Specify, Compile, Run: Hardware from PSL/LTL

Roderick Bloem, Stefan Galler, Barbara Jobstmann, Nir Piterman, Amir Pnueli, Martin Weiglhofer

Why is GR-1 interesting?
Why is GR-1 interesting?

• All operations can be done symbolically
• Game graph can be constructed directly symbolically
• Sub-formulas translated separately (product computation symbolic)
What is symbolic?

• Operating with set of states instead of single states, e.g., using formulas over variables

\[(x = 0) \lor (y \leq 1)\] represents all the state in which \(x = 0\) or \(y \leq 1\)

• Symbolic presentation of transition system (TS)

Given a set of variables \(x_1, \ldots, x_n\)

\(\varphi(x_1, \ldots, x_n)\)…predicate describing with states are initial

\(\tau(x_1, \ldots, x_n, x'_1, \ldots, x'_n)\)…transition predicate describing allowed transitions

\(\psi(x_1, \ldots, x_n)\)…(multiple) predicates describing fair states

(not always used, may more than one)
### LTL subset Corresponds to Symbolic TS

<table>
<thead>
<tr>
<th>Symbolic TS</th>
<th>LTL</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varphi(x_1,..,x_n)$</td>
<td>$\varphi(x_1,..,x_n)$</td>
</tr>
<tr>
<td>$\tau(x_1,..,x_n,x'_1,..,x'_n)$</td>
<td>$\text{always}(\tau(x_1,..,x_n,x'_1,..,x'_n)[x'_i := \text{next}(x_i)])$</td>
</tr>
<tr>
<td>$\psi(x_1,..,x_n)$</td>
<td>$\text{always}(\text{eventually}(\psi(x_1,..,x_n)))$</td>
</tr>
</tbody>
</table>

- Every finite TS can be translated into an LTL
- Every LTL formula that is conjunction of formulas above can be translated directly into TS
- Formula above are sufficient to encode any Buchi automaton, which is just a transition system
GR-1 Specifications

• Two sets of atomic propositions
  – Environment inputs I
  – System outputs O

• Specification has two parts
  – Environment specification
  – System specification

• Particular format of formulas
  – Boolean combination of I and O
  – always(A \rightarrow\text{next}(B))
  – always(eventually(B))
GR-1 Specifications

\[ \alpha_1 \land \ldots \land \alpha_n \rightarrow \beta_1 \land \ldots \land \beta_m \]

Environment:
i2
i1
always(i2)
always(o1 \rightarrow \text{next}(i1))

System:
o1
not o2
always(not o2)
always(event(o1 \text{ or } i1))
Corresponding Symbolic Game (SG)

Given

- a set of environment variables $x_1,..,x_n$ and
- a set of system variables $y_1,..,y_n$

$\varphi_e(x_1,..,x_n)$...initial states of environment (chooses first)

$\varphi_s(x_1,..,x_n,y_1,..,y_n)$...initial states of system

$\tau_e(x_1,..,x_n,y_1,..,y_n,x'_1,..,x'_n)$...env. transitions

$\tau_s(x_1,..,x_n,y_1,..,y_n,x'_1,..,x'_n,y'_1,..,y'_n)$...sys. transitions

$\psi_e(x_1,..,x_n,y_1,..,y_n)$...(multiple) environment fairness

$\psi_s(x_1,..,x_n,y_1,..,y_n)$...(multiple) system fairness
Given
- a formula $f(x_1,\ldots,x_n,y_1,\ldots,y_n)$
  describing set of target states

$CNext(f) = \forall x'_1,\ldots,x'_n : \tau_e(x_1,\ldots,x_n,y_1,\ldots,y_n,x'_1,\ldots,x'_n) \rightarrow$

$\exists y'_1,\ldots,y'_n : \tau_s(x_1,\ldots,x_n,y_1,\ldots,y_n,x'_1,\ldots,x'_n,y'_1,\ldots,y'_n) \land$

$f(x'_1,\ldots,x'_n,y'_1,\ldots,y'_n)$
Computing Attractor?

Given

- a formula \( f(x_1,\ldots,x_n,y_1,\ldots,y_n) \)
  describing set of target states

\[ Attr_0(f) = \mu X. f \lor \text{CNext}(X) \]
Obtaining a GR-1 Specification

- General formulas are translated to det. Büchi automata and encoded symbolically (if possible)

- Example
  - “If master i owns a locked bus, he must eventually release it”
  - pre = HMASTLOCK ∧ HBURST=INCR ∧ HMASTER=i
  - always(pre → eventually ¬HBUSREQ[i])
  - Modified construction rules [Maidl, Tonetta], e.g., p → φ = ¬ p ∨ (p ∧ φ)
  - Major advantage: no need to build automaton for full spec
GR-1 Game

$\alpha_1 \land \ldots \land \alpha_n \rightarrow \beta_1 \land \ldots \land \beta_m$

$SG_1 \land \ldots \land SG_n \rightarrow SG_1 \land \ldots \land SG_m$

Combining SGs is easy (inexpensive) = formula conjunction
Example

• Server for n clients
  – Request signals $r_1, \ldots, r_n$
  – Grant signals $g_1, \ldots, g_n$

• Protocol
Example

- **Environment**
  - initial: $\neg r_i$
  - always($r_i \neg g_i \rightarrow \text{next}(r_i)$)
  - always($\neg r_i g_i \rightarrow \text{next}(\neg r_i)$)
  - always($r_i$ and $g_i \rightarrow \text{eventually}(\neg r_i)$)

- **System**
  - initial: $\neg g_i$
  - always($\neg r_i \neg g_i \rightarrow \text{next}(\neg g_i)$)
  - always($r_i g_i \rightarrow \text{next}(g_i)$)
  - always($r_i \rightarrow \text{eventually}(g_i)$)

---

Let's construct symbolic game
Example

- **Environment**
  - initial: \( \neg r_i \)
  - always(\( r_i \neg g_i \rightarrow \text{next}(r_i) \))
  - always(\( \neg r_i g_i \rightarrow \text{next}(\neg r_i) \))
  - always(\( r_i \text{ and } g_i \rightarrow \text{eventually}(\neg r_i) \))

- **System**
  - initial: \( \neg g_i \)
  - always(\( \neg r_i \neg g_i \rightarrow \text{next}(\neg g_i) \))
  - always(\( r_i g_i \rightarrow \text{next}(g_i) \))
  - always(\( r_i \rightarrow \text{eventually}(g_i) \))

- **Symbolic TS**
  \( \varphi_e(r_1, r_2) \)
  \( \tau_e(r_1, r_2, g_1, g_2, r'_1, r'_2) \)
  \( \psi_e(r_1, r_2, g_1, g_2) \)

- **Symbolic TS**
  \( \varphi_s(r_1, r_2, g_1, g_2) \)
  \( \tau_s(r_1, r_2, g_1, g_2, r'_1, r'_2, g'_1, g'_2) \)
  \( \psi_s(r_1, r_2, g_1, g_2) \)
Example

• Environment
  - initial: \( \neg r_i \)
  - always(\( r_i \neg g_i \rightarrow \text{next}(r_i) \))
  - always(\( \neg r_i g_i \rightarrow \text{next}(\neg r_i) \))
  - always(\( r_i \text{ and } g_i \rightarrow \text{eventually}(\neg r_i) \))

• Symbolic TS

\[ \varphi_e(r_1, r_2) = \neg r_1 \land \neg r_2 \]

\[ \tau_e(r_1, r_2, g_1, g_2, r'_1, r'_2) = (r_1 \land \neg g_1 \rightarrow r'_1) \land (\neg r_1 \land g_1 \rightarrow \neg r'_1) \land \\
(r_2 \land \neg g_2 \rightarrow r'_2) \land (\neg r_2 \land g_2 \rightarrow \neg r'_2) \]

\[ \psi_e(r_1, r_2, g_1, g_2) \]
Special Treatment

• always($r_i$ and $g_i$ → eventually($\neg r_i$))

$$\neg r \text{ or } \neg g \quad r \quad \text{ and } g \quad \neg r$$

• Add new variables: $s_1, s_2$ (one for each client)

$$\varphi_e(r_1, r_2) = \neg r_1 \land \neg r_2 \land \neg s_1 \land \neg s_2$$

$$\tau_e(r_1, r_2, g_1, g_2, r'_1, r'_2) = (r_1 \land \neg g_1 \rightarrow r'_1) \land (\neg r_1 \land g_1 \rightarrow \neg r'_1) \land$$

$$\quad (r_2 \land \neg g_2 \rightarrow r'_2) \land (\neg r_2 \land g_2 \rightarrow \neg r'_2) \land$$

$$\quad (\neg s_1 \land (\neg r_1 \lor \neg g_1) \rightarrow \neg s'_1) \land (\neg s_1 \land r_1 \land g_1 \rightarrow s'_1) \land$$

$$\quad (s_1 \land \neg r_1 \rightarrow \neg s'_1) \land (s_1 \land r_1 \rightarrow s'_1) \land ....$$

$$\psi_e(r_1, r_2, g_1, g_2) = \neg s_1 \\
\psi'_e(r_1, r_2, g_1, g_2) = \neg s_2$$
Example (Continue)

• Same for system requirements
• We obtain symbolic game with GR-1 winning condition (implication of generalized Büchi)
• Solve GR-1 game using triple-nested fix-point shown before
• Winning strategy = correct system
Implementations

• Implemented as part of tlv and Jtlv
• Synthesis tool: Anzu
• Realizability checker in requirements analysis tool Rat
• Game solver in NuSMV: NuGAT
• Synthesis tool Ratsy (allows for graphical input of the specification automata and contains algorithm to debug specs)
• GAVS+, Chih-Hong Cheng (TUM/fortiss)
  http://richmodels.epfl.ch/synthesis
Applications

• Hardware Synthesis
• Robot controllers, combine discrete controller with continuous control and achieve, e.g., controllers for cars that autonomously search for parking [Kress-Gazit, Conner, et al.]
• Produce programs from live sequence charts [H. Kugler, I. Segall]
• Core of AspectLTL – an aspect-oriented programming language for LTL specifications [S. Maoz, Y. Sa’ar,]
Anzu

- Synthesis tool based on GR-1 approach
- Command-line tool based on CUDD
- Written in Perl
- Implementation of GR-1 Approach
- Circuit generation
Anzu: Input File

Input and output description

Environment specification
• Specification is given in terms of sequential inputs and outputs
• Flipflops keep track of last-state input & output, plus state of specification automata (state space of the game)
• Strategy-BDD is relation between combinational inputs and combinational outputs
• A circuit is a function
From Strategy BDD to Circuit

BDD represents \( R: I \times O \)

We need a function \( f: I \rightarrow O \) such that

if \( f(i) = o \) then \((i, o) \in R\) or \( \neg \exists o. (i, o) \in R \)

Multiple possibilities

– Kukula & Shiple describe way to obtain \( f': I \times P \rightarrow O \)
  s.t. varying \( P \) leads to all possible \( O \)s for a given \( I \)

– Also in Anzu: a simple cofactor based approach
Anzu: Output
Case Studies

• ARM’s AMBA AHB bus
  – High performance on-chip
  – Arbiter part of bus (determines control signals)

• FV Tutorial Design of IBM
  – Generic Buffer
  – Complete specification
Experiences of Case Studies

• Formalizing a property is sometimes hard
  – Master requests access and specifies type
  – AMBA spec is ambiguous (same in verification)

• No problems in other example (generic buffer) – tutorial design for formal verification of IBM

• Size of arbiter grows rapidly with #master
  – Quite unlike a manual implementation

• Performance depends on structure of spec
  – AMBA (initially 1h for #5, [Chatterjee et al.] 11min, state space, \( \sim 2^{40} \) for #5, \( \sim 2^{100} \) for #16)
Experiences of Case Studies

• Expressibility not a problem
• Deciding realizability is easy
• Specification is short and easy to understand
• Synthesis works