To Assume, Or Not To Assume
[Computing Adequately Permissive Assumptions for Synthesis]

TACAS '23

Ashwani Anand
MPI-SWS

with K. Mallik (ISTA), S.P. Nayak (MPI-SWS), and A.-K. Schmuck (MPI-SWS)
No cleaning strategy may exist if Environment is adversarial.
No cleaning strategy may exist if Environment is adversarial.

Environment rarely acts adversarially.
No cleaning strategy may exist if Environment is adversarial.

Humans may know this assumption. Computers don't.

Environment rarely acts adversarially.
Reactive Synthesis

Task for a system

Clean a house

Implementation

Roomba software
Reactive Synthesis

Task for a system

Clean a house

Implementation

Roomba software

Works well if no assumption is needed.
Reactive Synthesis

Task for a system

Clean a house

Implementation

Roomba software

Works well if no assumption is needed.

Might fail without some assumptions. E.g. the user will not block the path.
What has been done?

Chatterjee et. al. [CONCUR'08] introduce the notion of assumption for games on graphs.
What has been done?

Chatterjee et. al. [CONCUR'08] introduce the notion of assumption for games on graphs.

Their method requires solving NP-hard problem.
What has been done?

Chatterjee et. al. [CONCUR’08] introduce the notion of assumption for games on graphs.

Their method requires solving NP-hard problem.

Fails to give a sufficient assumption, even if it exists.
What has been done?

Chatterjee et. al. [CONCUR'08] introduce the notion of assumption for games on graphs.

Their method requires solving NP-hard problem.

Fails to give a sufficient assumption, even if it exists.
Reactive Synthesis : The Standard
Reactive Synthesis: The Standard

Convert to a game on graph
Reactive Synthesis: The Standard

Convert to a game on graph

Winning strategy acts as the software.
Reactive Synthesis: The Standard

Convert to a game on graph

Winning strategy acts as the software.

Find assumptions on the environment via the game graph.
The Return of Reactive Synthesis

Convert to a game on graph
The Return of Reactive Synthesis

Convert to a game on graph

Winning strategy under assumption acts as software

Compute assumptions

I will never block Roomba
The Return of Reactive Synthesis

Convert to a game on graph

Winning strategy under assumption acts as software

Suggest user

I will never block Roomba

Compute assumptions

Do not block me!
The Return of Reactive Synthesis

Do not block me!

Winning strategy
under assumption
acts as software

Suggest user

I will never block Roomba

Convert to a
game on graph

Compute assumptions
Precap

Assumptions computation
Precap

Novel Templates

Assumptions computation

Permissive
Complete
Precap

Assumptions computation

Novel Templates

- Permissive
- Complete
- Faster
Prepap

Assumptions computation

Novel Templates

Permissive
Complete
Faster
Games on Graphs
Games on Graphs
Games on Graphs
Always eventually visit \( h \)
Games on Graphs

Always eventually visit h

9
Games on Graphs

Always eventually visit h

$g \rightarrow i$
Games on Graphs

Always eventually visit \( h \)

\[ g \rightarrow i \rightarrow h \]
Games on Graphs

Always eventually visit h

$g \to i \to h \to h$
Games on Graphs

Always eventually visit $h$

$g \rightarrow i \rightarrow h \rightarrow h \rightarrow h$
Games on Graphs

Always eventually visit h

$g \rightarrow i \rightarrow h \rightarrow h \rightarrow h \rightarrow \ldots$
Games on Graphs

Always eventually visit $h$

$g \rightarrow i \rightarrow h \rightarrow h \rightarrow h \rightarrow \ldots$
Games on Graphs

Always eventually visit h

\[ g \rightarrow i \rightarrow h \rightarrow h \rightarrow h \rightarrow \ldots \]

Assumptions restrict the choices of Environment
Assumptions on Environment

$LTL$ formula $\psi$ on vertices of the game graph
Assumptions on Environment

LTL formula $\varphi$ on vertices of the game graph

Sufficient

If environment satisfies assumption, system can finish the task
Assumptions on Environment

LTL formula $\varphi$ on vertices of the game graph

Sufficient
If environment satisfies assumption, system can finish the task
Assumptions on Environment

LTL formula \( \varphi \) on vertices of the game graph

Sufficient
If environment satisfies assumption, system can finish the task

Implementable
Environment can satisfy the assumption
Assumptions on Environment

\[ \psi \] on vertices of the game graph

Sufficient
If environment satisfies assumption, system can finish the task

Implementable
Environment can satisfy the assumption

Leave the room
Assumptions on Environment

LTL formula \( \psi \) on vertices of the game graph

Sufficient
- If environment satisfies assumption, system can finish the task

Implementable
- Environment can satisfy the assumption

Permissive
- Assumption does not restrict the environment too much
Assumptions on Environment

LTL formula $\psi$ on vertices of the game graph

Sufficient
If environment satisfies assumption, system can finish the task

Implementable
Environment can satisfy the assumption

Permissive
Assumption does not restrict the environment too much
Assumptions on Environment

LTL formula \( \varphi \) on vertices of the game graph

Sufficient
If environment satisfies assumption, system can finish the task

Implementable
Environment can satisfy the assumption

Permissive
Assumption does not restrict the environment too much

\( \{ \text{Adequately Permissive Assumption} \} \)
Checkpoint

Assumptions computation

Permissive ✔️
Complete
Faster
Checkpoint

Assumptions computation

Novel Templates

Permissive Complete Faster
Computing Adequately Permissive Assumptions

Büchi objective: Always eventually visit
Büchi objective: Always eventually visit $\square$

No way of satisfying the objective from $c$. Hence, it must never be visited.
Computing Adequately Permissive Assumptions

Büchi objective: Always eventually visit □

No way of satisfying the objective from c. Hence, it must never be visited.

• Always not b→c.
Büchi objective: Always eventually visit

- Always not $b \rightarrow c$. 

Computing Adequately Permissive Assumptions

Büchi objective: Always eventually visit \( h \).

- Always not \( b \rightarrow c \).

Enough to compute assumptions to "reach" \( h \) from remaining vertices.
Computing Adequately Permissive Assumptions

Büchi objective: Always eventually visit $\square$

- Always not $b \rightarrow c$. 
Computing Adequately Permissive Assumptions

Büchi objective: Always eventually visit

- Always not b → c.
Computing Adequately Permissive Assumptions

Büchi objective: Always eventually visit

- Always not $b \rightarrow c$. 
Computing Adequately Permissive Assumptions

Büchi objective: Always eventually visit \( \square \)

- Always not \( b \rightarrow c \).
- Always eventually \( \{ f, g \} \Rightarrow \) always eventually \( f \rightarrow e \) or \( g \rightarrow i \)
Büchi objective: Always eventually visit

- Always not $b \rightarrow c$.
- Always eventually $\exists f, g \Rightarrow$ always eventually $f \rightarrow e$ or $g \rightarrow i$
Computing Adequately Permissive Assumptions

Büchi objective: Always eventually visit

- Always not $b \to c$.
- Always eventually $\exists\ f.g \Rightarrow$ always eventually $f \to e$ or $g \to i$
- Always eventually $\exists\ b \Rightarrow$ always eventually $b \to f$
Computing Adequately Permissive Assumptions

Büchi objective: Always eventually visit

- Always not $b \rightarrow c$.
- Always eventually $\exists f,g \Rightarrow$ always eventually $f \rightarrow e$ or $g \rightarrow i$
- Always eventually $\exists b \Rightarrow$ always eventually $b \rightarrow f$
Computing Adequately Permissive Assumptions

Büchi objective: Always eventually visit □

- Always not b→c.
- Always eventually \( \exists f.g \) ⇒ always eventually \( f→e \) or \( g→i \)
- Always eventually \( \exists b \) ⇒ always eventually \( b→f \)

→ Safety template

→ Group liveness templates
Computing Adequately Permissive Assumptions

Büchi objective: Always eventually visit \( \square \)

Runs in time \( O(m+n) \)

\# edges \# vertices

Safety template

- Always not \( b \rightarrow c \).
- Always eventually \( \Diamond f \cdot g \) \( \Rightarrow \) always eventually \( f \rightarrow e \) or \( g \rightarrow i \)
- Always eventually \( \Diamond b \) \( \Rightarrow \) always eventually \( b \rightarrow f \)

Group liveness templates
Computing Adequately Permissive Assumptions

cobüchi objective : Eventually always visit
Computing Adequately Permissive Assumptions

cobüchi objective : Eventually always visit $\square$

- Always not $b \rightarrow c$. 
Computing Adequately Permissive Assumptions

cOBüchi objective: Eventually always visit $\square$

Need to restrict from going further away from $h$ eventually.

- Always not $b \rightarrow c$. 
Computing Adequately Permissive Assumptions

coBüchi objective: Eventually always visit $h$

Need to restrict from going further away from $h$ eventually.

- Always not $b \rightarrow c$. 
Computing Adequately Permissive Assumptions

coBüchi objective: Eventually always visit \( h \)

Need to restrict from going further away from \( h \) eventually.

- Always not \( b \rightarrow c \).
- Eventually always not \( h \rightarrow f \).
Computing Adequately Permissive Assumptions

cOBüchi objective: Eventually always visit □

Need to restrict from going further away from h eventually.

- Always not b→c.
- Eventually always not h→f
coBüchi objective: Eventually always visit \( b \)

Need to restrict from going further away from \( h \) eventually.

- Always not \( b \rightarrow c \).
- Eventually always not \( h \rightarrow f \)
Computing Adequately Permissive Assumptions

cOBüchi objective: Eventually always visit

Need to restrict from going further away from eventually.

- Always not $b \rightarrow c$.
- Eventually always not $h \rightarrow f$
- Eventually always not $f \rightarrow g$
- Eventually always not $g \rightarrow f$
coBüchi objective: Eventually always visit $b$

- Always not $b \rightarrow c$.
- Eventually always not $h \rightarrow f$.
- Eventually always not $f \rightarrow g$.
- Eventually always not $g \rightarrow f$.

Need to restrict from going further away from $h$ eventually.
Computing Adequately Permissive Assumptions

cOBüchi objective: Eventually always visit $\blacksquare$

Need to restrict from going further away from $\blacksquare$ eventually.

- Always not $b \rightarrow c$.
- Eventually always not $h \rightarrow f$.
- Eventually always not $f \rightarrow g$.
- Eventually always not $g \rightarrow f$.
- Eventually always not $b \rightarrow a$. 
Computing Adequately Permissive Assumptions

cOBüchi objective: Eventually always visit

- Always not b→c.
- Eventually always not h→f
- Eventually always not f→g
- Eventually always not g→f
- Eventually always not b→a

Need to restrict from going further away from h eventually.
Computing Adequately Permissive Assumptions

cobuchi objective: Eventually always visit

- Always not $b \rightarrow c$.
- Eventually always not $h \rightarrow f$
- Eventually always not $f \rightarrow g$
- Eventually always not $g \rightarrow f$
- Eventually always not $b \rightarrow a$

Need to restrict from going further away from $h$ eventually.
Computing Adequately Permissive Assumptions

cOBüchi objective: Eventually always visit

Need to restrict from going further away from \( b \) eventually.

- Always not \( b \rightarrow c \).
- Eventually always not \( h \rightarrow f \).
- Eventually always not \( f \rightarrow g \).
- Eventually always not \( g \rightarrow f \).
- Eventually always not \( b \rightarrow a \).

Runs in time \( O(m+n) \)
Computing Adequately Permissive Assumptions

Parity objective: highest priority visited infinitely is even
Computing Adequately Permissive Assumptions

Parity objective: highest priority visited infinitely is even

Needs conditional group liveness templates

□ □ \( C_3 \Rightarrow \text{live group } (C_4 \cup C_6 \ldots) \)
Computing Adequately Permissive Assumptions

Parity objective: highest priority visited infinitely is even

Runs in time $O(n^4)$

Needs conditional group liveness templates

$\Box \Diamond c_3 \Rightarrow$ live group $(c_4 \lor c_6 \ldots)$
Checkpoint

Assumptions computation

Novel Templates

Permissive
Complete
Faster
Experiments

C++ Tool
SIMPA
Experiments

C++ Tool
SImPA

230 SYNTCOMP benchmarks
Experiments

* Chatterjee et al., CAV’10

C++ Tool
SImPA

230 SYNTCOMP benchmarks
Experiments

C++ Tool

SImPA

230 SYNTCOMP benchmarks

<table>
<thead>
<tr>
<th></th>
<th>SImPA</th>
<th>GIST*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean-time</td>
<td>64.8 s</td>
<td>1079.0 s</td>
</tr>
<tr>
<td>Non-timeout mean-time</td>
<td>64.8 s</td>
<td>209.2 s</td>
</tr>
<tr>
<td>Termination (1hr)</td>
<td>100%</td>
<td>74%</td>
</tr>
<tr>
<td>Generated assumptions</td>
<td>100%</td>
<td>91%</td>
</tr>
<tr>
<td>Fastest</td>
<td>100%</td>
<td>0%</td>
</tr>
</tbody>
</table>

* Chatterjee et. al., CAV’10
Experiments

![Graph showing relationship between SImPA and GIST](image)

<table>
<thead>
<tr>
<th>Feature</th>
<th>SImPA (s)</th>
<th>GIST* (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean-time</td>
<td>64.8 s</td>
<td>1079.0 s</td>
</tr>
<tr>
<td>Non-timeout mean-time</td>
<td>64.8 s</td>
<td>209.2 s</td>
</tr>
<tr>
<td>Termination (1hr)</td>
<td>100%</td>
<td>74%</td>
</tr>
<tr>
<td>Generated assumptions</td>
<td>100%</td>
<td>91%</td>
</tr>
<tr>
<td>Fastest</td>
<td>100%</td>
<td>0%</td>
</tr>
</tbody>
</table>

* Chatterjee et al., CAV’10

C++ Tool

SImpA

230 SYNTCOMP benchmarks

Always gives an assumption
Experiments

<table>
<thead>
<tr>
<th></th>
<th>SImPA</th>
<th>GIST*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean-time</td>
<td>64.8s</td>
<td>1079.0s</td>
</tr>
<tr>
<td>Non-timeout</td>
<td>64.8s</td>
<td>209.2s</td>
</tr>
<tr>
<td>mean-time</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Termination(1hr)</td>
<td>100%</td>
<td>74%</td>
</tr>
<tr>
<td>Generated</td>
<td>100%</td>
<td>91%</td>
</tr>
<tr>
<td>assumptions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fastest</td>
<td>100%</td>
<td>0%</td>
</tr>
</tbody>
</table>

* Chatterjee et al., CAV’10

C++ Tool
SImPA

230 SYNTCOMP benchmarks

Always gives an assumption

Orders of magnitude faster

Graph showing a scatter plot with axes labeled SImPA (s) and GIST (s) with data points and trend lines.
Summary

Novel Templates

Assumptions computation

Permissive
Complete
Faster
Assumptions on Environment

LTL formula \( \psi \) on vertices of the game graph

Sufficient: \( \forall s, e \in E \quad \psi \Rightarrow \exists s' \in s . \forall e', e \in e, s', e \text{ play } = \emptyset \)
Assumptions on Environment

LTL formula $\psi$ on vertices of the game graph

Sufficient: $v \in \ll s,e \gg \phi \Rightarrow \exists \pi_s \land \forall \pi_e, h(\pi_s, \pi_e, v) = \phi$

cooperative region

winning strategy

system strategy

environment strategy

play from $v$
Assumptions on Environment

LTL formula $\psi$ on vertices of the game graph

Sufficient: $v \in \{s,e\} \varphi \Rightarrow \exists \pi_s \forall \pi_e \; h(\pi_s, \pi_e, v) = \varphi$

- cooperative region
- winning strategy
- system strategy
- environment strategy
- $\pi_s, \pi_e$ play from $v$
Assumptions on Environment

LTL formula \( \psi \) on vertices of the game graph

Sufficient: \( v \in \llangle s,e \rrangle \phi \Rightarrow \exists \pi_s . \forall \pi_e , \llangle \pi_s \pi_e , v \rrangle = \psi \)

Implementable: \( \exists \pi_e . \forall \pi_s . \llangle \pi_s \pi_e \rrangle = \psi \)
Assumptions on Environment

LTL formula $\psi$ on vertices of the game graph

Sufficient: $v \in \llbracket s,e \rrbracket \phi \Rightarrow \exists s . \forall e . \psi \land (\llbracket s,e \rrbracket , v) = \phi$

Implementable: $\exists e . \forall s . \psi \land (\llbracket s,e \rrbracket ) = \psi$

Environment has a strategy to satisfy $\psi$ from every vertex
Assumptions on Environment

LTL formula $\psi$ on vertices of the game graph

Sufficient: $v \in \langle s, e \rangle \Phi \Rightarrow \exists \tau_s \cdot \forall \tau_e \cdot L(\tau_s \tau_e, v) = \psi$

Implementable: $\exists \tau_e \cdot \forall \tau_s \cdot L(\tau_s \tau_e) = \psi$

Permissive: $L(\Phi) \subseteq L(\psi)$
Assumptions on Environment

LTL formula \( \psi \) on vertices of the game graph

Sufficient: \( \forall v \in \langle s, e \rangle \phi \Rightarrow \exists \pi_s \cdot \forall \pi_e, L(\pi_s \pi_e, v) = \psi \)

Implementable: \( \exists \pi_e \cdot \forall \pi_s, L(\pi_s \pi_e) = \psi \)

Permissive: \( L(\phi) \subseteq L(\psi) \)

Adequately permissive assumption