Amir’s Research: Temporal Logic Synthesis

Roderick Bloem
Graz University of Technology

Roderick Bloem

Temporal Logic Synthesis

Gate Level

always @ (posedge clock) begin
  case (State)
    IF: begin
      NPC = PC + 12'd2;
      IR = instruction;
    end // case: IF
    ID: begin
      A = Registers[adFld1];
      B = Registers[adFld2];
    end // case: ID
    EX: begin
      if (memRef) begin
        ALUOutput = A + Imm;
      end // if (memRef)
      else if (regRegALU) begin
        if (funFld == ADD)
          ALUOutput = A + B;
        else if (funFld == SUB)
          ALUOutput = A - B;
        else if (funFld == AND)
          ALUOutput = A & B;
        else if (funFld == OR)
          ALUOutput = A | B;
        end
      end // case: EX
    MEM: begin
      if (memRef) begin
        if (opcode == LW)
          LMD = datain;
      end // if (memRef)
      if (branch) begin
        if (Cond)
          PC = ALUOutput[11:0];
        else
          PC = NPC;
      end else begin
        PC = NPC;
      end // else: !if(branch)
    end // case: MEM
    WB: begin
      if (regRegALU) begin
        if (adFld3 != 2'd0)
          Registers[adFld3] = ALUOutput;
      end else if (regImmALU) begin
        if (adFld2 != 2'd0)
          Registers[adFld2] = ALUOutput;
      end // case: WB
  end // case (State)
SystemC

SC_MODULE(producer) {
    public:
        sc_port<write_if> out;

    SCCTOR(producer) {
        SC_THREAD(main);
    }

    void main() {
        char c;
        while (true) {
            out.write(c);
            if(...) out.reset();
        }
    }
};

SC_MODULE(consumer) {
    public:
        sc_port<read_if> in;

    SCCTOR(consumer) {
        SC_THREAD(main);
    }

    void main() {
        char c;
        while (true) {
            in.read(c);
            cout<<in.num_available();
        }
    }
};
Temporal Logic

in req1, req2;
out grant1, grant2;

always( req1 \rightarrow \text{eventually! grant1} );
always( req2 \rightarrow \text{eventually! grant2} );
never( grant1 \land \text{grant2} );

describe WHAT system does, not HOW it does it
Construct Correct Systems Automatically from Spec

Don’t do the same thing twice! Use synthesis!
Satisfiability & Realizability

Two problems:

1. **Satisfiability**: Is there a trace that satisfies the spec?

2. **Realizability**: Is there a system that satisfies the spec?

Pre-Pnueli attempt: Synthesis through satisfiability
Satisfiability & Realizability

Satisfiability: Is there a trace that satisfies the spec?

Realizability: Is there a system that satisfies the spec?

Realizability \neq Satisfiability

always( (req1 \rightarrow grant1) \land (req2 \rightarrow grant2))

never ( grant1 \land grant2 )

Satisfiable? Realizable?
Satisfiable, but functionally impossible!
Distinguish inputs from outputs!
Satisfiability & Realizability

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

Realizability $\neq$ Satisfiability

always( grant1 $\leftrightarrow$ next req1 )

Satisfiable? Realizable?
Satisfiable, but not realizable: clairvoyant!
Take time into account!
Checkers and Systems

Checkers are passive
Judge if given behavior is allowed (satisfiability)
Used in verification

Systems are active
Construct correct behavior (realizability)
Result of synthesis

Synthesis: Convert checkers to systems
Temporal Logic Synthesis is a Game

**Chess**
- Two players: Black & White
- Alternate moves
- Rules of the game
- Goal: chess mate

**Synthesis**
- Two players: environment & system
- Alternate moves
- Specification
- Specification

**Synthesis:** Construct a perfect player
A Game

Every request is granted within a tick

\[ \text{always}(r \rightarrow g \lor \text{next}! g) \]

Winning region + Strategy
Playing Requires Lookahead

- **r1** is granted within a tick
- **r2** is granted immediately
- no simultaneous grants.
  (assuming no two **r2s** in a row)

\[ D[v] \quad D[w] \]

\[ r1 \quad g1 \quad g2 \quad r2 \]

Strategy = System
Turning Temporal Logic into Systems

- Church '62
- Büchi & Landweber '69
- Rabin '69
- Pnueli & Rosner '89
- Pnueli '77
- Vardi & Wolper '86

Annus mirabilis of temporal logic synthesis
and then nothing?
LTL Synthesis

LTL Formula → Büchi automaton → Rabin game → Strategy = System

<table>
<thead>
<tr>
<th>formula</th>
<th>system</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4.10^9</td>
</tr>
<tr>
<td>2</td>
<td>1.10^20</td>
</tr>
<tr>
<td>3</td>
<td>3.10^38</td>
</tr>
<tr>
<td>4</td>
<td>1.10^77</td>
</tr>
<tr>
<td>5</td>
<td>1.10^154</td>
</tr>
<tr>
<td>6</td>
<td>???</td>
</tr>
<tr>
<td>7</td>
<td>4.10^9</td>
</tr>
<tr>
<td>8</td>
<td>1.10^20</td>
</tr>
<tr>
<td>9</td>
<td>3.10^38</td>
</tr>
<tr>
<td>10</td>
<td>1.10^77</td>
</tr>
<tr>
<td>11</td>
<td>???</td>
</tr>
</tbody>
</table>

Amir & Roni almost killed temporal logic synthesis

nanoseconds since start of universe
Turning Temporal Logic into Systems

Sweeping things under rug:
KupfermanVardi05…
GR(1) Synthesis

<table>
<thead>
<tr>
<th>LTL Formula</th>
<th>deterministic Büchi automaton</th>
<th>GR(1) game</th>
<th>Strategy = System</th>
</tr>
</thead>
</table>

**Benefit**
- No exponential blowup

**Drawback**
- We lose expressibility
- harder to express properties?

Practical applicability?
AMBA Bus

- Industrial standard
- ARM’s AMBA AHB bus
  - High performance on-chip bus
  - Data, address, and control signals (pipelined)
  - Arbiter part of bus (determines control signals)
  - Up to 16 masters and 16 clients
AMBA Bus

- Master initiates transfer. It has the signals
  - HBUSREQi - Master i wants the bus
  - HLOCKi - Master i wants an uninterruptible access
  - HBURST - This access has length 1/4/incr
  - address & data lines

- The arbiter decides access
  - HGRANTi - Next transfer for master i
  - HMASTER[..] - Currently active master
  - HMASTLOCK - Current access is uninterruptible

- The clients synchronize the transfer
  - HREADY - Ready for next transfer

- Sequence for master
  - Ask; wait for grant; wait for hready; state transfer type & start transfer
AMBA Arbiter

Specification: 12 Guarantees, 3 Assumptions.

Example:

“When a locked unspecified length burst starts, new access does not start until current master i releases bus by lowering HBUSREQi.”

\[ \forall_i \ G( \ HMASTLOCK \land HBURST=INCR \land HMASTER=i \land \text{START} \rightarrow X(\neg\text{START} \lor \neg\text{HBUSREQ}_i) ) \]
New Spec

Most recent results go up to 16 masters
Experience

- Expressibility of GR(1) is sufficient
- Specification is short and easy to understand
- Synthesis works!

D(V) V  D(V)
We remove redundancy in code + specify + test.
- Is specifying really easier than coding?

- Specs often ambiguous (AMBA spec is)
  - you also have this problem when writing Verilog code
- How do we debug incorrect specs?
  - Iterative process of refining the spec
  - Only way I know to get the spec consistent & complete!
- How do we specify “requests come as soon as possible”?
- How do we specify systems that are not only correct, but also robust?
Challenges: Size

- Circuits are LARGE, size depends on parameter (#masters)
  - Quite unlike a manual implementation
- Size depends strongly on formulation of specification
  - Smarter Synthesis needed

What else can we use synthesis for?
Other Applications: Robots

Temporal logic motion planning for robots
[FainekosGirardKress-GazitPappas09]
Visit $\pi_1$ and $\pi_2$ repeatedly, avoid black box
int Fill = 1;
int Render = 0;
int i = j = 0;

fill() {
    if (i < N) {
        Im[Fill][i] = read();
i += 1;
goto L1;
    } Fill ^= 1;
    Render ^= 1;
i = 0;
goto L1;
}

render() {
    L1: if (j < N) {
        L2: write(Im[Render][j]);
        L3: j += 1;
        L4: goto L1;
    }
    L5: j = 0;
    L6: goto 1;
}

main() {
    fill() || render();
}
Other Applications: Atomicity

```
int Fill = 1;
int Render = 0;
int i = j = 0;

fill() {
    L1: if (i < N) {
        Im[Fill][i] = read();
        i += 1;
        goto L1;
    } Fill ^= 1;
    Render ^= 1;
    i = 0;
    goto L1;
}

render() {
    L1: if (j < N) {
        write(Im[Render][j]);
        j += 1;
        goto L1;
    } j = 0;
    goto 1;
}

main() {
    fill() || render();
}
```

Which atomic sections do we need to prevent race condition?

[VechevYahavYorsh10]
Challenges / Extensions

- How do we synthesize a network of systems? [PnueliRosner90]
- How do we synthesize asynchronous systems? [PnueliRosner89, Pnueli et al. 06]
- How do we synthesize timed systems? [Pnueli et al.]
- How do we synthesize mixed systems?
Conclusions

- Pnueli’s work – from seminal to practical
- Amir opened up multiple fields to whole groups of researchers
- Expect to hear more!