Counter-Strategy Guided Refinement of GR(1) Temporal Logic Specifications

Rajeev Alur, Salar Moarref, and Ufuk Topcu

Presented by
Morteza Lahijanian

Some slides are borrowed from Salar Moarref
Motivation

Specification:
- System must eventually grant every request
- If a request was cleared or granted, then the system must not grant it in the next time step
- If a request is cleared, then the signal is not valid
- System must issue a valid grant infinitely often
Motivation

Specification:

- System must eventually grant every request
- If a request was cleared or granted, then the system must not grant it in the next time step
- If a request is cleared, then the signal is not valid
- System must issue a valid grant infinitely often
Specification:

- **System must eventually grant every request**
- If a request was cleared or granted, then the system must not grant it in the next time step
- If a request is cleared, then the signal is not valid
- System must issue a valid grant infinitely often
Motivation

Specification:

- **System must eventually grant every request**
- If a request was cleared or granted, then the system must not grant it in the next time step
- If a request is cleared, then the signal is not valid
- System must issue a valid grant infinitely often
Motivation

Specification:

- System must eventually grant every request
- If a request was cleared or granted, then the system must not grant it in the next time step
- If a request is cleared, then the signal is not valid
- System must issue a valid grant infinitely often
Motivation

Specification:

- System must eventually grant every request
- If a request was cleared or granted, then the system must not grant it in the next time step
- If a request is cleared, then the signal is not valid
- System must issue a valid grant infinitely often
Motivation

Specification:

- System must eventually grant every request
- If a request was cleared or granted, then the system must not grant it in the next time step
- If a request is cleared, then the signal is not valid
- System must issue a valid grant infinitely often
Motivation

Specification:

- System must eventually grant every request
- If a request was cleared or granted, then the system must not grant it in the next time step
- If a request is cleared, then the signal is not valid
- System must issue a valid grant infinitely often
Motivation

Specification:
- System must eventually grant every request
- If a request was cleared or granted, then the system must not grant it in the next time step
- If a request is cleared, then the signal is not valid
- System must issue a valid grant infinitely often
Motivation

Specification:

- System must eventually grant every request
- If a request was cleared or granted, then the system must not grant it in the next time step
- If a request is cleared, then the signal is not valid
- System must issue a valid grant infinitely often
Motivation

Specification:
- System must eventually grant every request
- If a request was cleared or granted, then the system must not grant it in the next time step
- If a request is cleared, then the signal is not valid
- System must issue a valid grant infinitely often
Motivation

Specification:

- System must eventually grant every request
- If a request was cleared or granted, then the system must not grant it in the next time step
- If a request is cleared, then the signal is not valid
- System must issue a valid grant infinitely often
Motivation

 Specification:

- System must eventually grant every request
- If a request was cleared or granted, then the system must not grant it in the next time step
- If a request is cleared, then the signal is not valid
- **System must issue a valid grant infinitely often**
Motivation

Specification:

- System must eventually grant every request
- If a request was cleared or granted, then the system must not grant it in the next time step
- If a request is cleared, then the signal is not valid
- **System must issue a valid grant infinitely often**
Reactive Synthesis

Given a Specification, synthesize a system that satisfies the specification regardless of how the environment behaves.
Reactive Synthesis

Given a Specification, synthesize a system that satisfies the specification regardless of how the environment behaves.

- **Spec**: formal language, e.g., LTL
- **Game**: system vs. environment
  - env: tries to violate the spec
  - sys: tries to satisfy the spec
- **Unsatisfiable**: no input and output trace that satisfies the spec
- **If satisfiable, realizable?**
  - Yes: sys. has a winning strategy
  - No: env. has a winning strategy
Reactive Synthesis

Given a Specification, synthesize a system that satisfies the specification regardless of how the environment behaves.

- **Spec**: formal language, e.g., LTL
- **Game**: system vs. environment
  - env: tries to violate the spec
  - sys: tries to satisfy the spec
- **Unsatisfiable**: no input and output trace that satisfies the spec
- **If satisfiable, realizable?**
  - Yes: sys. has a winning strategy
  - No: env. has a winning strategy
Given a Specification, synthesize a system that satisfies the specification regardless of how the environment behaves.

- **Spec:** formal language, e.g., LTL
- **Game:** system vs. environment
  - env: tries to violate the spec
  - sys: tries to satisfy the spec
- **Unsatisfiable:** no input and output trace that satisfies the spec
- **If satisfiable, realizable?**
  - Yes: sys. has a winning strategy
  - No: env. has a winning strategy

Can we restrict env. “just enough” so that the sys. can have a winning strategy?
Linear Temporal Logic

Syntax: LTL is defined recursively as

\[ \varphi ::= p \mid \neg \varphi \mid \varphi \lor \psi \mid \varphi \land \psi \mid O \varphi \mid \varphi \mathbf{U} \psi \mid \Diamond \varphi \mid \Box \varphi \]

\[ \varphi \land \psi = \neg (\neg \varphi \lor \neg \psi) \]
\[ \varphi \rightarrow \psi = \neg \varphi \lor \psi \]
\[ \varphi \mathbf{U} \psi : \varphi \varphi \varphi \varphi \varphi \varphi \cdots \psi \psi \varphi \cdots \]
\[ \Diamond \varphi = true \mathbf{U} \varphi : \psi \cdots \varphi \psi \psi \varphi \cdots \]
\[ \Box \varphi = \neg \Diamond \neg \varphi : \varphi \varphi \varphi \varphi \varphi \varphi \cdots \varphi \varphi \varphi \varphi \varphi \cdots \]
Linear Temporal Logic

Syntax: LTL is defined recursively as

\[ \varphi := p \mid \neg \varphi \mid \varphi \lor \psi \mid \varphi \land \psi \mid O \varphi \mid \varphi \mathcal{U} \psi \mid \Diamond \varphi \mid \square \varphi \]

- \( \varphi \land \psi = \neg(\neg \varphi \lor \neg \psi) \)
- \( \varphi \rightarrow \psi = \neg \varphi \lor \psi \)

- \( \varphi \mathcal{U} \psi : \varphi \varphi \varphi \varphi \varphi \varphi \ldots \varphi \psi \varphi \ldots \)
- \( \Diamond \varphi = \text{true} \mathcal{U} \varphi : \psi \ldots \varphi \psi \psi \varphi \ldots \)
- \( \square \varphi = \neg \Diamond \neg \varphi : \varphi \varphi \varphi \varphi \varphi \varphi \varphi \ldots \varphi \varphi \varphi \varphi \varphi \ldots \)

Examples:

Eventually grant a request
Linear Temporal Logic

Syntax: LTL is defined recursively as

\[ \varphi := p \mid \neg \varphi \mid \varphi \lor \psi \mid \varphi \land \psi \mid 0 \varphi \mid \varphi \mathcal{U} \psi \mid \Diamond \varphi \mid \Box \varphi \]

- \[ \varphi \land \psi = \neg (\neg \varphi \lor \neg \psi) \]
- \[ \varphi \rightarrow \psi = \neg \varphi \lor \psi \]
- \[ \varphi \mathcal{U} \psi : \varphi \varphi \varphi \varphi \varphi \varphi \ldots \varphi \psi \varphi \ldots \]
- \[ \Diamond \varphi = \text{true} \mathcal{U} \varphi : \psi \ldots \varphi \psi \psi \varphi \ldots \]
- \[ \Box \varphi = \neg \Diamond \neg \varphi : \varphi \varphi \varphi \varphi \varphi \varphi \varphi \ldots \varphi \varphi \varphi \varphi \varphi \ldots \]

Examples:

Eventually grant a request \[ \varphi = \Box (r \rightarrow \Diamond g) \]
Generalized Reactivity (1)

GR(1): fragment of LTL

$$P = I \cup O$$

$$\varphi = \varphi_e \rightarrow \varphi_s$$

- Env. assumptions
- Sys. behavior

$$\varphi_e = \bigwedge_i \psi_i \bigwedge_i \square \psi_i \bigwedge_i \square \Diamond \psi_i$$
- Initial state
- Safety
- Fairness

$$\varphi_s = \bigwedge_i \psi_i \bigwedge_i \square \psi_i \bigwedge_i \square \Diamond \psi_i$$
- Initial state
- Safety
- Liveness

Why GR(1)?

- Reactive synthesis problem can be solved in exponential time (in comparison to doubly exponential!)

Piterman et al. 2006
Generalized Reactivity (1)

GR(1): \[ \varphi = \varphi_e \rightarrow \varphi_s \]
\[ P = I \cup O \]

Example:
\[ I = \quad O = \]

- System must eventually grant every request
- If a request was cleared or granted, then the system must not grant it in the next time step
- If a request is cleared, then the signal is not valid
- System must issue a valid grant infinitely often
Generalized Reactivity (1)

GR(1): \[ \varphi = \varphi_e \rightarrow \varphi_s \]
\[ P = I \cup O \]

\[ \varphi_s = \bigwedge_i \psi_i \bigwedge_i \square \psi_i \bigwedge_i \square \diamond \psi_i \]

Example:
\[ I = \{r, c\}, \quad O = \{g, v\} \]

- System must eventually grant every request
- If a request was cleared or granted, then the system must not grant it in the next time step
- If a request is cleared, then the signal is not valid
- System must issue a valid grant infinitely often
Generalized Reactivity (1)

GR(1):

\[ \varphi = \varphi_e \rightarrow \varphi_s \]

\[ P = I \cup O \]

Example:

\[ I = \{r, c\}, \quad O = \{g, v\} \]

- System must eventually grant every request
- If a request was cleared or granted, then the system must not grant it in the next time step
- If a request is cleared, then the signal is not valid
- System must issue a valid grant infinitely often
Generalized Reactivity (1)

GR(1):

\[ \varphi = \varphi_e \rightarrow \varphi_s \]

\[ P = I \cup O \]

Example:

\[ I = \{r, c\}, \quad O = \{g, v\} \]

- System must eventually grant every request
- If a request was cleared or granted, then the system must not grant it in the next time step
- If a request is cleared, then the signal is not valid
- System must issue a valid grant infinitely often

\[ \varphi_s = \bigwedge_i \psi_i \bigwedge_i \Box \psi_i \bigwedge_i \Box \Diamond \psi_i \]

Initial state
Safety
liveness

\[ r \rightarrow \text{?} \rightarrow g \]
\[ c \rightarrow \text{?} \rightarrow v \]

\[ \Box (r \rightarrow \Diamond g) \]
\[ \Box((c \lor g) \rightarrow O \neg g) \]
Generalized Reactivity (1)

GR(1):

\[ \varphi = \varphi_e \rightarrow \varphi_s \]

\[ P = I \cup O \]

Example:

\[ I = \{ r, c \}, \quad O = \{ g, v \} \]

- System must eventually grant every request
- If a request was cleared or granted, then the system must not grant it in the next time step
- If a request is cleared, then the signal is not valid
- System must issue a valid grant infinitely often

\[ \varphi_s = \bigwedge_i \psi_i \quad \bigwedge_i \square \psi_i \quad \bigwedge_i \square \Diamond \psi_i \]

Initial state  
Safety  
liveness

\[ \square \rightarrow \Diamond \]  
\[ \square \rightarrow \Diamond \]  
\[ \square \rightarrow \Diamond \]  

\[ \square (r \rightarrow \Diamond \ g) \]

\[ \square ((c \lor g) \rightarrow O \neg g) \]

\[ \square (c \rightarrow \neg v) \]
GR(1):

\[ \varphi = \varphi_e \rightarrow \varphi_s \]

\[ P = I \cup O \]

Example:

\[ I = \{r, c\}, \quad O = \{g, v\} \]

- System must eventually grant every request
- If a request was cleared or granted, then the system must not grant it in the next time step
- If a request is cleared, then the signal is not valid
- System must issue a valid grant infinitely often

\[ \square(r \rightarrow \Diamond g) \]

\[ \square((c \lor g) \rightarrow O \neg g) \]

\[ \square(c \rightarrow \neg v) \]

\[ \square \Diamond (g \land v) \]
Realizable?

Example:

\[
\varphi = \varphi_e \rightarrow \varphi_s
\]

\[
I = \{r, c\}, \quad O = \{g, v\}
\]

Say: \( \varphi_e = true \)

\[
\varphi_s = \Box (r \rightarrow \Diamond g)
\]

\[
\land \Box ((c \lor g) \rightarrow O \neg g)
\]

\[
\land \Box (c \rightarrow \neg v)
\]

\[
\land \Box \Diamond (g \land v)
\]
Realizable?

Example:

\[ \varphi = \varphi_e \rightarrow \varphi_s \]

\[ I = \{r, c\}, \quad O = \{g, v\} \]

Say: \( \varphi_e = \text{true} \)

\[ \varphi_s = \square(r \rightarrow \lozenge g) \]
\[ \quad \land \square((c \lor g) \rightarrow \neg O \neg g) \]
\[ \quad \land \square(c \rightarrow \neg v) \]
\[ \quad \land \lozenge (g \land v) \]

Realizable?

What if env keeps \( r \) and \( c \) high all the time?
Realizable?

Example:

$$\varphi = \varphi_e \to \varphi_s$$

$$I = \{r, c\}, \quad O = \{g, v\}$$

Say: \(\varphi_e = \text{true}\)

\[\begin{align*}
\varphi_s &= \Box (r \to \Diamond g) \\
&\quad \land \Box ((c \lor g) \to O \neg g) \\
&\quad \land \Box (c \to \neg v) \\
&\quad \land \Box \Diamond (g \land v)
\end{align*}\]

Realizable?

What if env keeps \(r\) and \(c\) high all the time?

No system can satisfy the spec.
Motivation

- Developing correct and complete formal specification
  - Challenging and tedious
  - Initial specifications often unrealizable
- Unrealizable specification
  - Often due to inadequate environment assumptions
  - Cannot be executed or simulated
- Counter-strategies?
Motivation

- Developing correct and complete formal specification
  - Challenging and tedious

**Goal:** automatically refining the constraints over the environment by adding assumptions in order to achieve realizability

- Cannot be executed or simulated
- Counter-strategies?
Applications

- Constructing an environment model
- Giving the user an insight into the specification
- Correcting the specification
- Constructing the interface specification
- And more..
Counter-Strategy Guided Refinement

1. Specification → Realizable
   - Yes → Done
   - No:
     - Choose & add
     - Generate candidates
     - Subset of variables
     - Counter-strategy
     - Patterns synthesis
     - Realizable
       - Yes → Done
       - No → Choose & add
Counter-Strategy Guided Refinement

1. Specification
2. Generate candidates
   - Choose & add
3. Realizable
   - Yes: Done
   - No: Counter-strategy
4. Counter-strategy
   - Subset of variables
   - Patterns synthesis
A winning strategy for the environment

Represented as a Moore machine
Counter-Strategy

- A winning strategy for the environment
- Represented as a Moore machine

\[ \varphi_s = \square(r \rightarrow \Diamond g) \]
\[ \land \square((c \lor g) \rightarrow 0 \neg g) \]
\[ \land \square(c \rightarrow \neg v) \]
\[ \land \square\Diamond (g \land v) \]
Candidate Assumption

- Infer LTL formulas which hold over all runs of the counter-strategy.
- Choose one and add its complement as assumption to the specification.
  - Strengthen the environment assumptions.
- Remove the counter-strategy from admissible environment behaviors.

$c$ holds all the time, so a candidate is $\Box \neg c$. 

Diagram: 
- States $S_0$, $S_1$, $S_2$, $S_3$.
- Edges with conditions: $g = 0$, $g = 1$.
- Initial state $S_0$ with $r=1, c=1$.
- Transition from $S_1$ to $S_2$ with $r=0, c=1$.
- Transition from $S_2$ to $S_3$ with $r=1, c=1$. 

Diagram shows transitions with conditions on the edges.
Given two equivalent formulas, we say that

- $\varphi_1$ is **stronger** than $\varphi_2$ if $\varphi_1 \rightarrow \varphi_2$
- $\neg \psi_1$ is a **weaker** assumption than $\neg \psi_2$ if $\neg \psi_2 \rightarrow \neg \psi_1$
- Adding $\neg \psi_2$ restricts the env. more than adding $\neg \psi_1$
Given two equivalent formulas, we say that

- \( \varphi_1 \) is **stronger** than \( \varphi_2 \) if \( \varphi_1 \rightarrow \varphi_2 \)
- \( \neg \psi_1 \) is a **weaker** assumption than \( \neg \psi_2 \) if \( \neg \psi_2 \rightarrow \neg \psi_1 \)
- Adding \( \neg \psi_2 \) restricts the env. more than adding \( \neg \psi_1 \)

**Examples:**
Both hold over all runs of \( \mathcal{M} \)

\[
\psi_1 = \Diamond (c \land \neg r) \\
\psi_2 = \Diamond (c) \\
\neg \psi_1 = \Box (\neg c \lor r) \\
\neg \psi_2 = \Box (\neg c)
\]
Removing Restrictive Formulas

- Given two equivalent formulas, we say that
  - $\varphi_1$ is **stronger** than $\varphi_2$ if $\varphi_1 \rightarrow \varphi_2$
  - $\neg \psi_1$ is a **weaker** assumption than $\neg \psi_2$ if $\neg \psi_2 \rightarrow \neg \psi_1$
  - Adding $\neg \psi_2$ restricts the env. more than adding $\neg \psi_1$

**Examples:**
Both hold over all runs of $Mc$

- $\psi_1 = \diamond (c \land \neg r)$
- $\psi_2 = \diamond (c)$

$\neg \psi_1 = \Box (\neg c \lor r)$
$\neg \psi_2 = \Box (\neg c)$

$\varphi_s = \Box (r \rightarrow \Diamond g)$
$\land \Box (((c \lor g) \rightarrow 0 \neg g))$
$\land \Box (c \rightarrow \neg v)$
$\land \Box \Diamond (g \land v)$
Counter-Strategy Guided Refinement

1. Specification
2. Generate candidates
3. Counter-strategy
4. Patterns synthesis
5. Subset of variables
6. Realizable
7. Choose & add
8. Done
9. Yes
10. No
Patterns

• LTL formulas of the form
  • $◊ □ ψ$, $◊ ψ$, $◊ (ψ ∧ O ψ')$, and
• Hold over all runs of the abstraction of counter-strategy
• Synthesize using graph search algorithms
Generalized Reactivity (1)

GR(1):

\[ P = I \cup O \]

\[ \varphi = \varphi_e \rightarrow \varphi_s \]

- Env. assumptions
- Sys. behavior

\[ \varphi_e = \bigwedge_i \psi_i \bigwedge_i \Box \psi_i \bigwedge_i (\psi_i \rightarrow O \psi_i) \bigwedge_i \Box \Diamond \psi_i \]
Patterns

- LTL formulas of the form
  - \( \Diamond \Box \psi \), \( \Diamond \psi \), \( \Diamond (\psi \land O \psi') \), and
- Hold over all runs of the abstraction of counter-strategy
- Synthesized using graph search algorithms
Eventually Patterns

- $q_i$ is eventually visited

- **Eventually configuration:** any run of FTS eventually visits a state from configuration $C$

---

**Algorithm 2: Generating $\Diamond \psi$ patterns**

**Input:** Finite state transition system $\mathcal{T}_c = \langle Q, \{q_0\}, \delta \rangle$

**Input:** $\beta$, maximum number of states in generated patterns

**Output:** a set of patterns of the form $\Diamond \psi$ where $\mathcal{T}_c \models \Diamond \psi$

1. Patterns := $\{\Diamond q_0\}$;
2. $\Diamond$Configurations := $\{q_0\}$;
3. **foreach** $Q' \subseteq Q - \{q_0\}$ with non-decreasing order of $|Q'|$ where $|Q'| \leq \beta$ **do**
   4. if $\not\exists Q'' \in \Diamond$Configurations s.t. $Q'' \subseteq Q'$ **then**
   5. Let $\mathcal{T}'_c = \langle Q - Q', \{q_0\}, \delta' \rangle$ where $\delta' = \{(q, q') \in \delta | q \not\in Q' \land q' \not\in Q'\}$;
   6. if there is no infinite run from $q_0$ in $\mathcal{T}'_c$ **then**
      7. Add $Q'$ to $\Diamond$Configurations;
      8. Let $\psi = \Diamond \bigvee_{q_i \in Q'} q_i$;
      9. Add $\psi$ to Patterns;
4. return Patterns;
Eventually Patterns

- $q_i$ is eventually visited
- **Eventually configuration:** any run of FTS eventually visits a state from configuration $C$

---

**Algorithm 2: Generating $\Diamond \psi$ patterns**

**Input:** Finite state transition system $T_c = \langle Q, \{q_0\}, \delta \rangle$

**Input:** $\beta$, maximum number of states in generated patterns

**Output:** a set of patterns of the form $\Diamond \psi$ where $T_c \models \Diamond \psi$

1. Patterns := $\{\Diamond q_0\}$;
2. $\Diamond$Configurations := $\{q_0\}$;
3. **foreach** $Q' \subseteq Q - \{q_0\}$ with non-decreasing order of $|Q'|$ where $|Q'| \leq \beta$ **do**
4. **if** $\not\exists Q'' \in \Diamond$Configurations s.t. $Q'' \subseteq Q'$ **then**
5. **if** there is no infinite run from $q_0$ in $T'_c$ **then**
6. Add $Q'$ to $\Diamond$Configurations;
7. Let $\psi = \Diamond \bigvee_{q_i \in Q'} q_i$;
8. Add $\psi$ to Patterns;
9. return Patterns;
Eventually Patterns

+ $q_i$ is eventually visited

+ **Eventually configuration**: any run of FTS eventually visits a state from configuration $C$

---

**Algorithm 2: Generating $\Diamond\psi$ patterns**

**Input**: Finite state transition system $\mathcal{T}_c = \langle Q, \{q_0\}, \delta \rangle$

**Input**: $\beta$, maximum number of states in generated patterns

**Output**: a set of patterns of the form $\Diamond\psi$ where $\mathcal{T}_c \models \Diamond\psi$

1. Patterns := $\{\Diamond q_0\}$;
2. $\Diamond$Configurations := $\{q_0\}$;
3. **foreach** $Q' \subseteq Q - \{q_0\}$ with non-decreasing order of $|Q'|$ where $|Q'| \leq \beta$ **do**
   
   **if** $\nexists Q'' \in \Diamond$Configurations s.t. $Q'' \subseteq Q'$ **then**
   
   Let $\mathcal{T}_c' = \langle Q - Q', \{q_0\}, \delta' \rangle$ where $\delta' = \{(q, q') \in \delta | q \notin Q' \land q' \notin Q'\}$;
   
   **if** there is no infinite run from $q_0$ in $\mathcal{T}_c'$ **then**
   
   Add $Q'$ to $\Diamond$Configurations;
   
   Let $\psi = \Diamond \bigvee_{q_i \in Q'} q_i$;
   
   Add $\psi$ to Patterns;
4. return Patterns;
 Eventually Patterns

- $q_i$ is eventually visited
- **Eventually configuration:** any run of FTS eventually visits a state from configuration $C$

---

**Algorithm 2: Generating $\diamond \psi$ patterns**

**Input:** Finite state transition system $\mathcal{T}_c = \langle Q, \{q_0\}, \delta \rangle$

**Input:** $\beta$, maximum number of states in generated patterns

**Output:** a set of patterns of the form $\diamond \psi$ where $\mathcal{T}_c \models \diamond \psi$

```
1 Patterns := \{\diamond q_0\};
2 \diamond Configurations := \{q_0\};
3 \foreach Q' \subseteq Q - \{q_0\} \text{ with non-decreasing order of} \mid Q' \mid \text{ where } \mid Q' \mid \leq \beta \text{ do}
4 \quad \text{if } \forall Q'' \in \diamond Configurations \text{ s.t. } Q'' \subseteq Q' \text{ then}
5 \quad \quad \text{Let } \mathcal{T}'_c = \langle Q - Q', \{q_0\}, \delta' \rangle \text{ where}
6 \quad \quad \quad \delta' = \{(q, q') \in \delta | q \not\in Q' \land q' \not\in Q'\};
7 \quad \quad \text{if there is no infinite run from } q_0 \text{ in } \mathcal{T}'_c \text{ then}
8 \quad \quad \quad \text{Add } Q' \text{ to } \diamond Configurations;
9 \quad \quad \quad \text{Let } \psi = \diamond \bigvee_{q_i \in Q'} q_i;
10 \quad \quad \quad \text{Add } \psi \text{ to Patterns;}
11 \quad return Patterns;
```

\[ \diamond q_0 \]

- $\{q_1\}$, $\{q_2\}$, $\{q_3\}$ not in $C$
- $\{q_1, q_3\}$ & $\{q_2, q_3\}$ in $C$
- $\{q_1, q_2, q_3\}$ in $C$, NOT MINIMAL
Eventually Patterns

- \( q_i \) is eventually visited
- **Eventually configuration**: any run of FTS eventually visits a state from configuration \( C \)

---

**Algorithm 2: Generating \( \Diamond \psi \) patterns**

**Input**: Finite state transition system \( T_c = (Q, \{q_0\}, \delta) \)
**Input**: \( \beta \), maximum number of states in generated patterns
**Output**: a set of patterns of the form \( \Diamond \psi \) where \( T_c \models \Diamond \psi \)

1. Patterns := \( \{\Diamond q_0\} \);
2. Configurations := \( \{q_0\} \);
3. **foreach** \( Q' \subseteq Q - \{q_0\} \) with non-decreasing order of \( |Q'| \) where \( |Q'| \leq \beta \) **do**
4. **if** \( \forall Q'' \in \Diamond \text{ Configurations } \) s.t. \( Q'' \subseteq Q' \) **then**
5. **let** \( T'_c = (Q - Q', \{q_0\}, \delta') \) where \( \delta' = \{(q, q') \in \delta | q \notin Q' \land q' \notin Q'\} \);
6. **if** there is no infinite run from \( q_0 \) in \( T'_c \) **then**
7. Add \( Q' \) to \( \Diamond \text{ Configurations} \);
8. **let** \( \psi = \Diamond \bigvee_{q_i \in Q'} q_i \);
9. Add \( \psi \) to Patterns;
10. return Patterns;

\( \Diamond q_0 \)

\{\( q_1 \), \( q_2 \), \( q_3 \)\} not in \( C \)

\{\( q_1 \), \( q_3 \)\} & \{\( q_2 \), \( q_3 \)\} in \( C \), NOT MINIMAL

\{\( \Diamond q_0 \), \( \Diamond (q_1 \lor q_3) \), \( \Diamond (q_2 \lor q_3) \)\}
Eventually Always Patterns

- Complement of liveness/fairness formulas
- Find $Q_{cycles}$

\[ \psi = \Diamond \Box \bigvee_{q \in Q_{cycle}} q \]
Eventually Always Patterns

- Complement of liveness/fairness formulas
- Find $Q^{cycles}$

$$
\psi = \Diamond \Box \bigvee_{q \in Q^{cycle}} q
$$

![Diagram of a strongly connected component including a cycle](image)
Eventually Always Patterns

- Complement of liveness/fairness formulas
- Find $Q_{cycles}$

$$\psi = \diamond \Box \bigvee_{q \in Q_{cycle}} q$$

$$\diamond \Box (q_1 \lor q_2 \lor q_3)$$

Strongly connected components including cycle
Eventually – Next Patterns

\[ \Diamond (\psi_1 \land \Box \psi_2) \]

- First find \( \Diamond \psi_1 \)
- For each of \( \Diamond \psi_1 \) the following is generated

\[ \Diamond (\psi_1 \land \Box \bigvee_{q \in Next(\psi_1)} q) \]

\[ Next(\psi_1) = \{ q_i \in Q \mid \exists q_j \in C \text{ s.t. } (q_j, q_i) \in \delta \} \]
Eventually – Next Patterns

\( \bigdiamond (\psi_1 \land \bigcirc \psi_2) \)

- First find \( \bigdiamond \psi_1 \)
- For each of \( \bigdiamond \psi_1 \) the following is generated

\[ \bigdiamond (\psi_1 \land \bigcirc \bigvee_{q \in \text{Next}(\psi_1)} q) \]

\( \text{Next}(\psi_1) = \{q_i \in Q \mid \exists q_j \in C \text{ s.t. } (q_j, q_i) \in \delta\} \)

Recall: \[ \{\bigdiamond q_0, \bigdiamond (q_1 \lor q_3), \bigdiamond (q_2 \lor q_3)\} \]
Eventually – Next Patterns

\[ \diamond (\psi_1 \land \bigcirc \psi_2) \]

- First find \( \diamond \psi_1 \)
- For each of \( \diamond \psi_1 \) the following is generated

\[ \diamond (\psi_1 \land \bigcirc \bigvee_{q \in \text{Next}(\psi_1)} q) \]

\[ \text{Next}(\psi_1) = \{q_i \in Q \mid \exists q_j \in \mathcal{C} \text{ s.t. } (q_j, q_i) \in \delta\} \]

Recall:

\[ \{\diamond q_0, \diamond (q_1 \lor q_3), \diamond (q_2 \lor q_3)\} \]

- \( \diamond (q_0 \land \bigcirc (q_1 \lor q_3)) \)
- \( \diamond ((q_1 \lor q_3) \land \bigcirc (q_2 \lor q_3)) \)
- \( \diamond ((q_2 \lor q_3) \land \bigcirc (q_1 \lor q_3)) \)
\[\diamondsuit (q_1 \lor q_2 \lor q_3)\]
\[\diamondsuit q_0, \diamondsuit q_1, \diamondsuit q_2, \diamondsuit q_3,\]
\[\diamondsuit (q_0 \land \Box q_1), \diamondsuit (q_1 \land \Box q_2), \diamondsuit (q_2 \land \Box q_3), \diamondsuit (q_3 \land \Box q_1)\]
Counter-Strategy Guided Refinement

1. Specification
2. Generate candidates
   - Choose & add
   - Subset of variables
   - Patterns synthesis
3. Realizable
   - Yes → Done
   - No → Counter-strategy
Synthesizing Candidate Patterns

- Replace states in patterns with state predicates
  \[ \Diamond \Box (q_1 \lor q_2 \lor q_3) \]

- Complement the formula
  \[ \Box \Diamond \neg c \]

- Add to the formula and check again:
  \[ \varphi_e \land \Box \Diamond \neg c \rightarrow \varphi_s \]
A subset of variables for each pattern type
- may contribute to unrealizability problem
- are underspecified

Smaller subset of variables
- Simpler formulas
- More restrictive

\(\Box (c \lor r)\) vs. \(\Box c\)
Properties of patterns Assumptions

- Synthesized patterns are
  - Minimal
    - Removing any state leads to unsatisfiable formulas
    - Strongest formulas of the specified form

- Synthesized assumptions
  - Rule out the counter-strategy
  - Restricts the environment as weakly as possible
Case Study

- Lift Controller (env. = buttons)
  - Once request is made, it cannot be withdrawn
  - Once the request is fulfilled, it is removed
  - Initially no requests

\[
\phi_e = \phi_{init}^e \land \phi_{11}^e \land \phi_{12}^e \land \phi_{13}^e \land \phi_{21}^e \land \phi_{22}^e \land \phi_{23}^e  \\
\phi_{init}^e = (\neg b_1 \land \neg b_2 \land \neg b_3)  \\
\phi_{1i}^e = \Box (b_i \land f_i \rightarrow \bigcirc \neg b_i)  \\
\phi_{2i}^e = \Box (b_i \land \neg f_i \rightarrow \bigcirc b_i) \quad 1 \leq i \leq 3
\]
Case Study

- Lift Controller (sys. = lift)
  - Must be only on one of the floors at each time
  - Can move only one floor at each time
  - Initially starts on the first floor

\[
\phi_s = \phi_{\text{init}}^s \land \phi_1^s \land \bigwedge_i \phi_{2,i}^s \land \phi_3^s \land \bigwedge_j \phi_{4,j}^s \land \phi_5^s
\]

\[
\phi_{\text{init}}^s = f_1 \land \neg f_2 \land \neg f_3,
\]

\[
\phi_1^s = \Box (\neg(f_1 \land f_2) \land \neg(f_2 \land f_3) \land \neg(f_1 \land f_3)),
\]

\[
\phi_{2,i}^s = \Box (f_i \rightarrow \bigcirc (f_{i-1} \lor f_i \lor f_{i+1})),
\]

\[
\phi_3^s = \Box ((f_1 \land \bigcirc f_2) \lor (f_2 \land \bigcirc f_3) \rightarrow (b_1 \lor b_2 \lor b_3)),
\]

and

\[
\phi_{4,j}^s = \Box \Diamond (b_j \rightarrow f_j).
\]
Case Study

- Lift Controller (sys. = lift)
  - Must be only on one of the floors at each time
  - Can move only one floor at each time
  - Initially starts on the first floor

\[ \phi_s = \phi_{\text{init}}^s \land \phi_1^s \land \phi_2^s \land \phi_3^s \land \phi_4^s \land \phi_5^s \]

\[ \phi_{\text{init}}^s = f_1 \land \neg f_2 \land \neg f_3, \]

\[ \phi_1^s = \Box(\neg(f_1 \land f_2) \land \neg(f_2 \land f_3) \land \neg(f_1 \land f_3)), \]

\[ \phi_{2,i}^s = \Box(f_i \rightarrow \Box(f_{i-1} \lor f_i \lor f_{i+1})), \]

\[ \phi_3^s = \Box((f_1 \land \Box f_2) \lor (f_2 \land \Box f_3) \rightarrow (b_1 \lor b_2 \lor b_3)), \]

and

\[ \phi_{4,j}^s = \Box \Diamond (b_j \rightarrow f_j). \]
Case Study

- Lift Controller (sys. = lift)
  - Must be only on one of the floors at each time
  - Can move only one floor at each time
  - Initially starts on the first floor

\[ \phi_s = \phi_{init}^s \land \phi_1^s \land \phi_{2,i}^s \land \phi_3^s \land \phi_{4,j}^s \land \phi_5^s \]

\[ \phi_{init}^s = f_1 \land \neg f_2 \land \neg f_3, \]
\[ \phi_1^s = \Box (\neg(f_1 \land f_2) \land \neg(f_2 \land f_3) \land \neg(f_1 \land f_3)), \]
\[ \phi_{2,i}^s = \Box (f_i \rightarrow \bigcirc (f_{i-1} \lor f_i \lor f_{i+1})), \]
\[ \phi_3^s = \Box ((f_1 \land \bigcirc f_2) \lor (f_2 \land \bigcirc f_3) \rightarrow (b_1 \lor b_2 \lor b_3)), \]

and
\[ \phi_{4,j}^s = \Box \Diamond (b_j \rightarrow f_j). \]

Realizable
\[ \phi = \phi_e \rightarrow \phi_s \]

Unrealizable
\[ \phi' = \phi_e \rightarrow \phi_s \land \Box \phi_{5,j}^s \]
\[ \phi_{5,j}^s = \Box \Diamond (f_j) \]
Refinement with \{b_1, b_2, b_3\}

- Generated candidates:

\[ \psi_1 = \Box \Diamond (b_1 \lor b_2 \lor b_3) \]

\[ \psi_2 = \Box ((\neg b_1 \land \neg b_2 \land \neg b_3) \rightarrow \Diamond (b_1 \lor b_2 \lor b_3)) \]
Case Study

- Refinement with \( \{b_1, b_2, b_3\} \)
  - Generated candidates:

More reasonable

\[
\psi_1 = \Box \Diamond (b_1 \lor b_2 \lor b_3)
\]

\[
\psi_2 = \Box (\neg b_1 \land \neg b_2 \land \neg b_3) \rightarrow \Box (b_1 \lor b_2 \lor b_3)
\]

Realizable

\[
\phi' = \phi_e \rightarrow \phi_s \land \bigwedge_j \phi_{5,j}^s
\]

\[
\phi_{5,j}^s = \Box \Diamond (f_j)
\]
Conclusion

- Counter-strategy guided refinement of GR(1) specifications
- Refining the unrealizable specification by adding assumptions
  - Simple GR(1) formulas
    - Easy to understand and validate by the user
  - As weak as possible in the specified structure