Specification Format for Reactive Synthesis Problems

Ayrat Khalimov
SYNT 2015
• “Every request should be granted”: $G(r \rightarrow Fg)$

• “No spurious grants”

Let’s specify “spurious grants” in RE:

$$(.,..)\ast (.,g)(\neg r, \neg g)^+ (\neg r, g)$$
In LTL: $(.,.)^*(.,g)(\neg r, \neg g)^+ (\neg r, g)$

- $F(g U \neg r \neg g U \neg r g) \rightarrow$ (NO! It accepts $(r \neg g)(\neg r g)$)

- $F(g U X(\neg r \neg g U X \neg r g))$?

- $F(g \land (g U (\neg r \neg g \land (\neg r \neg g U \neg r g))))$
Synthesis flow

LTL properties → synthesizer → implementation
Synthesis flow

LTL properties

ωRE
automata
partial implementations

format that supports these all

synthesizer that can handle the format

implementation

format that supports these all
Synthesis flow

- LTL properties
- \( \omega \text{RE} \) automata
- Partial implementations

Translator into SYNTCOMP

Any SYNTCOMP synthesizer

Implementation
Outline of the talk

- LTL properties
- ωRE automata
- partial implementations

translator into SYNTCOMP

any SYNTCOMP synthesizer → implementation

new format (extended SMV)
translator extended SMV -> SYNTCOMP
synthesis example: a Huffman encoder
Format requirements

- embedded into existing programming language
- modular
- property language agnostic (LTL, $\omega$RE, automata...)
- fast synthesizers
Proposed format

• embedded into existing programming language
  - SMV
• modular
  - part of SMV
• property language agnostic (LTL, ωRE, automata...)
  - automata
• fast synthesizers
  - SYNTCOMP
Comparison with ([1])([2])

- embedded into existing programming language
  - SMV (SMV) (Promela)
- modular
  - part of SMV (part of SMV) (part of Promela)
- property language agnostic (LTL, ωRE, automata...)
  - automata (LTL patterns) (LTL + relations)
- fast synthesizers
  - SYNTCOMP (original GR1) (SLUGS GR1)
EXTENDED SMV
MODULE main
VAR
  input: 0..10;
  state: boolean;
  x: 0..10;
DEFINE
  x_is_2input := (x = input + input);
ASSIGN
  init(state) := FALSE;
  next(state) := (x = 0 | x_is_2input);
  init(x) := 0;
  next(x) := x + input;
LTLSPEC
  G(state | (x != 10))
MODULE module1(i1,i2)
VAR
  x: ...
...

MODULE module2(i1)
VAR
  out: ...

MODULE main
VAR
  input: ...
VAR
  m1: module1(input, m2.out);
  m2: module2(m1.x);
MODULE helper1(input1,input2) // we can define and use SMV modules as usually
VAR
  state: 0..100;
DEFINE
  reached42 := state=42;
...

MODULE main // module ‘main’ contains a specification
VAR
  CPUread: boolean; // only boolean is allowed
VAR -- controllable
  valueOut: boolean; // only boolean is allowed
VAR
  h: helper1(readA, valueOut); // we can instantiate modules as usually
DEFINE
  // signals defined in the module can be referred to in the property automata
  a := TRUE;
  b := FALSE;

  writtenA := CPUwrite & valueIn=a & done;
  readA := CPUread & valueOut=a & done;
  is42 := h.reached42;
  ...
  // thus we can use variables ‘is42’, ‘readA’, ‘writtenA’ in property automata below

SYS_AUTOMATON_SPEC // guarantees in the GOAL automata format
  guarantee1.gff;
  !guarantee2.gff; // ‘!’ signals to negate the automaton

ENV_AUTOMATON_SPEC // assumptions in the GOAL automata format
  assumption1.gff;
  !assumption2.gff;
  ...

MODULE helper1(input1,input2)  // we can define and use SMV modules as usually
VAR
  state: 0..100;
DEFINE
  reached42 := state=42;
...

MODULE main  // Only main can have specifications
VAR
  CPUread: boolean;  // only boolean is allowed

VAR --controllable
  valueOut: boolean;  // only boolean is allowed

VAR
  h: helper1(readA, valueOut);  // we can instantiate modules as usually

DEFINE
  // signals defined in the module can be referred to in the property automata
  a := TRUE;
  b := FALSE;

  writtenA := CPUwrite & valueIn=a & done;
  readA := CPUread & valueOut=a & done;
  is42 := h.reached42;
...
  // thus we can use signals (i.e. `a`) from helper module in the automaton below

SYS_AUTOMATON SPEC
  guarantee1.gff;
  !guarantee2.gff;  // `!' signals to negate the automaton

ENV_AUTOMATON SPEC
  assumption1.gff;
  !assumption2.gff;
  ...

Extended SMV
TRANSLATION INTO SYNTCOMP
Standard: $\mathbf{G} \neg \text{bad}$

Extended with liveness:

$$(\neg \text{bad} \ \mathbf{W} \ \neg \text{inv}) \ \land \ (\mathbf{G} \ \text{inv} \ \rightarrow \ \mathbf{GF} \ \text{just})$$
Working flow

Original specification (extended SMV) → using GOAL → SMV → using smvflaten → flattened SMV → using smvtoaig → extended SYNTCOMP → justice_2_safety.py → standard SYNTCOMP → synthesizer

aisy.py or from SYNTCOMP

automata:
- determinization
- complementation

flattening into a boolean SMV module

boolean SMV to AIGER translation
SYNTHEIZING HUFFMAN ENCODER
Huffman encoding

A, B, C, ...

encoder

01, 101, 1101, ...

decoder

A, B, C, ...

“more often appearing letters have shorter ciphers”
### Letters Frequency Table

<table>
<thead>
<tr>
<th>Letter</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>8</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>4</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>3</td>
</tr>
<tr>
<td>F</td>
<td>3</td>
</tr>
<tr>
<td>G</td>
<td>1</td>
</tr>
<tr>
<td>H</td>
<td>1</td>
</tr>
<tr>
<td>I</td>
<td>2</td>
</tr>
<tr>
<td>J</td>
<td>1</td>
</tr>
<tr>
<td>K</td>
<td>1</td>
</tr>
<tr>
<td>L</td>
<td>1</td>
</tr>
<tr>
<td>M</td>
<td>1</td>
</tr>
<tr>
<td>N</td>
<td>3</td>
</tr>
<tr>
<td>O</td>
<td>4</td>
</tr>
<tr>
<td>P</td>
<td>2</td>
</tr>
<tr>
<td>Q</td>
<td>1</td>
</tr>
<tr>
<td>R</td>
<td>1</td>
</tr>
<tr>
<td>S</td>
<td>4</td>
</tr>
<tr>
<td>T</td>
<td>2</td>
</tr>
<tr>
<td>U</td>
<td>1</td>
</tr>
<tr>
<td>V</td>
<td>1</td>
</tr>
<tr>
<td>W</td>
<td>1</td>
</tr>
<tr>
<td>X</td>
<td>2</td>
</tr>
<tr>
<td>Y</td>
<td>1</td>
</tr>
<tr>
<td>Z</td>
<td>1</td>
</tr>
</tbody>
</table>
Synthesizing a Huffman encoder

Specification

A1. “input dataIn is within range 1..27”
A2. “dataIn does not change until incl. the moment when done is high”

G1. \( G(\text{done} \rightarrow X \text{enq}_{\text{dec}}) \)
G2. \( G \neg \text{diff} \)
G3. \( GF \text{ done} \)
The specification:
- # latches = 45
- # AND gates = 3k

The model has:
- # AND gates = 130k (120k)

Timings:
- 2min (4min)

The model is as expected
Conclusion & discussion

• Adapted the SMV format to synthesis tasks
• Provided scripts to translate into the SYNTCOMP

• Is SMV good enough or Verilog should be used?
• Should we support LTL/RE formats?
• Should we support GR1 or full LTL semantics?
• Should we support partial information?
• Simpler ways to translate?

thank you
HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC)

SITUATION:
There are 14 competing standards.

14?! RIDICULOUS!
We need to develop one universal standard that covers everyone’s use cases.

YEAH!

SOON:

SITUATION:
There are 15 competing standards.