PROGRAM SYNTHESIS

RUZICA PISKAC
YALE UNIVERSITY

Oregon Programming Languages
Summer School 2023
ACKNOWLEDGMENTS

• Based liberal theft of ideas and reuse of slides from Rajeev Alur, Roderick Bloem, Krishnendu Chatterjee, Ruediger Ehlers, Bernd Finkbeiner, Priyanka Golia, Andreas Griesmayer, Tom Henzinger, Georg Hofferek, Swen Jacobs, Barbara Jobstmann, Ayrat Khalimov, Bettina Koenighofer, Robert Koenighofer, Andreas Morgenstern, Nir Piterman, Amir Pnueli, Yaniv Sa’ar, Swen Schewe, Klaus Schneider, Armando Solar-Lezama, Stefan Staber, Emina Torlak, Moshe Vardi, and many others

• Synthesizing good synthesis slides is a group effort
CONSTRUCT CORRECT SYSTEMS AUTOMATICALLY FROM SPEC

Don’t do same thing twice!

Diagram:
- Specification
- Implementation
- Requirements
- synthesis
- verification
CONSTRUCT CORRECT SYSTEMS AUTOMATICALLY FROM SPEC

Don’t do same thing twice!

Specifying is easier than implementing!
Removes need to Code!

Correct by Construction!
THIS TUTORIAL

• Reactive synthesis – from Church’s synthesis problem to scalable software
• Deductive synthesis – from the seminar Manna-Waldinger paper to scalable software
• Can reactive and deductive synthesis be friends?
• Syntax-guided synthesis
• New applications of software synthesis
REACTIVE SYNTHESIS
• Reactive systems: embedded systems, GUIs, robots, hardware circuits, ...

• Church synthesis problem (1957):
  • Given a requirement $\varphi$ on the input-output behavior of a Boolean circuit, compute a circuit $C$ that satisfies $\varphi$.

• Reactive synthesis: given a specification written in LTL (linear temporal logic), automatically compute the program that satisfies the specification.
TODAY’S LECTURE

Finite State Reactive systems
  • Continuous interaction with environment
  • Do not terminate
  • Discrete time
  • Correctness statements are temporal (temporal logic, automata)

Tomorrow’s lecture: functions
  • Start with input, terminate with output (non-termination = bug)
  • Correctness is input/output relation (Hoare logic)
SYNTHESIS

Given
Input and output signals
Specification $\phi$ of behavior

Determine
Realizability: Is there a system that realizes specification?
Synthesis: If system exists, construct it

For any input trace $I$, we have $I || S(I) \models \phi$
## LINEAR TEMPORAL LOGIC (LTL)

<table>
<thead>
<tr>
<th>LTL</th>
<th>PSL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p )</td>
<td>( p )</td>
<td>( p )</td>
</tr>
<tr>
<td>( X \varphi )</td>
<td>next ( \varphi )</td>
<td>( \varphi )</td>
</tr>
<tr>
<td>( G \varphi )</td>
<td>always ( \varphi )</td>
<td>( \varphi ) ( \varphi ) ( \varphi ) ( \varphi ) ( \varphi )</td>
</tr>
<tr>
<td>( F \varphi )</td>
<td>eventually! ( \varphi )</td>
<td>( \varphi ) ( \varphi ) ( \varphi ) ( \varphi )</td>
</tr>
<tr>
<td>( \varphi \ U \psi )</td>
<td>( \varphi ) until! ( \psi )</td>
<td>( \varphi ) ( \varphi ) ( \varphi ) ( \psi )</td>
</tr>
</tbody>
</table>

plus Boolean connectors (\( \lor \), \( \land \), \( \neg \), \( \rightarrow \)) and nesting
LTL SYNTAX

• If $\varphi$ is an atomic propositional formula, it is a formula in LTL

• If $\varphi$ and $\psi$ are LTL formulas, so are $\varphi \land \psi$, $\varphi \lor \psi$, $\neg \varphi$, $\varphi \mathcal{U} \psi$ (until), $X \varphi$ (next), $F \varphi$ (eventually), $G \varphi$ (always)

• Interpretation: over computations $\pi: \omega \to 2^V$ which assigns truth values to the elements of $V$ at each time instant

  $\pi \models X \varphi$ iff $\pi^1 \models \varphi$

  $\pi \models G \varphi$ iff $\forall i \cdot \pi^i \models \varphi$

  $\pi \models F \varphi$ iff $\exists i \cdot \pi^i \models \varphi$

  $\pi \models \varphi \mathcal{U} \psi$ iff $\exists i \cdot \pi^i \models \psi \land \forall j \cdot 0 \leq j < i \Rightarrow \pi^j \not\models \varphi$

Here, $\pi^i$ is the $i^{th}$ state on a path
EXPRESSING PROPERTIES IN LTL

- Good for safety \((G \neg)\) and liveness \((F)\) properties

- Express:
  - When a request occurs, it will eventually be acknowledged
    - \(G (\text{request} \Rightarrow F \text{acknowledge})\)
  - A path contains infinitely many \(q\)’s
    - \(G F q\)
  - At most a finite number of states in a path satisfy \(\neg q\) (or property \(q\) eventually stabilizes)
    - \(F G q\)
  - Action \(s\) precedes \(p\) after \(q\)
    - \([\neg q U (q \land [\neg p U s])]\)
    - Note: hard to do correctly.
SATISFIABILITY & REALIZABILITY

Satisfiability:

Is there a trace that satisfies spec?

Realizability:

Is there a system that satisfies spec?
SATISFIABILITY & REALIZABILITY

Satisfiability: Is there a trace that satisfies the spec?

Realizability: Is there a system that satisfies the spec?

input req1, req2

output grant1, grant2

G( (req1 \rightarrow \text{grant1}) \land (req2 \rightarrow \text{grant2}) )

G \neg( grant1 \land grant2 )

Satisfiable?

Yes

Realizable?

No

Inputs universally quantified
SATISFIABILITY & REALIZABILITY

Satisfiability: Is there a trace that satisfies the spec?
Realizability: Is there a system that satisfies the spec?

Realizability $\neq$ Satisfiability

**input** req1

**output** grant1

$G(\; grant1 \leftrightarrow X \; req1 )$

Satisfiable?

**Yes** (No matter how we set grant1).

Realizable?

**No**, clairvoyant!
FORMAL VERIFICATION

Given:
System provides outputs
A specification

One Player: (not a game!)
• Environment provides inputs

System is good if it fulfills the spec for all possible inputs
SYNTHESIS IS A GAME

Given:

- **System provides outputs**
- A specification

**Two Players** (a game!)
- Environment provides inputs
- **System provides outputs**

System is good if it fulfills the spec **for all possible inputs**
REACTIVE SYNTHESIS SETTINGS

Reactive Systems
• Constant interaction
• No Termination
• E.g. Cell phones, Operating Systems, Powerpoint

Finite State
• Non-terminating, finite systems are graphs with loops
• Not our current focus: functions
  • “Create a function that computes sqrt(2)”
EXAMPLE I: CHESS

- Environment determines black moves
- System determines white moves
- Winning condition:
  - If all black moves are legal, then all white moves are legal and eventually, white reaches checkmate

Easy to specify!
**CHECKERS AND SYSTEMS**

**Checkers** are passive
Judge if given behavior is OK
Used in verification

**Systems** are active
Construct correct behavior
Result of synthesis

**Property Synthesis: systems not checkers**

Checks moves

Thinks of moves
SYNTHESIS

1. Specify
2. Create Game
3. Solve Game
4. Create System
EXAMPLE II: ARBITER

Input: r0, r1

Output: g0, g1

What is the specification?
EXAMPLE II: ARBITER

Input: r0, r1
Output: g0, g1

\[ G(r_0 \rightarrow Fg_0) \]
\[ G(r_1 \rightarrow Fg_1) \]
\[ G(\neg g_0 \lor \neg g_1) \]
ARBITER SPECIFICATION

Deterministic Büchi automaton for

\( G(r \rightarrow Fg) \)

Accepting states must be visited infinitely often

1. Specify
2. Create Game
3. Solve Game
4. Create System

Don’t get stuck here!
ARBITER GAME

Game for

G(r → Fg)

Box: environment
Circle: system

Accepting states must be visited infinitely often
ARBITER GAME

Game for
\[ G(r \rightarrow Fg) \]

Box: environment
Circle: system

Accepting states must be visited infinitely often

1. Specify
2. Create Game
3. Solve Game
4. Create System
ARBITER STRATEGY

Game for
\( G(r \rightarrow Fg) \)

1. Specify
2. Create Game
3. **Solve Game**
4. Create System

Box: environment
Circle: system
ARBITER STRATEGY

Game for
\[ G(r \rightarrow \text{Fg}) \]

Box: environment
Circle: system

1. Specify
2. Create Game
3. **Solve Game**
4. Create System
ARBITER STRATEGY

Game for
G(r \rightarrow Fg)

initial state = q0

while()
{
    r = getinput();
    if(state==q0 && r==0) {g=0; state=q0}
    if(state==q0 && r==1) {g=0; state=q1}
    if(state==q1) {g=1; state=q0}
    if(state==q1) {g=1; state=q0}
}