



Fundamental Algorithms for System Modeling, Analysis, and Optimization



Stavros Tripakis

UC Berkeley
EECS 244
Fall 2016

Controller and Program Synthesis

From verification to synthesis

Verification:

first write program (or model of a system), then specify formal properties, then check correctness.

Synthesis:

first specify formal properties, then let synthesizer automatically generate a correct program.

Put another way:

from imperative (how) to declarative (what) design;
“raising the level of abstraction”.

EECS 144/244, UC Berkeley: 2

What is synthesis?

Roughly:

$$\exists P: \forall x: \varphi(x, P(x))$$

Many different variants, depending on what is P, φ , and how search is done.

Very old topic (Church, 1960s) recently rejuvenated.

EECS 144/244, UC Berkeley: 3

Program synthesis and proofs

From 2nd order formula

$$\exists P: \forall x: \varphi(x, P(x))$$

to 1st order formula

$$\forall x: \exists y: \varphi(x, y)$$

Synthesizing program P can be done by proving **constructively** that the above formula is valid.

Deductive program synthesis.

EECS 144/244, UC Berkeley: 4

Dimensions in Synthesis (Gulwani)

Concept Language (Application)

- Programs
 - Straight-line programs
- Automata
- Queries
- Sequences

Also: logic synthesis

User Intent (Ambiguity)

- Logic, Natural Language
- Examples, Demonstrations/Traces

Search Technique (Algorithm)

- SAT/SMT solvers (Formal Methods)
- A*-style goal-directed search (AI)
- Version space algebras (Machine Learning)

PPDP 2010: "Dimensions in Program Synthesis", Gulwani.

EECS 144/244, UC Berkeley: 5

Compilers vs. Synthesizers (Gulwani)

Dimension	Compilers	Synthesizers
Concept Language	Executable Program	Variety of concepts: Program, Automata, Query, Sequence
User Intent	Structured language	Variety/mixed form of constraints: logic, examples, traces
Search Technique	Syntax-directed translation (No new algorithmic insights)	Uses some kind of search (Discovers new algorithmic insights)

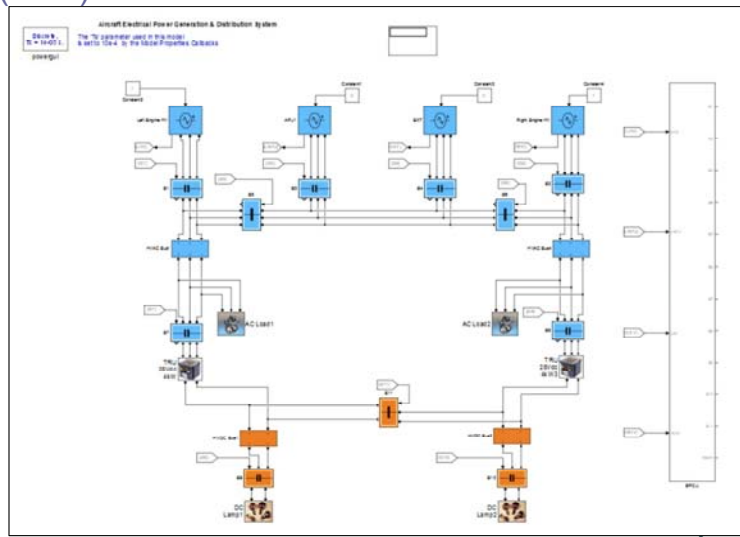
EECS 144/244, UC Berkeley: 6

MOTIVATING EXAMPLE

EECS 144/244, UC Berkeley: 7

Designing controllers can be tricky and time consuming

Example: Electrical Power Generation and Distribution System (EPS) of a modern aircraft



Thanks to:
Pierluigi Nuzzo
Antonio Iannopolo

Designing controllers can be tricky and time consuming

Example: EPS requirements (in English)

- Assumptions:
- A2) At least one power source is always “healthy” (i.e. it is operational and can be inserted into the network to deliver power);
 - A3) Failures can only affect the power sources; once a power source becomes “unhealthy” (i.e. it is not operational and cannot be inserted into the network to deliver power), it will never return to be “healthy” (e.g., turned back on) during the cruising phase of the mission;
 - A4) An AC bus is correctly powered if the root-mean-square (RMS) voltage at its loads is between 110 V and 120 V and the frequency is 400 Hz.

Under the above assumptions, the BPCU offers the following guarantees:

- Guarantees:
- G1) At start-up all the power source contactors are “open”;
 - G2) In normal conditions (i.e. no faults or failures in the system) G_L and G_R are “on” and provide power for the left side and the right side of the system, respectively; auxiliary power units are “off”; C_9 and C_{10} are open (“off”);
 - G3) No AC bus is powered by more than one power source at the same time, i.e. AC power sources can never be paralleled;
 - G4) It never happens that both the APUs are inserted into the network at the same time;
 - G5) AC buses cannot be unpowered for more than a well-defined length of time;
 - G6) DC buses must always stay powered, at least in a “reduced performance” mode, which occurs when only one HVRU is used;
 - G7) The left AC bus B_1 must always be powered from the first available source from the ordered list (G_L, A_L, A_R, G_R);
 - G8) The right AC bus B_2 must always be powered from the first available source from the ordered list (G_R, A_R, A_L, G_L).

Designing controllers can be tricky and time consuming

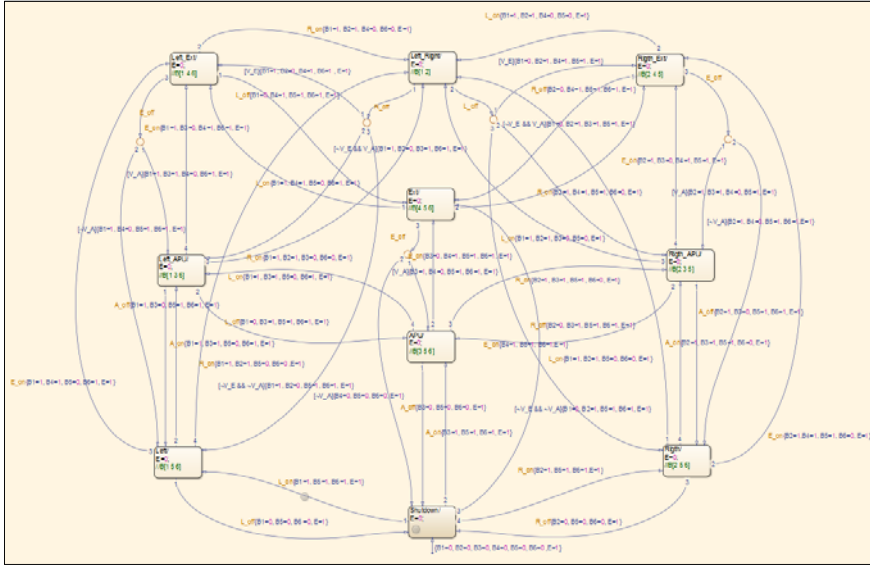
Example: EPS requirements (in English) – zooming in

- A2) At least one power source is always “healthy” (i.e. it is operational and can be inserted into the network to deliver power);

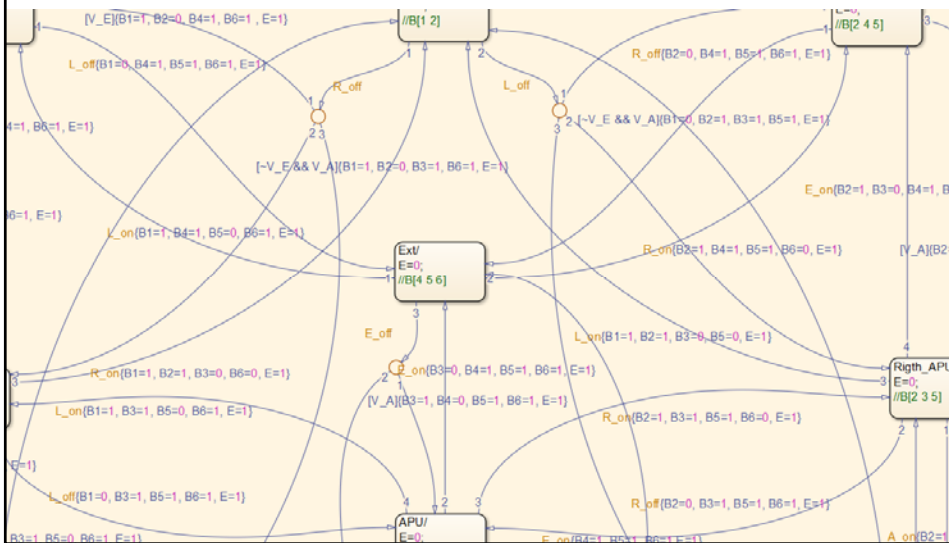
G1) At start-up all the power source contactors are “open”;

- G3) No AC bus is powered by more than one power source at the same time, i.e. AC power sources can never be paralleled;

Designing controllers can be tricky and time consuming
 Example: EPS “hand-written” controller



Designing controllers can be tricky and time consuming
 Example: EPS “hand-written” controller – zooming in



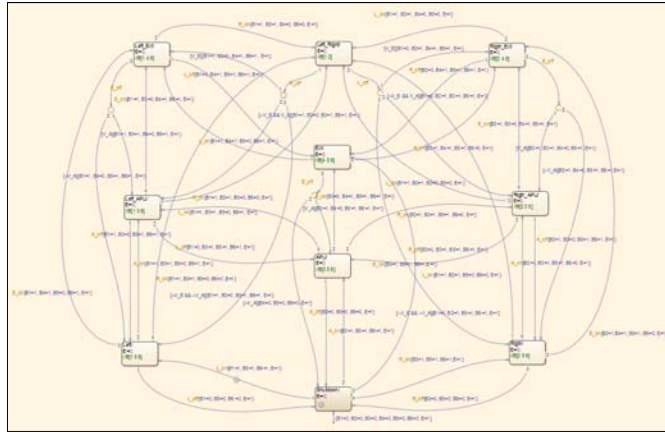
Designing controllers can be tricky and time consuming

Example: EPS “hand-written” controller

Design time ~ 1 week [Nuzzo] (but have to verify also)

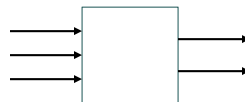
For a real controller, it could be months [e.g., robotic controllers, Willow Garage]

Can design time be improved?



Declarative specification of controllers

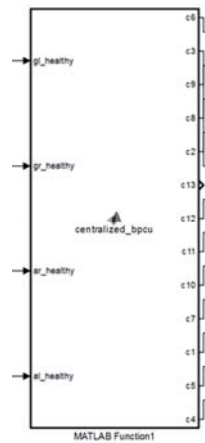
At the outset the controller is just a box with inputs and outputs:



Declarative specification of controllers

At the outset the controller is just a box with inputs and outputs:

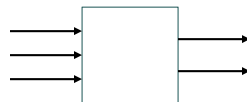
Example: EPS controller



EECS 144/244, UC Berkeley: 15

Declarative specification of controllers

At the outset the controller is just a box with inputs and outputs:



We can specify the input-output behavior of the controller in a high-level language, e.g., in **temporal logic**.

EECS 144/244, UC Berkeley: 16

Declarative specification of controllers

Example: LTL specification for EPS

~40 lines

```
#Assumptions
(gl_healthy & gr_healthy & al_healthy & ar_healthy)
[](gl_healthy | gr_healthy | al_healthy | ar_healthy)
[]!(gl_healthy -> X(gl_healthy))
[]!(gr_healthy -> X(gr_healthy))
[]!(al_healthy -> X(al_healthy))
[]!(ar_healthy -> X(ar_healthy))

#Guarantees
(!c1 & !c2 & !c3 & !c4 & !c5 & !c6 & !c7 & !c8 & !c9 & !c10 & !c11 & !c12 & !c13)
[]X(c7 & X(c8) & X(c11) & X(c12) & X(c13))

[]!(c2 & c3)
[]!(c1 & c5 & (al_healthy | ar_healthy))
[]!(c4 & c6 & (al_healthy | ar_healthy))
[]!(X(gl_healthy) & X(gr_healthy)) -> X(c2) & X(c3) & X(c9) & X(c10)
[]!(X(gl_healthy) & X(gr_healthy)) -> X(c9) & X(c10)

[]X(gl_healthy) -> X(c1)
[]X(gr_healthy) -> X(c4)
[]X(al_healthy) -> X(c2)
[]X(ar_healthy) -> X(c3)

[]X(gl_healthy) -> X(c1)
[]X(gr_healthy) -> X(c4)
...
```

```
#Guarantees
...
[]!(gl_healthy -> X(c5))
[]!(gr_healthy -> X(c6))

[]!(X(gl_healthy) & X(gr_healthy)) -> (X(c5) & X(c6))

[]!(X(gl_healthy) & X(al_healthy) & X(gr_healthy)) -> (X(c2) & X(c3))

[]!(X(gl_healthy) & X(gr_healthy) & X(al_healthy) & !c3 & !c2) -> X(c2)

[]!(X(al_healthy) & c2) -> X(c2)
[]!(X(ar_healthy) & c3) -> X(c3)

[]!(X(gl_healthy) & X(al_healthy) & X(ar_healthy) & !c2) -> X(c3)

[]!(X(gr_healthy) & X(ar_healthy) & X(al_healthy) & !c3) -> X(c2)

[]!(gl_healthy & !al_healthy & !ar_healthy) -> X(c6)

[]!(gr_healthy & !ar_healthy & !al_healthy) -> X(c5)
...
```

EECS 144/244, UC Berkeley: 17

Declarative specification of controllers

Example: LTL specification for EPS

Close mapping from English to LTL:

A2) At least one power source is always “healthy” (i.e. it is operational and can be inserted into the network to deliver power);



```
[](gl_healthy | gr_healthy | al_healthy | ar_healthy)
```

EECS 144/244, UC Berkeley: 18

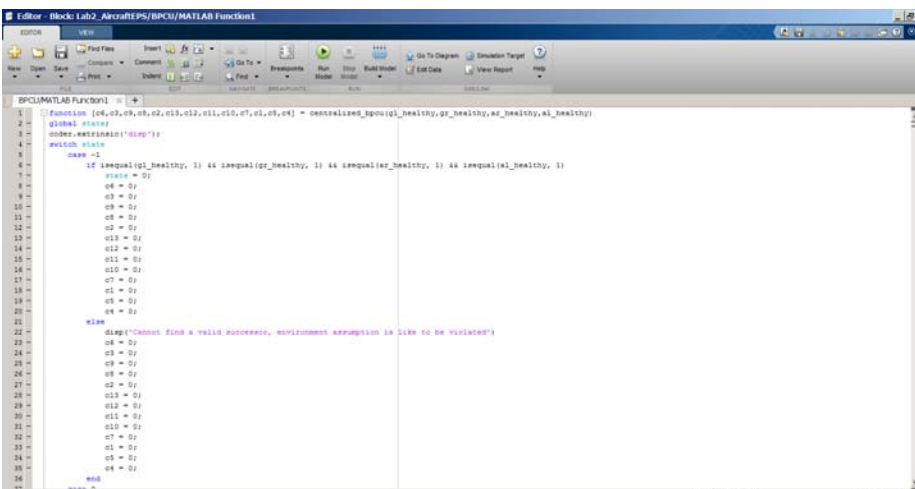
The controller synthesis problem

Given formula **specification** (e.g., in LTL) synthesize **controller** (e.g., FSM) which implements the specification (or state that such a controller does not exist).

EECS 144/244, UC Berkeley: 19

Automatic controller synthesis from declarative specifications

Example: controller for EPS synthesized from previous LTL spec using Tulip (Caltech) ~3k lines of Matlab



```
Editor - Block: Lab2_AbrakeEPS/BPCU/MATLAB Functions
function [q0,q1,q2,q3,q4,q5,q6,q7,q8,q9,q10,q11,q12,q13,q14,q15,q16,q17,q18,q19,q20] = decentralized_tpuu(q1_healthy,q2_healthy,q3_healthy,q4_healthy,q5_healthy);
global states;
coder.extrinsic('diag');
switch states
case 0
if isequal(q1_healthy, 1) && isequal(q2_healthy, 1) && isequal(q3_healthy, 1) && isequal(q4_healthy, 1)
q0 = 0;
q1 = 0;
q2 = 0;
q3 = 0;
q4 = 0;
q5 = 0;
q6 = 0;
q7 = 0;
q8 = 0;
q9 = 0;
q10 = 0;
q11 = 0;
q12 = 0;
q13 = 0;
q14 = 0;
q15 = 0;
q16 = 0;
q17 = 0;
q18 = 0;
q19 = 0;
q20 = 0;
else
diag('Cannot find a valid successor, environment assumption is likely to be violated');
q0 = 0;
q1 = 0;
q2 = 0;
q3 = 0;
q4 = 0;
q5 = 0;
q6 = 0;
q7 = 0;
q8 = 0;
q9 = 0;
q10 = 0;
q11 = 0;
q12 = 0;
q13 = 0;
q14 = 0;
q15 = 0;
q16 = 0;
q17 = 0;
q18 = 0;
q19 = 0;
q20 = 0;
end
end
```

Automatic controller synthesis from declarative specifications

Example: controller for EPS synthesized using Tulip (Caltech), ~40 states – zooming in

```
switch state
case -1
  if isequal(gl_healthy, 1) && isequal(gr_healthy, 1) && isequal(ar_healthy, 1) && isequal(al_healthy, 1)
    state = 0;
    c6 = 0;
    c3 = 0;
    c9 = 0;
    c8 = 0;
    c2 = 0;
    c13 = 0;
    c12 = 0;
    c11 = 0;
    c10 = 0;
    c7 = 0;
    c1 = 0;
    c5 = 0;
    c4 = 0;
  else
    disp('Cannot find a valid successor, environment assumption is like to be violated')
    c6 = 0;
    c3 = 0;
    c9 = 0;
    c8 = 0;
    c2 = 0;
```

Synthesis in these two lectures

Part 1: Controller synthesis and game solving.

Part 2: Example-guided and syntax-guided synthesis.

CONTROLLER SYNTHESIS

EECS 144/244, UC Berkeley: 23

Declarative specification of controllers

At the outset the controller is just a box with inputs and outputs:



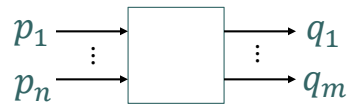
We can specify the input-output behavior of the controller in a high-level language, e.g., in **temporal logic**.

EECS 144/244, UC Berkeley: 24

Controller synthesis (reactive synthesis)

[Pnueli-Rosner, POPL 1989]

Given **interface** of controller:



and given **temporal logic formula** φ over set of input/output variables,

synthesize **a controller (= state machine) M**, such that **all** behaviors of M (for any sequence of inputs) satisfy φ .

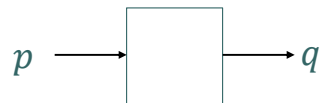
Note: other notions of controller synthesis exist in the literature.

See "Bridging the gap" paper on the course web site for details.

EECS 144/244, UC Berkeley: 25

Examples

Consider controller interface:



and specifications

$$\varphi_1 = G(p \rightarrow Xq)$$

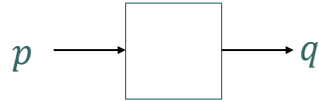
$$\varphi_2 = G(p \leftrightarrow Xq)$$

$$\varphi_3 = G(q \leftrightarrow Xp)$$

EECS 144/244, UC Berkeley: 26

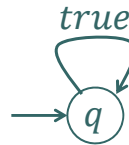
Examples

Consider controller interface:

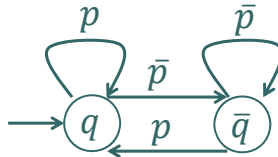


and specifications

$$\varphi_1 = G(p \rightarrow Xq)$$



$$\varphi_2 = G(p \leftrightarrow Xq)$$



$$\varphi_3 = G(q \leftrightarrow Xp)$$

No solution: controller cannot foresee the future!

EECS 144/244, UC Berkeley: 27

Satisfiability vs. realizability

Satisfiability: exists some behavior that satisfies the specification. (In this behavior, we may choose both inputs and outputs as we wish.)

Realizability: exists controller that implements the specification. Must work for all input sequences, since inputs are uncontrollable.

Inherently different problems, also w.r.t. complexity:

LTL satisfiability: PSPACE

LTL realizability: 2EXPTIME

EECS 144/244, UC Berkeley: 28

Controller synthesis algorithms: computing strategies in games

Solving safety games

Solving reachability games

Solving deterministic Büchi games (liveness)

Remarks on the general LTL synthesis problem

EECS 144/244, UC Berkeley: 29

Controller synthesis algorithms

Solving safety games

Solving reachability games

Solving deterministic Büchi games (liveness)

Remarks on the general LTL synthesis problem

EECS 144/244, UC Berkeley: 30

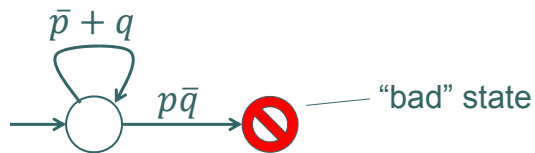
Safety automata

In some fortunate cases, the LTL specification can be translated to a **safety** automaton.

Example: $\varphi = G(p \rightarrow q)$



Automaton:



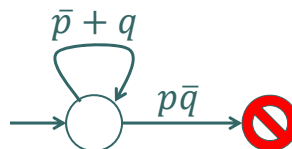
EECS 144/244, UC Berkeley: 31

"Spreading" a safety automaton to a game

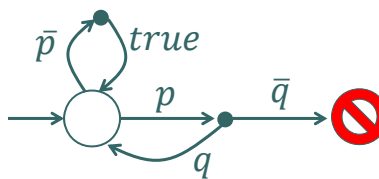
[Ehlers PhD thesis, 2013]

We need to separate the input moves from the output moves:

Automaton:



Game:



EECS 144/244, UC Berkeley: 32

Safety games

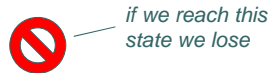
Input (environment) states:



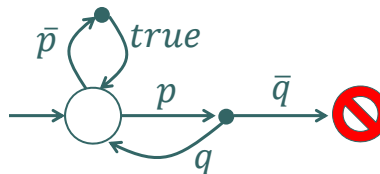
Output (controller) states:



Bad state:



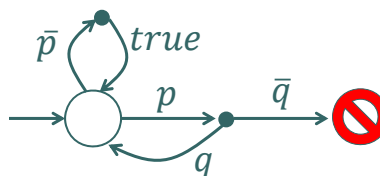
Goal: find **winning strategy** = avoiding bad state



EECS 144/244, UC Berkeley: 33

Solving safety games

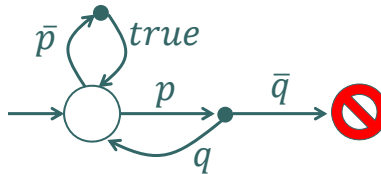
1. Compute set of losing states, starting with $Losing := \{\text{red circle with slash}\}$;
2. If initial state in $Losing$, no strategy exists.
3. Otherwise, all remaining states are winning. Extract strategy from them by choosing outputs that avoid the losing states.



EECS 144/244, UC Berkeley: 34

Solving safety games

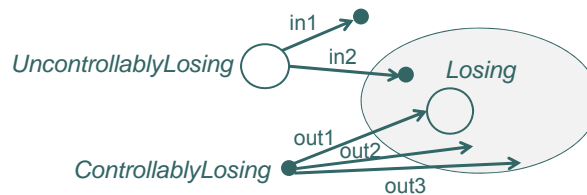
1. Compute set of losing states, starting with $Losing := \{\ominus\}$;
 - repeat
 - $UncontrollablyLosing := \{s \mid s \text{ has uncontrollable succ in } Losing\}$;
 - $ControllablyLosing := \{s \mid \text{all controllable succs of } s \text{ are in } Losing\}$;
 - $Losing := Losing \cup UncontrollablyLosing \cup ControllablyLosing$;
 - until $Losing$ does not change;



EECS 144/244, UC Berkeley: 35

Solving safety games

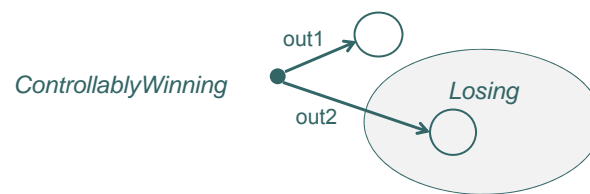
1. Compute set of losing states, starting with $Losing := \{\ominus\}$;
 - repeat
 - $UncontrollablyLosing := \{s \mid s \text{ has uncontrollable succ in } Losing\}$;
 - $ControllablyLosing := \{s \mid \text{all controllable succs of } s \text{ are in } Losing\}$;
 - $Losing := Losing \cup UncontrollablyLosing \cup ControllablyLosing$;
 - until $Losing$ does not change;



EECS 144/244, UC Berkeley: 36

Solving safety games

- Extracting the strategy: “cut” controllable transitions in order to avoid losing states.
- Strategy is **state-based** (also called “positional”, or “memoryless”).



EECS 144/244, UC Berkeley: 37

Controller synthesis algorithms

Solving safety games

Solving reachability games

Solving deterministic Büchi games (liveness)

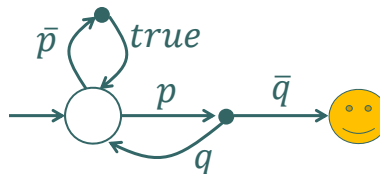
Remarks on the general LTL synthesis problem

EECS 144/244, UC Berkeley: 38

Reachability games: dual of safety games

Reachability game: trying to reach a target state.

Observation: what is *Losing* for the safety player is *Winning* for the reachability player (and vice versa).

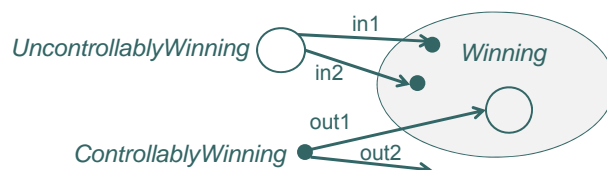


EECS 144/244, UC Berkeley: 39

Solving reachability games: direct algorithm

1. Compute set of *Winning* states;

- $Winning := \{\text{😊}\};$
- repeat
 - $Winning := Winning \cup ForceNext(Winning);$
- until *Winning* does not change;
 - $ForceNext(S) := \{s \mid \text{all uncontrollable succs of } s \text{ are in } S\} \cup \{s \mid s \text{ has controllable succ in } S\}$

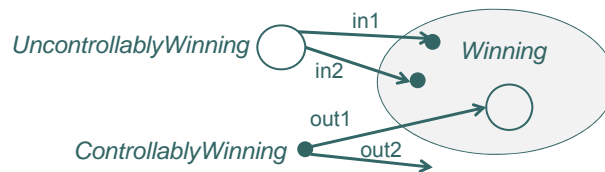


EECS 144/244, UC Berkeley: 40

How to extract strategies in reachability games?

Similarly as for safety games:

Extract strategy from **ForceNext(S)**: ensure you choose the right controllable transition that leads in winning state.



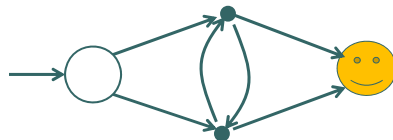
Is strategy state-based?

Yes!

EECS 144/244, UC Berkeley: 41

How to extract strategies in reachability games?

Similarly as for safety games: BUT, a subtlety:



Need to fix successor the first time state is added in *Winning*.

EECS 144/244, UC Berkeley: 42

Controller synthesis algorithms

Solving safety games

Solving reachability games

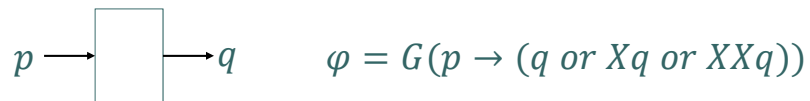
Beyond safety and reachability games

Remarks on the general LTL synthesis problem

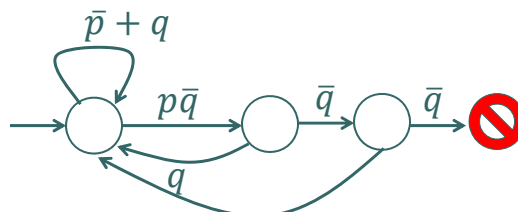
EECS 144/244, UC Berkeley: 43

What about other types of properties?

Bounded response specifications can be translated to safety automata/games:



Automaton:



EECS 144/244, UC Berkeley: 44

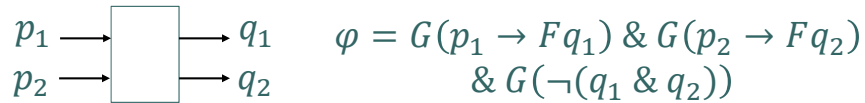
What about liveness properties?

What about **unbounded response**?



$$\varphi = G(p \rightarrow Fq)$$

More interesting example:



$$\varphi = G(p_1 \rightarrow Fq_1) \ \& \ G(p_2 \rightarrow Fq_2) \\ \& \ G(\neg(q_1 \ \& \ q_2))$$

EECS 144/244, UC Berkeley: 45

Synthesis for general LTL specifications

Given LTL specification φ :

If φ can be translated to a **deterministic Büchi automaton**, then can extend the previous ideas to solving Büchi games.

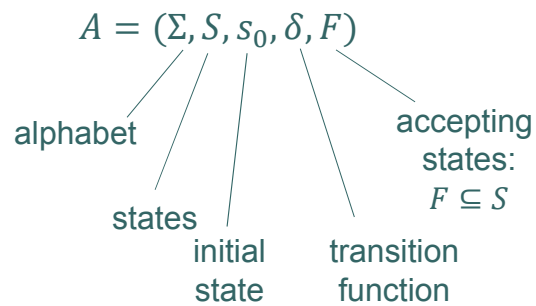
Otherwise, solution involves more advanced topics, such as tree automata. Will not be covered in this course.

Note: LTL **cannot** always be translated to deterministic Büchi automata.

EECS 144/244, UC Berkeley: 46

Büchi automata

Syntactically same as finite state automata:



But Büchi automata accept **infinite words**.

A run must visit an accepting state infinitely often.

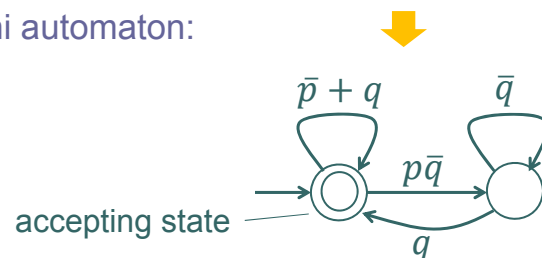
EECS 144/244, UC Berkeley: 47

From LTL to Büchi automata

Consider unbounded response property:

$$\varphi = G(p \rightarrow Fq)$$

Büchi automaton:



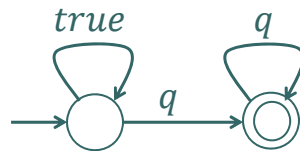
EECS 144/244, UC Berkeley: 48

From LTL to Büchi automata

Consider LTL formula:

$$\varphi = FGq$$

Büchi automaton? (s.t. **there exists** an accepting run)



Is there a deterministic Büchi automaton for this spec?

No!

EECS 144/244, UC Berkeley: 49

Controller synthesis algorithms

Solving safety games

Solving reachability games

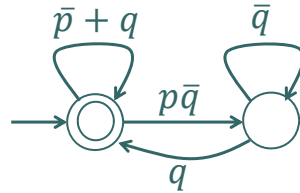
Solving deterministic Büchi games (liveness)

Remarks on the general LTL synthesis problem

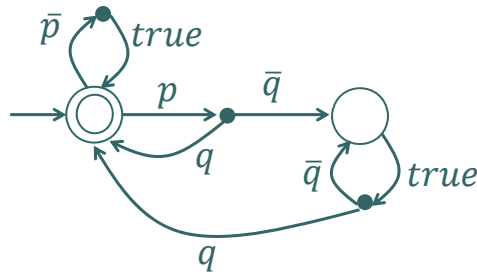
EECS 144/244, UC Berkeley: 50

Spreading Büchi automata to Büchi games

Büchi automaton:



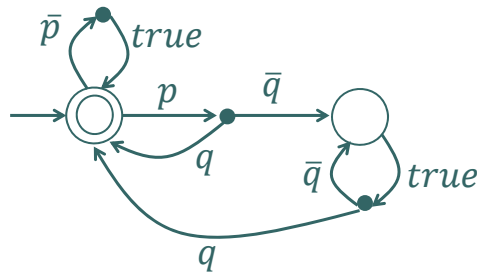
Büchi game:



EECS 144/244, UC Berkeley: 51

Solving deterministic Büchi games

1. Compute set of **RecurrentAccepting** states = accepting states from which controller can force returning to an accepting state infinitely often.
2. Solve reachability game with target = RecurrentAccepting.



EECS 144/244, UC Berkeley: 52

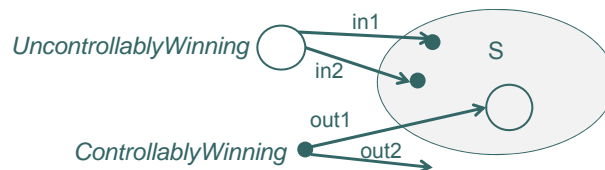
Solving deterministic Büchi games

1. Compute set of **RecurrentAccepting** states = accepting states from which controller can force returning to an accepting state infinitely often.
 - $RecAcc :=$ set of all accepting states;
 - repeat
 - $Revisit := \{ \}$;
 - repeat
 - $Revisit := Revisit \cup \mathbf{ForceNext}(Revisit \cup RecAcc)$;
 - until $Revisit$ does not change;
 - $RecAcc := RecAcc \cap Revisit$;
 - until set $RecAcc$ does not change;

EECS 144/244, UC Berkeley: 53

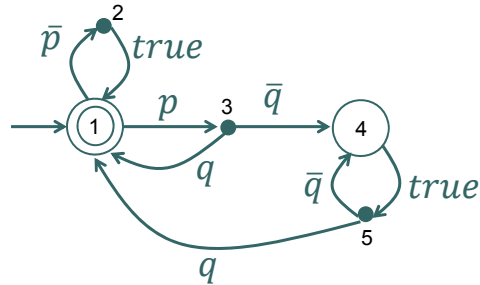
Recall

$\mathbf{ForceNext}(S) := \{ s \mid \text{all uncontrollable succs of } s \text{ are in } S \}$
 $\cup \{ s \mid s \text{ has controllable succ in } S \}$



EECS 144/244, UC Berkeley: 54

Solving deterministic Büchi games – Example



- $RecAcc :=$ set of all accepting states;
- repeat
 - $Revisit := \{ \}$;
 - repeat
 - $Revisit := Revisit \cup \mathbf{ForceNext}(Revisit \cup RecAcc)$;
 - until $Revisit$ does not change;
 - $RecAcc := RecAcc \cap Revisit$;
- until set $RecAcc$ does not change;

EECS 144/244, UC Berkeley: 55

Computing recurrent accepting states: a subtle relation with reachability games

1. Compute set of **RecurrentAccepting** states = accepting states from which controller can force returning to an accepting state infinitely often.

- $RecAcc :=$ set of all accepting states;
- repeat
 - $Revisit := \{ \}$;
 - repeat
 - $Revisit := Revisit \cup \mathbf{ForceNext}(Revisit \cup RecAcc)$;
 - until $Revisit$ does not change;
 - $RecAcc := RecAcc \cap Revisit$;
- until set $RecAcc$ does not change;

(almost) a reachability game iteration

EECS 144/244, UC Berkeley: 56

Solving reachability games vs. computing Revisit

1. Compute set of *Winning* states:

- $Winning := \{\text{😊}\};$
- repeat
 - $Winning := Winning \cup \mathbf{ForceNext}(Winning);$
- until *Winning* does not change;

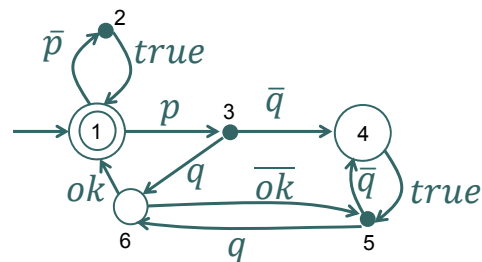
What is the difference?
Does it matter?

2. Compute *Revisit*:

- $Revisit := \{ \};$
- repeat
 - $Revisit := Revisit \cup \mathbf{ForceNext}(Revisit \cup RecAcc);$
- until *Revisit* does not change;

EECS 144/244, UC Berkeley: 57

Solving deterministic Büchi games – modified example



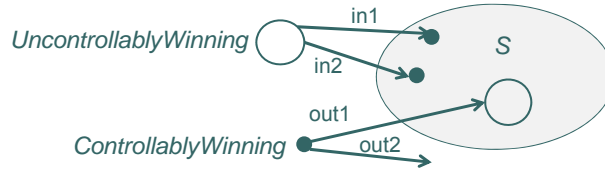
- $RecAcc :=$ set of all accepting states;
- repeat
 - $Revisit := \{ \};$
 - repeat
 - $Revisit := Revisit \cup \mathbf{ForceNext}(Revisit \cup RecAcc);$
 - until *Revisit* does not change;
 - $RecAcc := RecAcc \cap Revisit;$
- until set *RecAcc* does not change;

EECS 144/244, UC Berkeley: 58

How to extract strategies in deterministic Büchi games?

Similarly as for reachability games:

Extract strategy from **ForceNext(S)**: ensure you choose the right controllable transition that leads in winning state.



Careful to choose the transition the first time state is added to S.

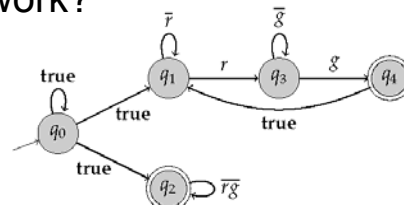
Is strategy state-based?

Yes!

EECS 144/244, UC Berkeley: 59

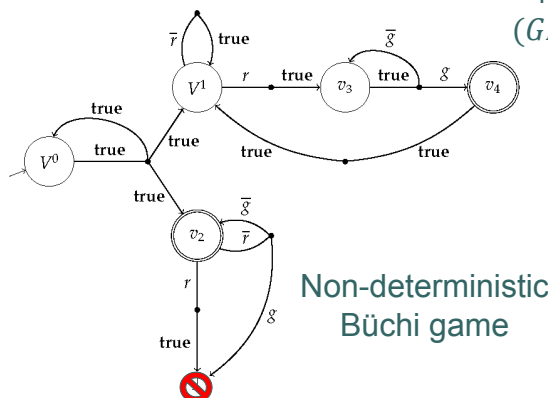
What about non-deterministic Büchi games? Does same algorithm work?

Not quite:
algorithm sound
but incomplete.



Non-deterministic automaton for
 $(GFr \wedge GFg) \vee (FG\bar{r} \wedge FG\bar{g})$

r : input
 g : output



Non-deterministic
Büchi game

[Ruediger Ehlers,
PhD thesis, 2013]

EECS 144/244, UC Berkeley: 60

Controller synthesis algorithms

Solving safety games

Solving reachability games

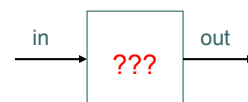
Solving deterministic Büchi games (liveness)

Remarks on the general LTL synthesis problem

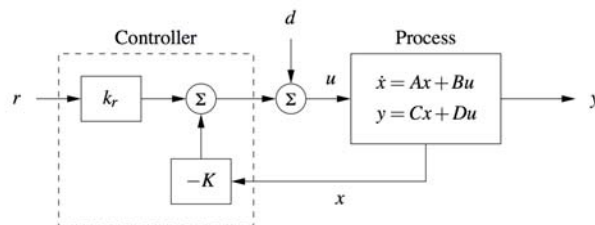
EECS 144/244, UC Berkeley: 61

Controller synthesis: EE vs. CS ?

CS: synthesize outputs to implement φ :



EE: synthesize inputs to stabilize a physical process/plant:

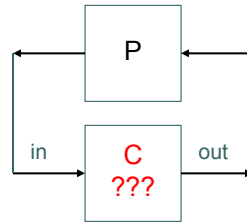


Not different: plant inputs = controller outputs (and vice versa).

EECS 144/244, UC Berkeley: 62

Can we capture plants in the CS synthesis problem?

CS: given plant P (say, a FSM), synthesize controller C , so that closed-loop system satisfies φ :



Can we reduce this problem to the standard LTL synthesis problem?

EECS 144/244, UC Berkeley: 63

Remarks, assessment

Despite some (mostly isolated) success stories, controller synthesis hasn't really caught on yet in practice.

Why is that?

- Normal: things like that take time (c.f. model-checking)
- 2EXPTIME is a horrible (worst-case) complexity (remember: even linear is too expensive because of state explosion!)
- Tools still impractical
- Synthesis of real, complex systems from complete specs impractical (imagine full synthesis of complete Intel microchip from LTL specs ...)
- Lack of good debugging (e.g., counter-examples)
- Need: better tools, better methods (incremental, interactive, ...)
- **Great opportunities for research!**

EECS 144/244, UC Berkeley: 64

References

1. Pnueli, A., Rosner R., *On the Synthesis of a Reactive Module*, POPL 1989.
2. Ehlers, R., *Symmetric and Efficient Synthesis*, PhD thesis, 2013.
3. Jobstmann, B., *Reachability and Buchi Games*, slides available online, 2010.
4. Ehlers et al, *Bridging the Gap between Supervisory Control and Reactive Synthesis: Case of Full Observation and Centralized Control*, WODES 2014.
5. Wang et al, *The Theory of Deadlock Avoidance via Discrete Control*, POPL 2009. (“Success story” of supervisor synthesis applied to deadlock removal in concurrent software.)

EECS 144/244, UC Berkeley: 65

PROGRAM SYNTHESIS

EECS 144/244, UC Berkeley: 66

The “modern” approach to program synthesis

- **Interactive:**
 - computer-aided programming
 - programmer solves key problems (e.g., provides **program skeleton**), synthesizer fills in (boring or tedious) details (e.g., **missing guards/assignments**)
- **Search-for-patterns based:**
 - synthesis = search among set of user-defined patterns
- **Solver based:**
 - **heavily uses verifiers like SAT and SMT solvers**
 - often in a counter-example guided loop

EECS 144/244, UC Berkeley: 67

Example: programming by sketching [Solar-Lezama, Bodik, et al.]

Parallel Parking by Sketching

Ref: Chaudhuri, Solar-Lezama (PLDI 2010)

```
Err = 0.0;
for(t = 0; t < T; t += dT){
  if(stage == STRAIGHT){
    if(t > ??) stage = INTURN;
  }
  if(stage == INTURN){
    car.ang = car.ang - ??;
    if(t > ??) stage = OUTTURN;
  }
  if(stage == OUTTURN){
    car.ang = car.ang + ??;
    if(t > ??) break;
  }
  simulate_car(car);
  Err += check_collision(car);
}
Err += check_destination(car);
```

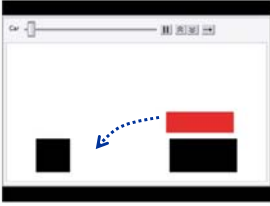
When to start turning?

Backup straight

How much to turn?

Turn

Straighten



Enables programmers to focus on high-level solution strategy

38 y: 68

Using SAT and SMT solvers for synthesis

Recall: what is synthesis?

$$\exists P: \forall x: \varphi(x, P(x))$$

Usually re-written as:

$$\exists P: \forall x: pre(x) \rightarrow post(x, P(x))$$

i.e., if input satisfies precondition, then output will satisfy postcondition.

EECS 144/244, UC Berkeley: 69

Using SAT and SMT solvers for synthesis

$$\exists P: \forall x: pre(x) \rightarrow post(x, P(x))$$

Example of pre(), post():

$$pre(x1, x2): number(x1) \wedge number(x2)$$

$$post(x1, x2, y): x1 \leq y \wedge x2 \leq y \wedge (x1 = y \vee x2 = y)$$

i.e., the spec for max(x1,x2).

EECS 144/244, UC Berkeley: 70

First: using SAT and SMT solvers for verification

Suppose we already have a program P.

Then instead of checking whether P is **correct**

$$\forall x: pre(x) \rightarrow post(x, P(x))$$

we can check whether P is **wrong**

$$\exists x: pre(x) \wedge \neg post(x, P(x))$$

i.e., we can check **satisfiability** of the formula

$$pre(x) \wedge \neg post(x, P(x))$$

EECS 144/244, UC Berkeley: 71

Hold on: are programs formulas?

Consider a simple loop-free program:

```
function P(int x) returns (real y)
{
  int tmp := 0;
  if (x >= 0) then {
    tmp++;
    y := tmp*x;
  }
  else
    y := -x;
  return y;
}
```

Formula:

$$P(x, y) = (x \geq 0 \wedge y = x) \vee (x < 0 \wedge y = -x)$$

EECS 144/244, UC Berkeley: 72

Hold on: are programs formulas?

What about real programs?

Loops, data structures, libraries, pointers, threads, ...

Translation to formulas much harder, but verification tools are available that do this, constantly making progress.

We will assume we have a formula $P(x,y)$ representing the program P: "*y is the output of P for input x*".

EECS 144/244, UC Berkeley: 73

Back to using SAT and SMT solvers for verification

We can check **satisfiability** of the formula

$$pre(x) \wedge \neg post(x, P(x))$$

or, writing P as predicate on both input and output variables:

$$pre(x) \wedge P(x, y) \wedge \neg post(x, y)$$

Satisfiable => P is wrong: we get a **counter-example** (x,y)

Unsatisfiable => P is correct (for all x)

EECS 144/244, UC Berkeley: 74

Using SAT and SMT solvers for synthesis

What can be done when we don't have the program P ?

$$pre(x) \wedge P(x, y) \wedge \neg post(x, y)$$

Hint: what if we have a finite/small number of candidate programs?

Iterate and search!

EECS 144/244, UC Berkeley: 75

Programs with “holes”

Almost-complete programs:

```
Err = 0.0;
for(t = 0; t < T; t += dT){
  if(stage == STRAIGHT){
    if(t > ??) stage = INTURN;
  }
  if(stage == INTURN){
    car.ang = car.ang - ??;
    if(t > ??) stage = OUTTURN;
  }
  if(stage == OUTTURN){
    car.ang = car.ang + ??;
    if(t > ??) break;
  }
  simulate_car(car);
  Err += check_collision(car);
}
Err += check_destination(car);
```

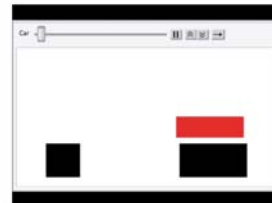
When to start turning?

Backup straight

How much to turn?

Turn

Straighten



EECS 144/244, UC Berkeley: 76

Programs with “holes”

What should we replace “??” with?

Patterns:

integer constants

linear expressions of the form $ax + by + c$ where x, y are variables in the program

...

Even with these restrictions, **infinite set of candidates** ...

Search may take a long time or never terminate.

Can we do better?

EECS 144/244, UC Berkeley: 77

Asking the solver to find the program

Suppose our program has 1 hole, to be filled with an integer variable.

Then, the formula characterizing the program becomes

$$P(h, x, y)$$

Can we use the solver to find the right h ?

Check satisfiability of

Free variable: solver
must find right value

$$\forall x, y: pre(x) \wedge P(h, x, y) \rightarrow post(x, y)$$

EECS 144/244, UC Berkeley: 78

Problem: universal quantification ...

$$\forall x, y: pre(x) \wedge P(h, x, y) \rightarrow post(x, y)$$

Today's solvers check satisfiability of quantifier-free formulas (mostly).

What can we do about that?

Hint: what if we have a finite number of **positive examples**? i.e., I/O pairs (x, y) satisfying $pre(x) \wedge post(x, y)$.

EECS 144/244, UC Berkeley: 79

Example-guided synthesis

Suppose we have a finite number of positive examples, say 2: $(x_1, y_1), (x_2, y_2)$.

That is: we know that these hold:

$$pre(x_1), pre(x_2), post(x_1, y_1), post(x_2, y_2)$$

So it suffices to check satisfiability of

$$P(h, x_1, y_1) \wedge P(h, x_2, y_2)$$

EECS 144/244, UC Berkeley: 80

Example-guided synthesis

In general, for n positive examples and k hole variables:

$$\bigwedge_{i=1}^n P(h_1, h_2, \dots, h_k, x_i, y_i)$$

We turned universal quantification into finite conjunction!

EECS 144/244, UC Berkeley: 81

Example-guided synthesis

What if solver finds this formula unsatisfiable ?

$$\bigwedge_{i=1}^n P(h_1, h_2, \dots, h_k, x_i, y_i)$$

Unsatisfiable => no program exists!

This is **sound**: if no program exists that works even in this finite set of examples, we cannot hope to find a program that works for all examples.

EECS 144/244, UC Berkeley: 82

Example-guided synthesis

What if solver finds this formula satisfiable ?

$$\bigwedge_{i=1}^n P(h_1, h_2, \dots, h_k, x_i, y_i)$$

Satisfiable $\Rightarrow P(h_1, h_2, \dots, h_k)$ is only a candidate.

It still needs to be verified for **all** I/O pairs.

We can again use the solver for that!

EECS 144/244, UC Berkeley: 83

Example-guided synthesis

$$\bigwedge_{i=1}^n P(h_1, h_2, \dots, h_k, x_i, y_i)$$

Satisfiable $\Rightarrow P(h_1, h_2, \dots, h_k)$ is only a candidate.

Verify it by checking satisfiability of

$$pre(x) \wedge \underbrace{P(h_1, h_2, \dots, h_k, x, y)}_{\text{These are now fixed}} \wedge \neg post(x, y)$$

These are now fixed

If formula is unsatisfiable then we are done!

What if formula is satisfiable?

Our candidate is wrong. We get a counter-example: (x^*, y^*)

What then?

EECS 144/244, UC Berkeley: 84

Adding negative examples to the synthesizer's inputs

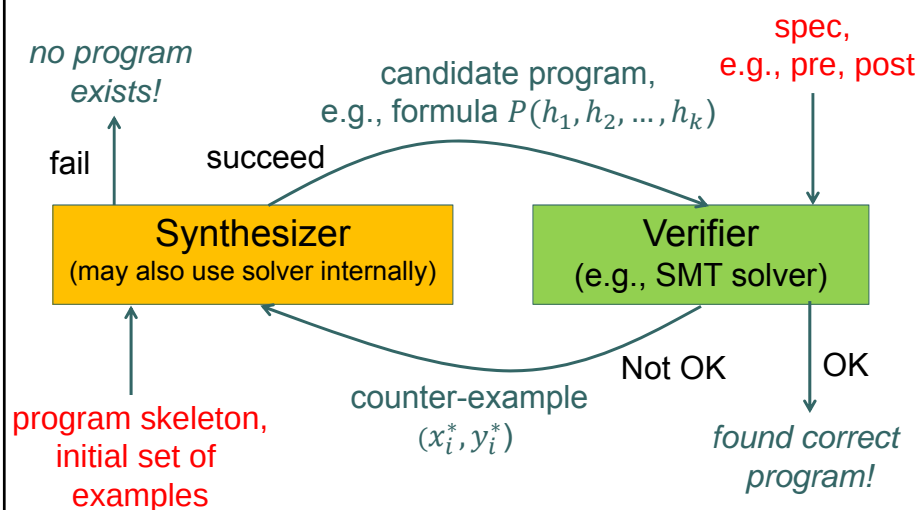
In general, for n positive examples, m negative examples, and k hole variables:

$$\bigwedge_{i=1}^n P(h_1, h_2, \dots, h_k, x_i, y_i) \wedge \bigwedge_{i=1}^m \neg P(h_1, h_2, \dots, h_k, x_i^*, y_i^*)$$

Alternative: the user could provide the correct output for the counter-example input, or we could use a reference (correct and deterministic) program.

EECS 144/244, UC Berkeley: 85

Counter-example guided synthesis



EECS 144/244, UC Berkeley: 86

References

1. Solar-Lezama. *Program sketching*. STTT Vol 15, Issue 5-6, Oct 2013.
2. Alur, Bodik, et al. *Syntax-Guided Synthesis*. FMCAD 2013.
3. International Journal on Software Tools for Technology Transfer, Special Issue on Synthesis, Volume 15, Issue 5-6, October 2013.
4. Course by Ras Bodik and Emina Torlak. CS294 – *Program Synthesis for Everyone*. <http://www.cs.berkeley.edu/~bodik/cs294fa12>