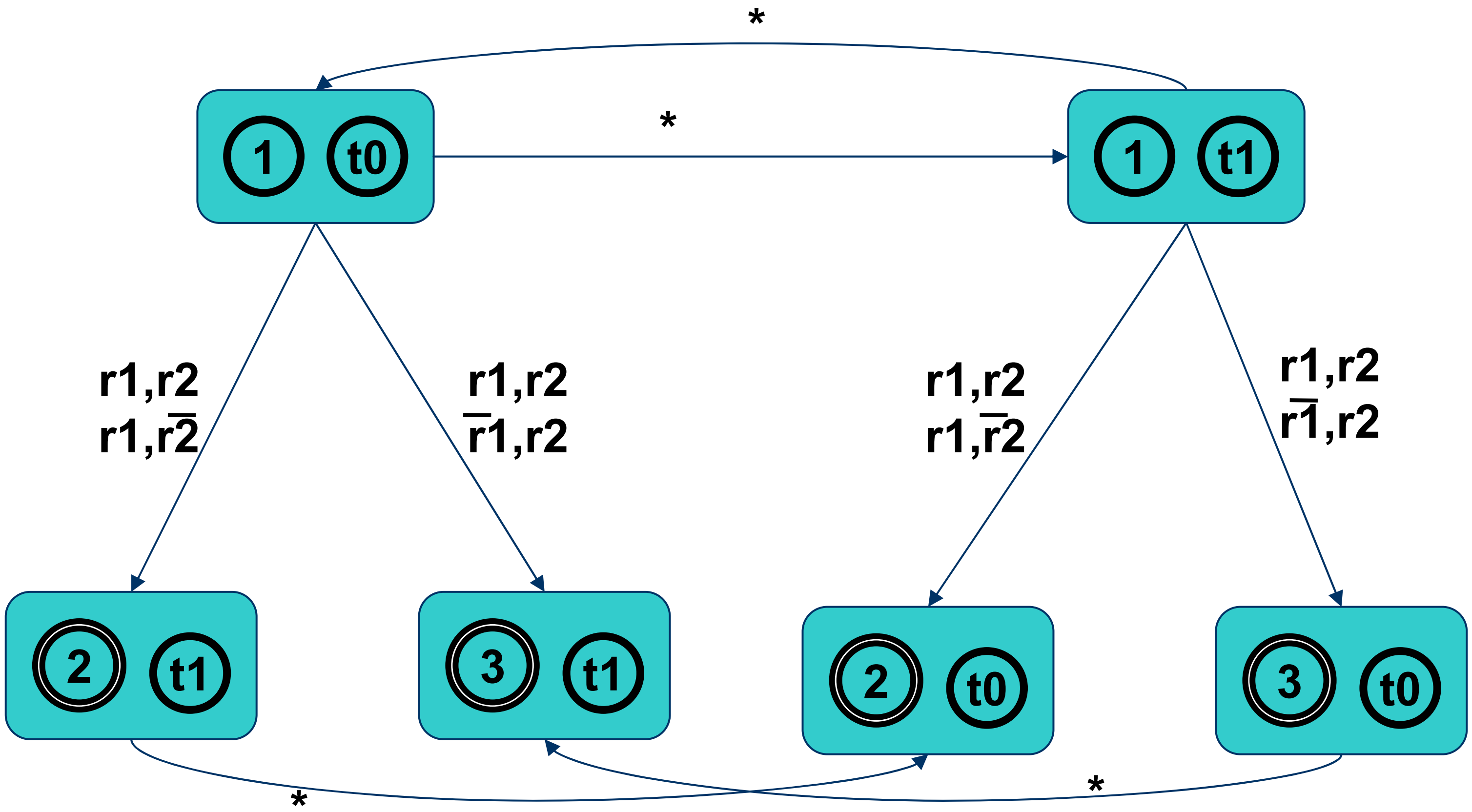
  states from F appear only finitely often

Run graph Example





Run graph example

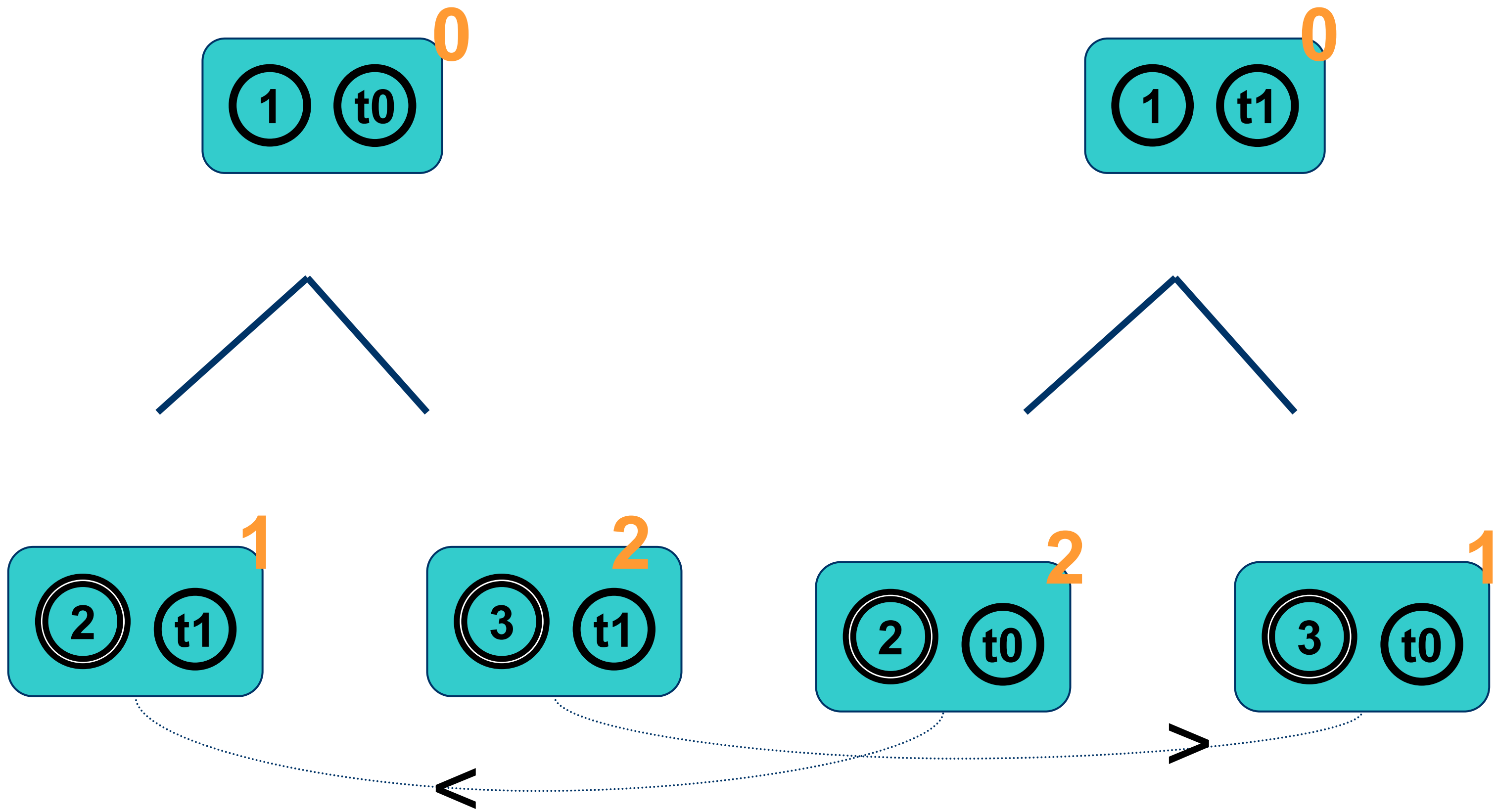


Check for accepting run graph

- Only need to decide if a path with infinitely many rejecting state appearances exists

Idea: Check if we can find a partial ordering on the run graph nodes, such that each path with rejecting nodes contains a maximal rejecting node, from which no further rejecting node is reachable.

Annotations



Annotations

- An **annotation** of a transition system $\mathcal{I} = (T, t_0, \tau, o)$ on a universal co-Büchi automaton $U = (\Sigma, \Upsilon, Q, q_0, \delta, F)$ **is a function λ: $Q \times T \rightarrow \{\_\} \cup \mathbb{N}$.**
- It is called *c-bounded* if its image is contained in $\{0,\ldots,c\}$ where $c \in \mathbb{N}$
- It is *bounded* if it is c-bounded for some c.

Valid annotations

- An **annotation is *valid*** iff
  - **all states reachable** from the initial state (q_0, t_0) are **annotated with a natural number**
  - **Values are not decreasing** along a possible path
  - **Values are increasing** from a state **towards some rejecting successor state**

Acceptance

Theorem:
A finite-state Σ-labeled Υ-transition system $\mathrm{I} = (T, t_0, \tau, o)$ is accepted by a universal co-Büchi automaton $U = (\Sigma, \Upsilon, Q, q_0, \delta, F)$ iff it has a valid $(|T| \bullet |F|)$-bounded annotation.

How to find an implementation

- Given a specification and some architecture
- How do we efficiently find an implementation with a valid annotation?

Idea: Describe the properties of the specification
and a valid annotation
in a **constraint system**, such that
solving the system provides an implementation.

Constraint system

- Given specification B = (Σ, Υ, Q, q_0, δ, F)
- Create a constraint system, such that any transition system $\mathrm{I} = (T, t_0, \tau, o)$ satisfying the constraint system satisfies B
- For now we only consider the **fully informed case** (equivalent to only one process)

Constraint system – some tools

- Some abbreviations to describe the constraints:
  - $\tau_v(t) = \tau(t,v)$
  - for all $\alpha \in V$. $\alpha(t)$ iff $\alpha \in o(t)$
  - for all $q \in Q$. $\lambda^{\#}_q(t) = \lambda(q,t)$ iff $\lambda(q,t) \in \mathbb{N}$
  - for all $q \in Q$. λ👌$_q(t)$ iff $\lambda(q,t) \in \mathbb{N}$

- where $\lambda(q,t)$ represents some annotation

Constraints

- $\forall\ \alpha \in O_{env},\ v \subseteq O_{env},\ t \in T.$
  - $\alpha(\tau_v(t))$ iff $\alpha \in v$
  - $\neg\alpha(\tau_v(t))$ otherwise

Input preserving

- $\lambda^{👌}_{q0}(t_0)$

Initial state annotated

- $\forall t.\ \lambda^{👌}_{q}(t) \wedge (q‘,v) \in \delta(q,o(t))$

  $\rightarrow \lambda^{👌}_{q‘}(\tau_v(t)) \wedge\ \lambda^{\#}_{q‘}(\tau_v(t)) \geq_q (\lambda^{\#}_{q}‘(t))$

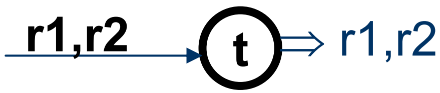
  where: $\geq_q$ is $>$ iff $q \in F$, $\geq_q$ is $\geq$ otherwise

Valid Annotation

Single process example constraints

Input preserving

1. $\forall\, t \in T.$
   - $r_1(\tau_{r_1,r_2}(t)) \wedge r_1(\tau_{r_1,\overline{r_2}}(t)) \wedge r_2(\tau_{r_1,r_2}(t)) \wedge r_2(\tau_{\overline{r_1},r_2}(t))$
   - $\neg r_1(\tau_{\overline{r_1},\overline{r_2}}(t)) \wedge \neg r_1(\tau_{\overline{r_1},r_2}(t)) \wedge \neg r_2(\tau_{\overline{r_1},\overline{r_2}}(t)) \wedge \neg r_2(\tau_{r_1,\overline{r_2}}(t))$

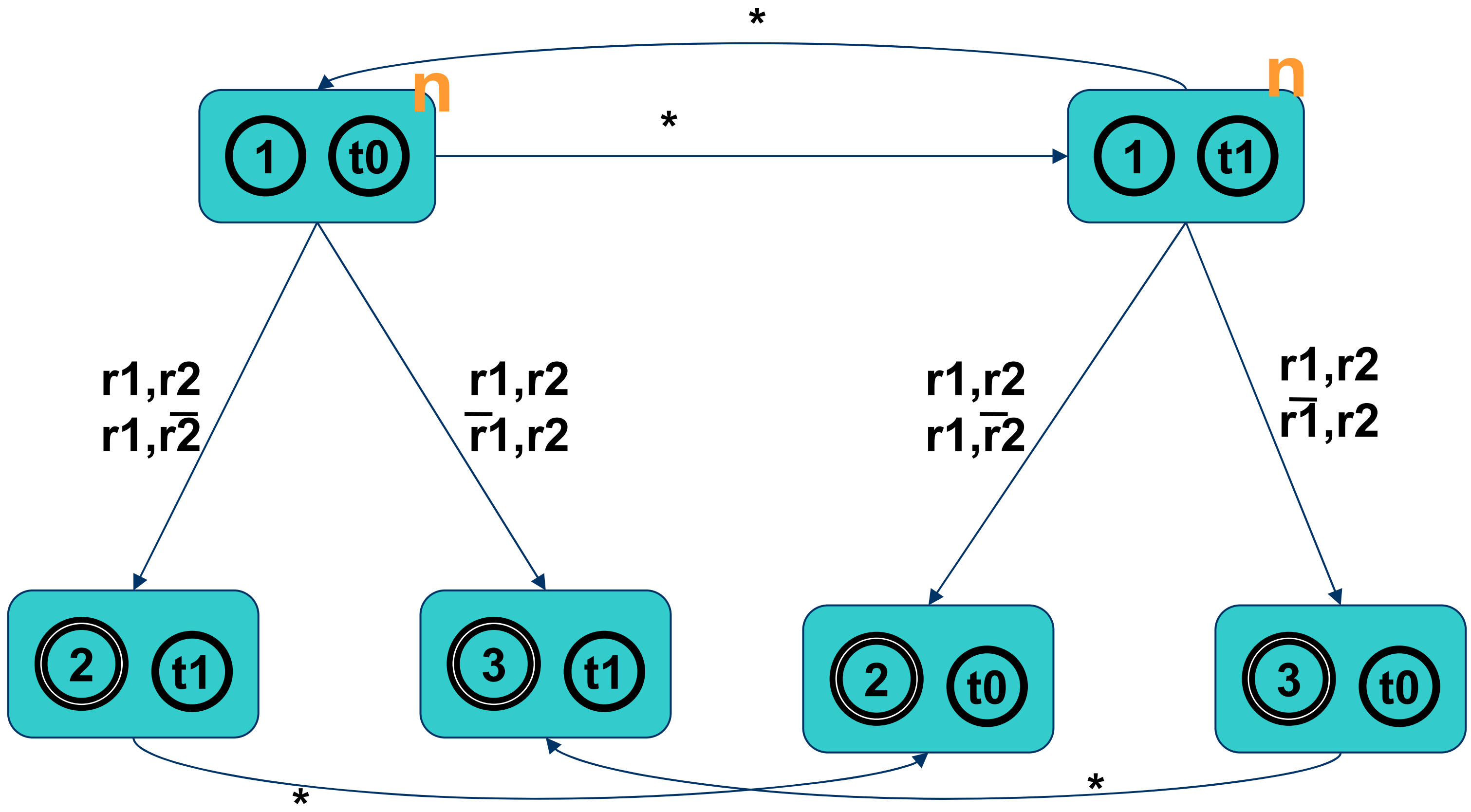


Single process example constraints

2. $\lambda^{👌}{}_1(t_0), \neg r_1(t_0), \neg r_2(t_0)$

3. $\forall\ t \in T.$
   - $\lambda^{👌}{}_1(\tau_{\overline{r_1},\overline{r_2}}(t)), \lambda^{\#}{}_1(\tau_{\overline{r_1},\overline{r_2}}(t)) \geq \lambda^{\#}{}_1(t)$
   - $\lambda^{👌}{}_1(\tau_{r_1,r_2}(t)), \lambda^{\#}{}_1(\tau_{r_1,r_2}(t)) \geq \lambda^{\#}{}_1(t)$
   - $\lambda^{👌}{}_1(\tau_{\overline{r_1},r_2}(t)), \lambda^{\#}{}_1(\tau_{\overline{r_1},r_2}(t)) \geq \lambda^{\#}{}_1(t)$
   - $\lambda^{👌}{}_1(\tau_{r_1,\overline{r_2}}(t)), \lambda^{\#}{}_1(\tau_{r_1,\overline{r_2}}(t)) \geq \lambda^{\#}{}_1(t)$

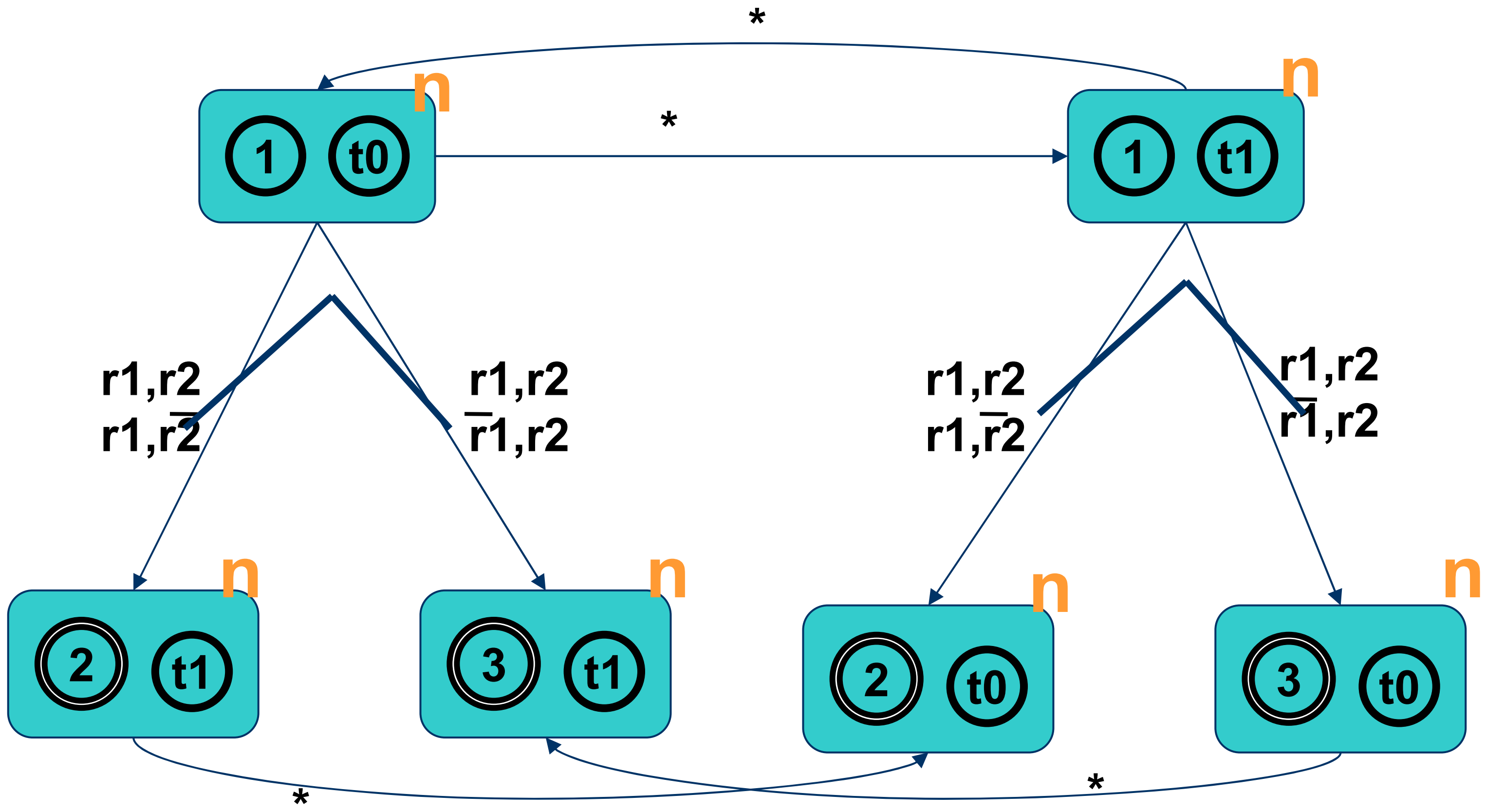
Single process example constraints



Single process example constraints

4. $\forall t.\ \lambda^{\text{👌}}_1(t) \rightarrow \neg g_1(t) \vee \neg g_2(t)$
5. $\forall\ t \in T.\ \lambda^{\text{👌}}_1(t) \wedge r_1(t) \rightarrow$
   - $\lambda^{\text{👌}}_2(\tau_{\overline{r_1},\overline{r_2}}(t)),\ \lambda^{\#}_2(\tau_{\overline{r_1},\overline{r_2}}(t)) > \lambda^{\#}_1(t)$
   - $\lambda^{\text{👌}}_2(\tau_{\overline{r_1},r_2}(t)),\ \lambda^{\#}_2(\tau_{\overline{r_1},r_2}(t)) > \lambda^{\#}_1(t)$
   - $\lambda^{\text{👌}}_2(\tau_{r_1,\overline{r_2}}(t)),\ \lambda^{\#}_2(\tau_{r_1,\overline{r_2}}(t)) > \lambda^{\#}_1(t)$
   - $\lambda^{\text{👌}}_2(\tau_{r_1,r_2}(t)),\ \lambda^{\#}_2(\tau_{r_1,r_2}(t)) > \lambda^{\#}_1(t)$
6. … analogous for r_2 and states (2,t)

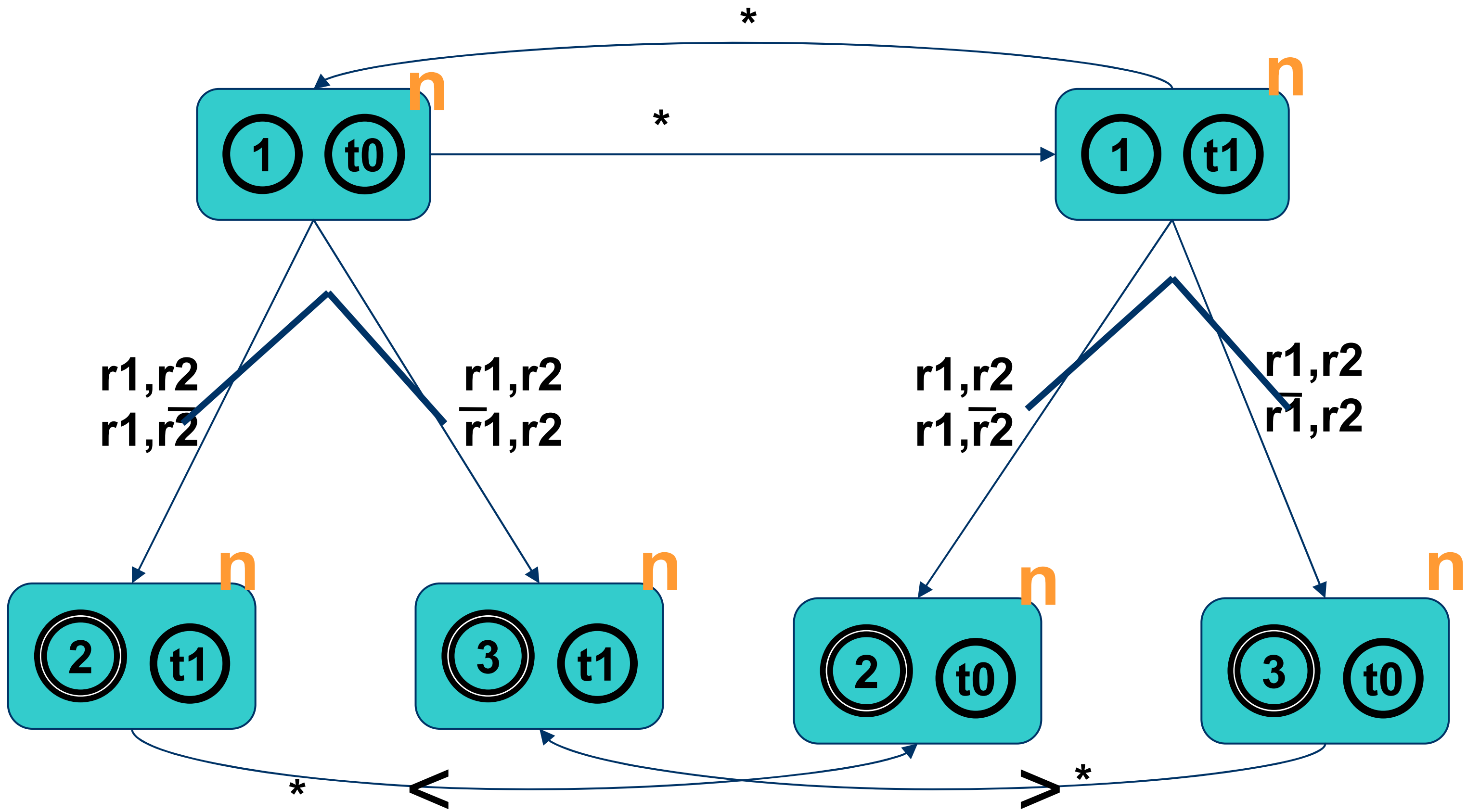
Single process example constraints



Single process example constraints

7. $\forall\ t \in T.\ \lambda^{👌}{}_2(t) \wedge \neg g_1(t) \rightarrow$
   - $\lambda^{👌}{}_2(\tau_{\bar{r}_1,\bar{r}_2}(t)),\ \lambda^{\#}{}_2\,(\tau_{\bar{r}_1,\bar{r}_2}(t)) > \lambda^{\#}{}_2(t)$
   - $\lambda^{👌}{}_2\,(\tau_{\bar{r}_1,r_2}(t)),\ \lambda^{\#}{}_2\,(\tau_{\bar{r}_1,r_2}(t)) > \lambda^{\#}{}_2\,(t)$
   - $\lambda^{👌}{}_2\,(\tau_{r_1,\bar{r}_2}(t)),\ \lambda^{\#}{}_2\,(\tau_{r_1,\bar{r}_2}(t)) > \lambda^{\#}{}_2\,(t)$
   - $\lambda^{👌}{}_2\,(\tau_{r_1,r_2}(t)),\ \lambda^{\#}{}_2\,(\tau_{r_1,r_2}(t)) > \lambda^{\#}{}_2\,(t)$
8. analogous for $\lambda^{👌}{}_3(t) \wedge \neg g_2(t)$

Single process example constraints



Do we know which bound to choose?

- **Not known how large bound** need to be iff some specification realizable **in general**
- But, **bound can be estimated for fully informed architectures** (each process has complete knowledge of the environment)

Constraints for distributed system

- Find family of transition systems $\{\mathrm{I}_p = (T_p, t_{0p}, \tau_p, o_p) \mid p \in P\}$ such that their composition $\mathrm{I} = (T, t_0, \tau, o)$ satisfies φ given by $B = (\Sigma, \Upsilon, Q, q_0, \delta, F)$
- I_p has to act equally on states it cannot distinguish
- Its output may only depend on its own state

Constraints for distributed system

- Let d_p map states $t \in T$ into T_p
- Let p_α refer to the process outputing α

- $\forall$ v,v' $\subseteq O_{env}$ where $v \cap I_p = v' \cap I_p$.

  $d_p(\tau_v(t)) = d_p(\tau_{v'}(t))$
- $\forall v \subseteq O_{env} \cap I_p$. $\forall$ t,u $\in T$.

  $d_p(t) = d_p(u) \wedge_{\alpha \in I_p \setminus O_{env}} (\alpha(d_{p_\alpha}(t)) \leftrightarrow \alpha(d_{p_\alpha}(u)))$
  $\rightarrow d_p(\tau_v(t)) = d_p(\tau_v(u))$

Distributed example constraints

- 1-3, 4-5 stay the same

4. $\forall t.\ \lambda^{\text{👌}}{}_1(t) \rightarrow \neg g_1(d_1(t)) \vee \neg g_2(d_2(t))$
7. $\forall\ t \in T.\ \lambda^{\text{👌}}{}_2(t) \wedge \neg g_1(d_1(t)) \rightarrow$
   - $\lambda^{\text{👌}}{}_2(\tau_{\bar{r}_1,\bar{r}_2}(t)),\ \lambda^{\#}{}_2\,(\tau_{\bar{r}_1,\bar{r}_2}) > \lambda^{\#}{}_2(t)$
   - $\lambda^{\text{👌}}{}_2\,(\tau_{\bar{r}_1,r_2}(t)),\ \lambda^{\#}{}_2\,(\tau_{\bar{r}_1,r_2}) > \lambda^{\#}{}_2\,(t)$
   - $\lambda^{\text{👌}}{}_2\,(\tau_{r_1,\bar{r}_2}(t)),\ \lambda^{\#}{}_2\,(\tau_{r_1,\bar{r}_2}) > \lambda^{\#}{}_2\,(t)$
   - $\lambda^{\text{👌}}{}_2\,(\tau_{r_1,r_2}(t)),\ \lambda^{\#}{}_2\,(\tau_{r_1,r_2}) > \lambda^{\#}{}_2\,(t)$
8. … analogous for r_2

Distributed example constraints

9. $\forall\ t \in T.$
   - $d_1(\tau_{r_1,r_2}(t))=d_1(\tau_{r_1,\overline{r_2}}(t))$
   - $d_1(\tau_{\overline{r_1},r_2}(t))=d_1(\tau_{\overline{r_1},\overline{r_2}}(t))$
   - $d_2(\tau_{r_1,r_2}(t))=d_2(\tau_{\overline{r_1},r_2}(t))$
   - $d_2(\tau_{r_1,\overline{r_2}}(t))=d_2(\tau_{\overline{r_1},\overline{r_2}}(t))$
10. $\forall\ t, u \in T.\ d_1(u) \wedge (g_1(d_1(t)) \leftrightarrow g_1(d_1(u))) \rightarrow$
   - $d_1(\tau_{r_1,r_2}(t))=d_1(\tau_{r_1,r_2}(u))$
   - $d_1(\tau_{\overline{r_1},r_2}(t))=d_1(\tau_{\overline{r_1},r_2}(u))$
11. analogous for g_2

Experiments and results

- Several experiments with SMT solver Yices on a 2.6 Ghz Opteron system
- The simple arbiter specification can be solved in 7-8 seconds (if 8+ states allowed)
- Usually it takes much longer to show unsatisfiability than to compute a solution if there is one
- Good guessing of the needed states can significantly increase performance

Conclusions

- Constraint system gives us a comparatively quick synthesis by ignoring unnecessary large solutions in the search space
- Real world restrictions can be taken into account
- We can tackle undecidable problems by approaching them iteratively
- Performance may be increased by good guessing of the minimal bound(s)

Questions?

Annotation theorem proof

- Consider run graph G on $\mathcal{I}$:
  - Case G not accepting: there is a lasso with rejecting state (q,t) in the loop (so $q \in F$).

    Assume some valid annotation λ exists.

    Then for the successor(s) (q',t') of (q,t) holds:
    - $\lambda(q,t) < \lambda(q',t')$,
    - along the loop it holds ‚≤'
    - After one „round" for some descendant (q'',t'') of (q',t') it holds $\lambda(q'',t'') < \lambda(q,t)$

    so $\lambda(q,t) < \lambda(q,t)$ while the image of λ is $\mathbb{N}$!!

Annotation theorem proof

– Case G accepting: no lasso with rejecting state. A $(|T| \bullet |F|)$-bounded annotation given by assigning to each vertex $(q,t) \in V$ the highest number of rejecting states occuring on some path to it, while assigning '$\_$' to all (q',t') not in G

Run graph

- A run graph of a co-Büchi automaton $B = (\Sigma, \Upsilon, Q, q_0, \delta, F)$ on a Σ-labeled Υ-transition $\mathrm{I} = (T, t_0, \tau, o)$ is a minimal directed graph $G=(V,E)$ such that
  - $V \subseteq Q \times T$
  - $(q_0, t_0) \in V$
  - $\forall (q,t) \in V$: $\{(q',v) \in Q \times \Upsilon | ((q,t),(q', \tau(t,v))) \in E\}$ satisfies $\delta(q,o(t))$