GenSys - A Scalable Fixed-Point Engine for Maximal Controller Synthesis over Infinite State Logical Games

Stanly Samuel[1], Deepak D’Souza[1], Raghavan Komondoor[1]

Accepted at ESEC/FSE 2021: Demonstrations Track

[1]Indian Institute of Science, Bangalore, India

Background: Synthesis

Church's problem

- Stated by Alonzo Church in 1957:

  Given a requirement which a circuit is to satisfy, we may suppose the requirement expressed in some suitable logistic system which is an extension of restricted recursive arithmetic. The synthesis problem is then to find recursion equivalences representing a circuit that satisfies the given requirement (or alternatively, to determine that there is no such circuit)

- The requirement taken as transformation from one infinite bit string to another
  - Transforming every α to β such that $\phi(\alpha, \beta)$ holds
  - Transformations expected to be nonaticipatory and computable with finite memory
  - Finite state automata replaces logical circuits

Alonzo Church

APPLICATION OF RECURSIVE ARITHMETIC TO THE PROBLEM OF CIRCUIT SYNTHESIS

by Alonzo Church

A paper presented at the Summer Institute of Symbolic Logic at Ithaca, N. Y. , in July, 1957 - with revisions made in August , 1957.

Background: Synthesis

1957 to early 2000's:

- Largely of theoretical interest.
- Highly intractable due to the large state space. Complexity can be doubly exponential in nature.
- Undecidable due to infinite state space.

21st century:

- Several tools broach the surface of practicality and show empirically efficient solutions.

Alonzo Church

APPLICATION OF RECURSIVE ARITHMETIC TO THE
PROBLEM OF CIRCUIT SYNTHESIS

by Alonzo Church

A paper presented at the Summer Institute of Symbolic Logic at Ithaca, N. Y. , in July, 1957 - with revisions made in August , 1957.

Background: Reactive Synthesis

Temporal Specification
(Declarative)

Synthesizer

Controller
(Reactive entity)

Background: Reactive Synthesis

Temporal Specification
(Declarative)

Synthesizer

Controller
(Reactive entity)

E.g.: A simple bus arbiter

Synthesizer

r → **Controller Logic ?** → **g**

Background: Reactive Synthesis

Temporal Specification
(Declarative)

Synthesizer

Controller
(Reactive entity)

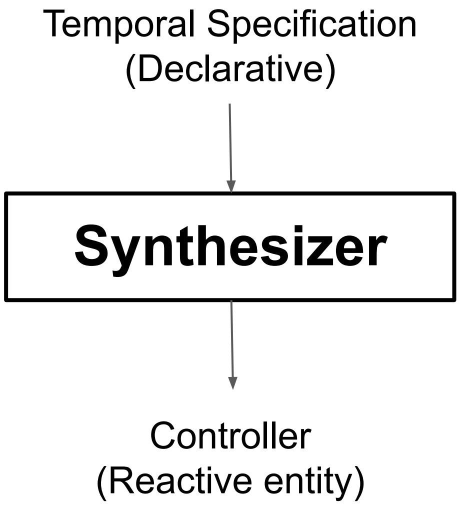
E.g.: A simple bus arbiter

"Whenever **r** is true,
eventually **g** must be true"

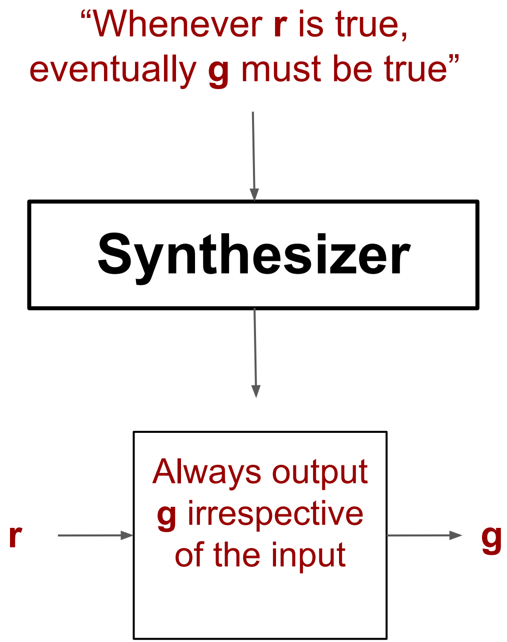
Synthesizer

r → **Controller Logic ?** → **g**

Background: Reactive Synthesis

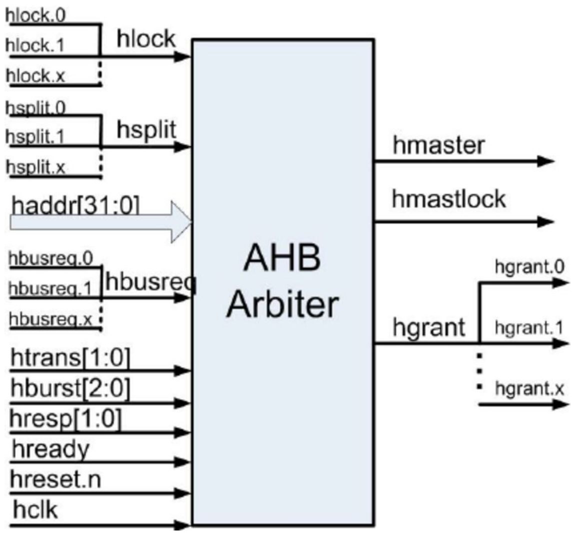


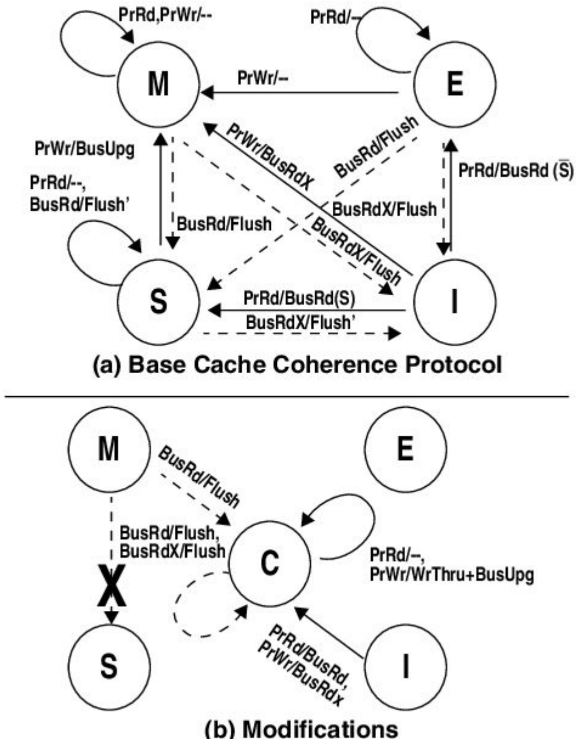
E.g.: A simple bus arbiter



Practical Motivation: Reactive Synthesis

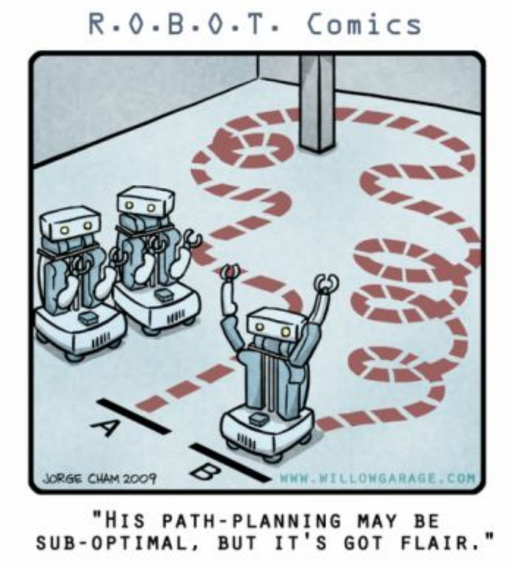
- Finite state systems:
  - AMBA Bus Arbiter
  - Cache Coherence Protocols

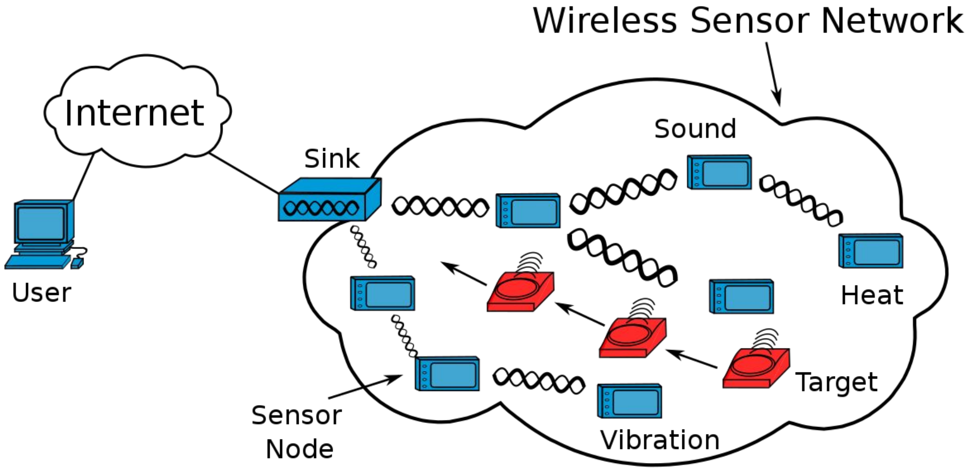




Practical Motivation: Reactive Synthesis

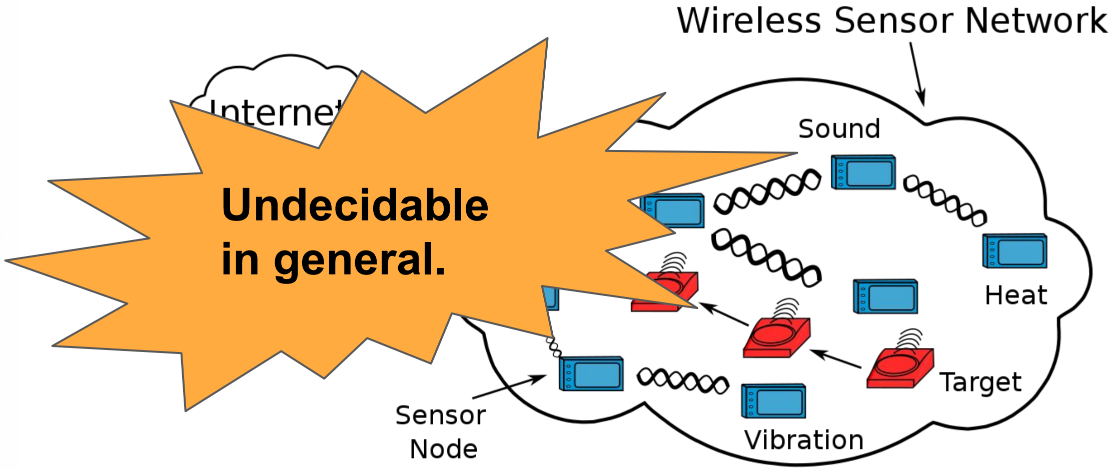
- **Infinite state systems:**
  - **Robot Motion Planning**
  - **Minimum Backlog Problem in Wireless Sensor Networks**





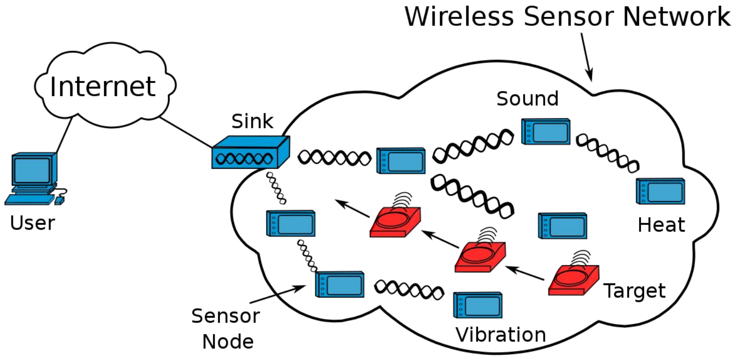
Practical Motivation: Reactive Synthesis

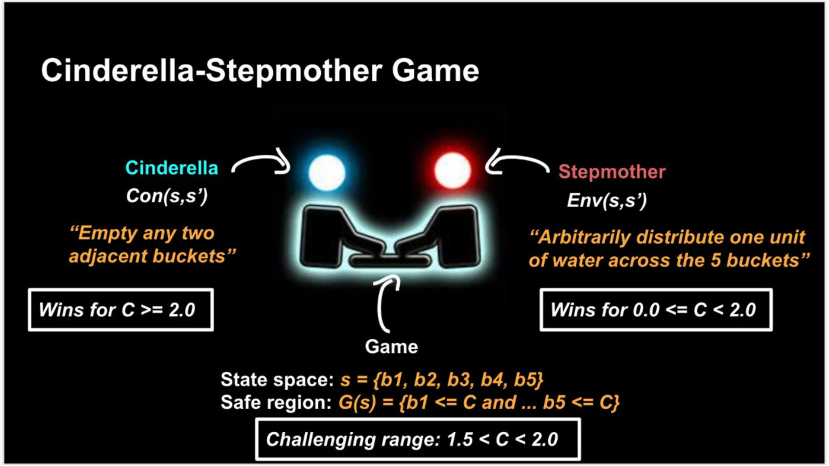
- **Infinite state systems:**
  - **Robot Motion Planning**
  - **Minimum Backlog Problem in Wireless Sensor Networks**



Minimum Backlog Problem in Wireless Sensor Networks

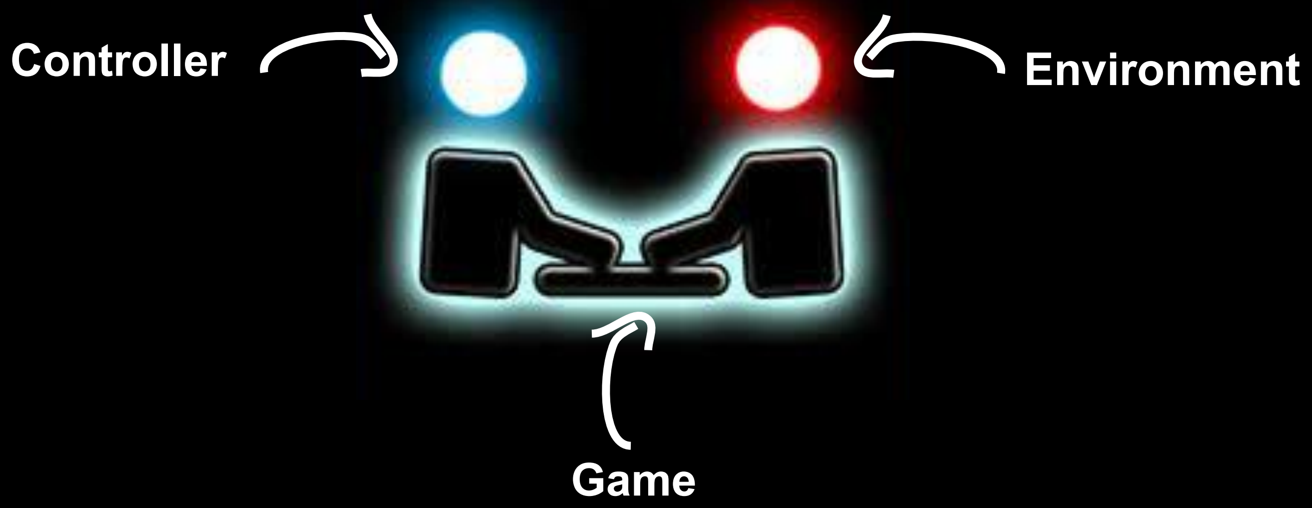
Abstraction



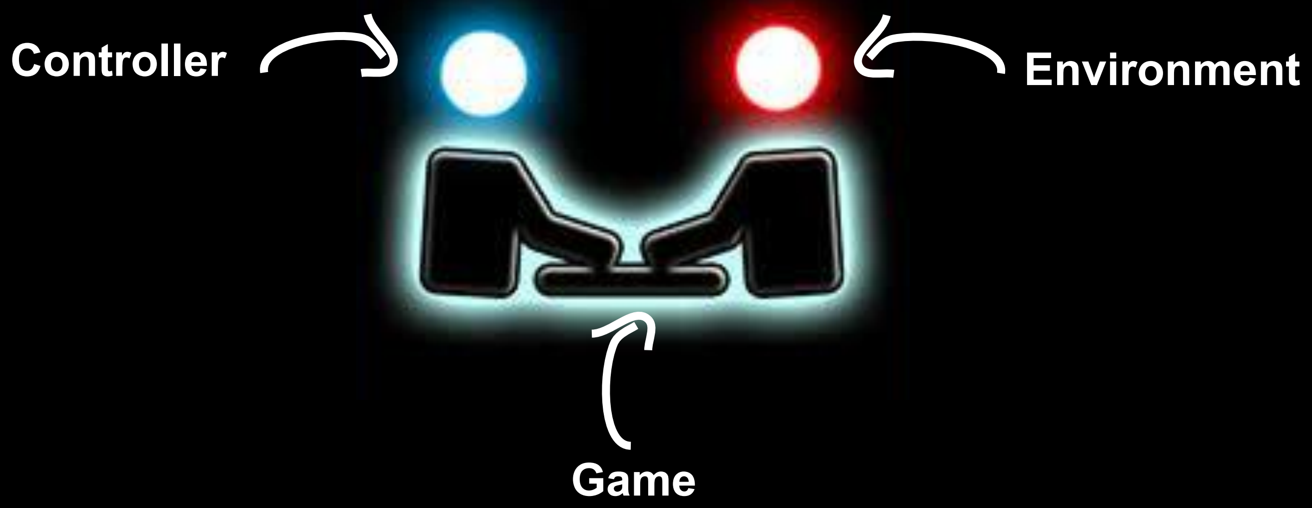


GenSys - Synthesize a maximal controller that adheres to a given temporal specification over a system with an infinite state space.

A Two Player Infinite Game over an Infinite State Space



A Two Player Infinite Game over an Infinite State Space



State space: *s*

A Two Player Infinite Game over an Infinite State Space

Controller

Environment

Game

State space: *s* Safe region: *G(s)*

A Two Player Infinite Game over an Infinite State Space

Controller

Con(s,s')

Environment

Env(s,s')

Game

State space: *s* Safe region: *G(s)*

A Two Player Infinite Game over an Infinite State Space

Controller

Con(s,s')

Environment

Env(s,s')

Game

State space: *s* Safe region: *G(s)*

Play of the game: $s \rightarrow s' \rightarrow s'' \rightarrow s''' \;\ldots$

A Two Player Infinite Game over an Infinite State Space

Controller

Con(s,s')

Environment

Env(s,s')

Game

State space: *s* Safe region: *G(s)*

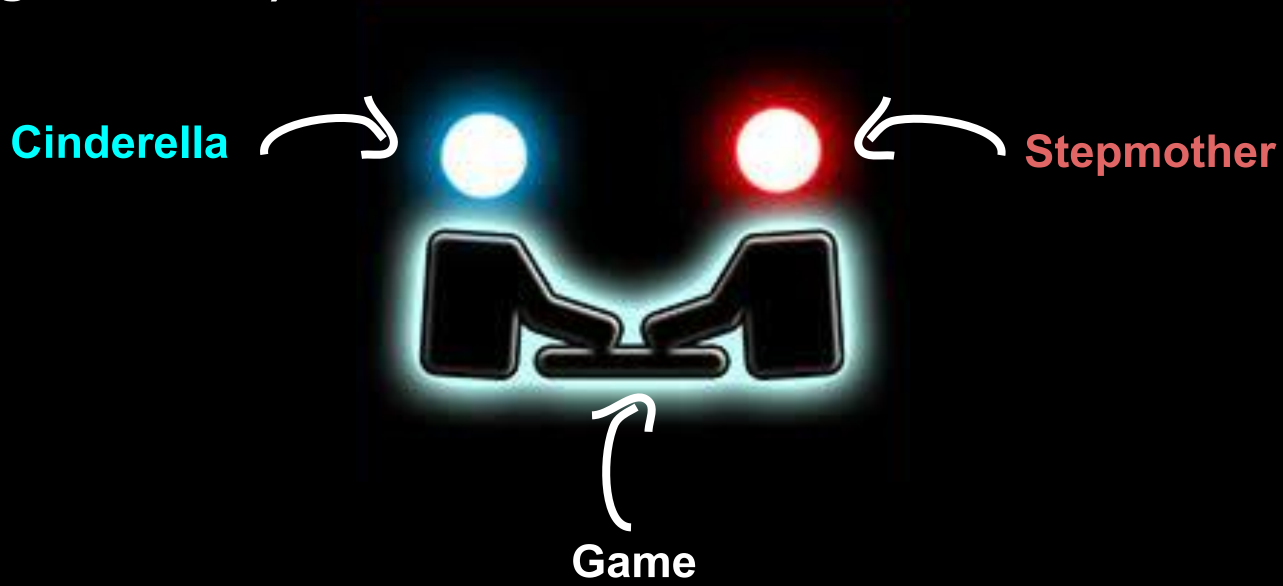
Play of the game: *s* → *s'* → *s''* → *s'''* …

Winning Condition:
<u>Safety</u>

Synthesize

1. Winning region W(s).
2. Strategy for Controller

Cinderella-Stepmother Game (Abstraction of the Minimum Backlog Problem)



Cinderella-Stepmother Game

Cinderella

Con(s,s')

Stepmother

Env(s,s')

Game

Cinderella-Stepmother Game

Cinderella
Con(s,s')

Stepmother
Env(s,s')

Game

State space: $s = \{b1, b2, b3, b4, b5\}$

Cinderella-Stepmother Game

Cinderella
Con(s,s')

Stepmother
Env(s,s')

Game

State space: $s = \{b1, b2, b3, b4, b5\}$

b2
b1
b3
b5
b4

Cinderella-Stepmother Game

Cinderella

Con(s,s')

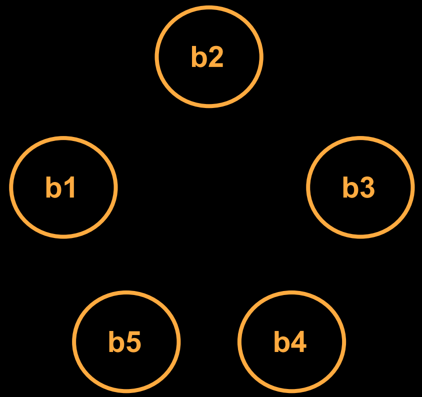
"Empty any two adjacent buckets"

Stepmother

Env(s,s')

Game

State space: ***s = {b1, b2, b3, b4, b5}***



Cinderella-Stepmother Game

Cinderella

Con(s,s')

"Empty any two adjacent buckets"

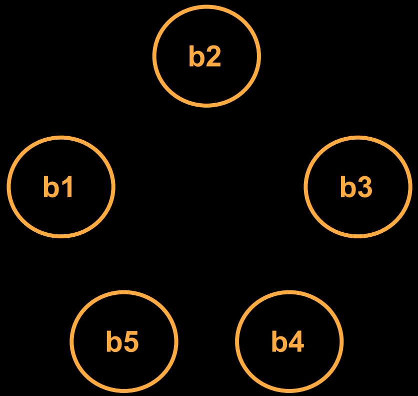
Stepmother

Env(s,s')

"Arbitrarily distribute one unit of water across the 5 buckets"

Game

State space: *s = {b1, b2, b3, b4, b5}*



Cinderella-Stepmother Game

Cinderella

Con(s,s')

"Empty any two adjacent buckets"

Stepmother

Env(s,s')

"Arbitrarily distribute one unit of water across the 5 buckets"

Game

State space: *s = {b1, b2, b3, b4, b5}*

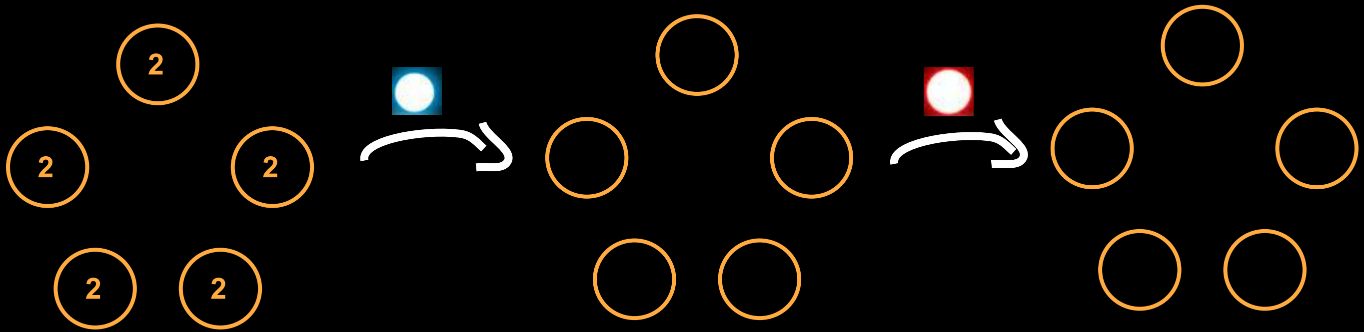
Safe region: *G(s) = {b1 <= C and ... b5 <= C}*

Cinderella-Stepmother Game

Play of the game:

Example: Let bucket overflow limit C be 3.

Can Cinderella ALWAYS win against any move of the Environment?

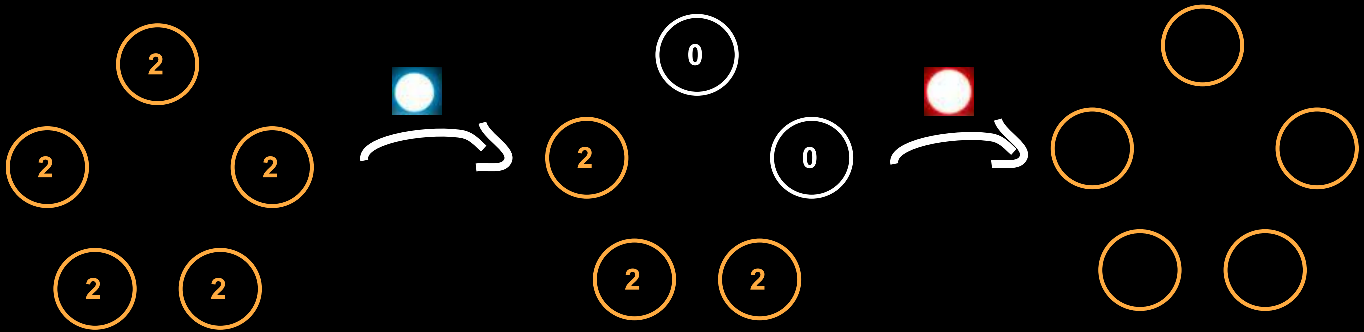


Cinderella-Stepmother Game

Play of the game: $s \rightarrow s' \rightarrow s'' \rightarrow s''' \;\dots$

Example: Let bucket overflow limit C be 3.

Can Cinderella ALWAYS win against any move of the Environment?

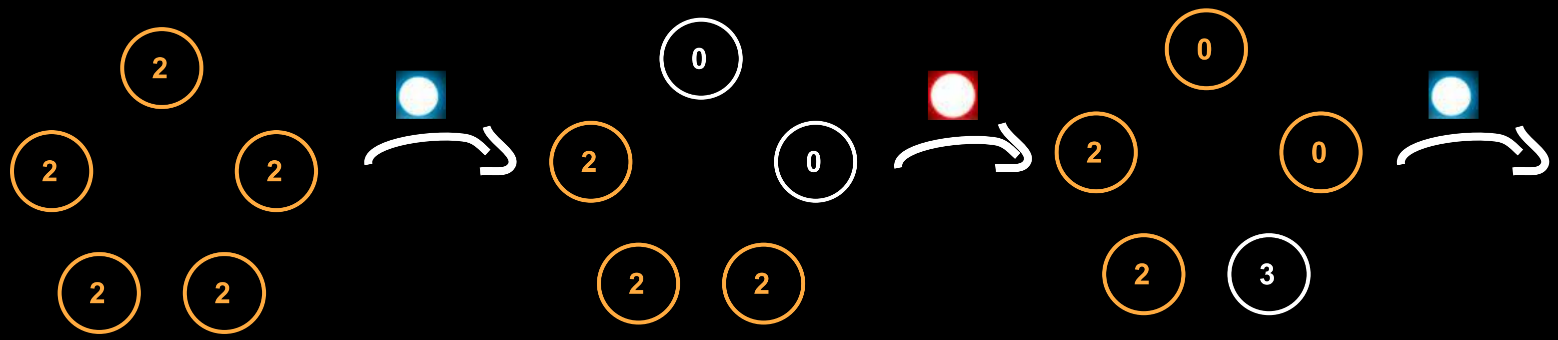


Cinderella-Stepmother Game

Play of the game: $s \rightarrow s' \rightarrow s'' \rightarrow s''' \;\ldots$

Example: Let bucket overflow limit C be 3.

Can Cinderella ALWAYS win against any move of the Environment?

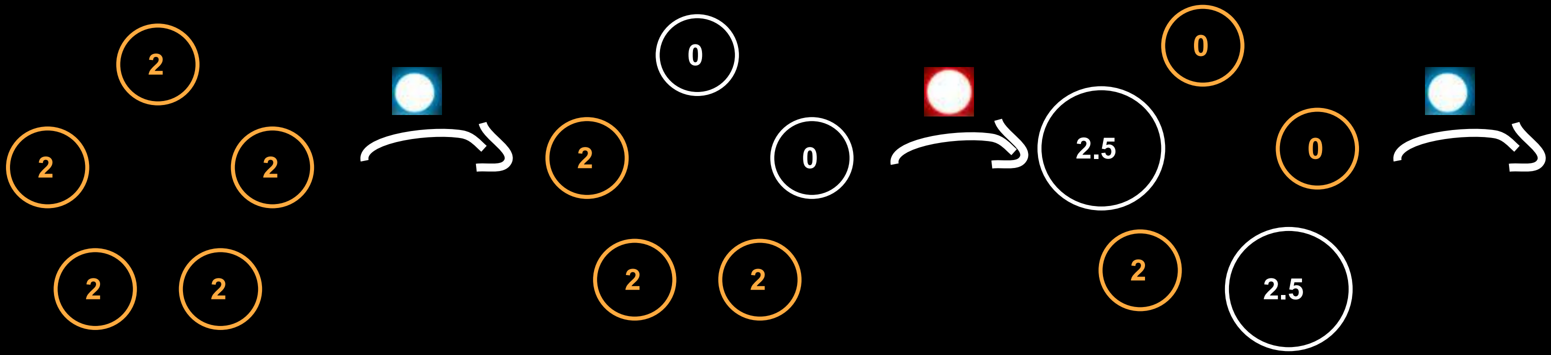


Cinderella-Stepmother Game

Play of the game: $s \rightarrow s' \rightarrow s'' \rightarrow s''' \ \ldots$

Example: Let bucket overflow limit C be 3.

Can Cinderella ALWAYS win against any move of the Environment?



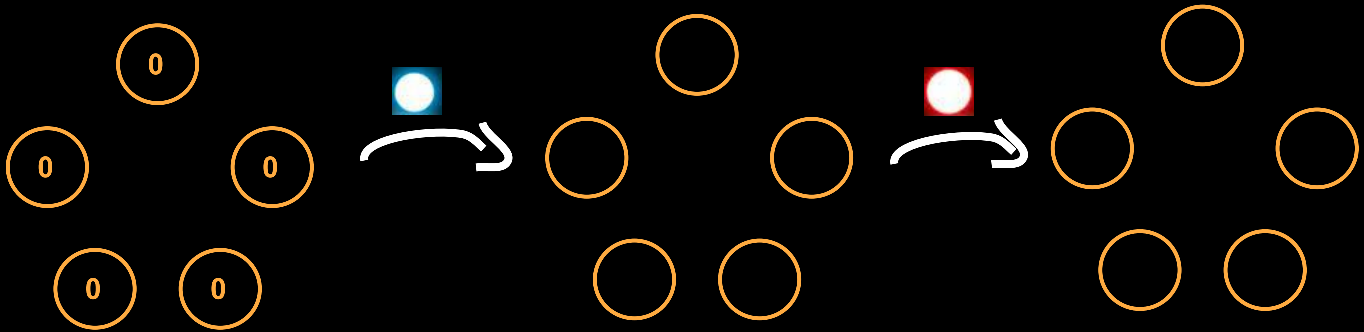
For C =3, {2,2,2,2,2} <u>will not</u> be included in W(s) as there exists a strategy for environment to win!

Cinderella-Stepmother Game

Play of the game: $s \rightarrow s' \rightarrow s'' \rightarrow s''' \ \ldots$

Example: Let bucket overflow limit C be 3.

Can Cinderella ALWAYS win against any move of the Environment?



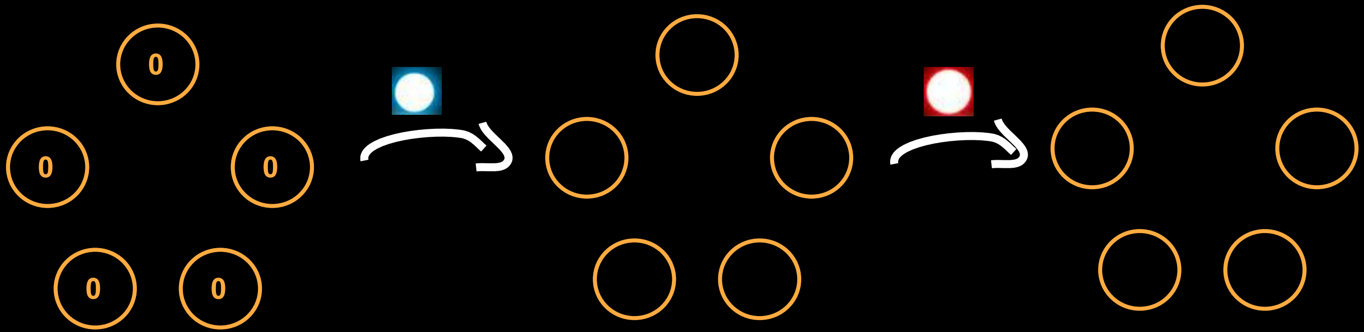
For C =3, {0,0,0,0,0} <u>will</u> be included in W(s) as there exists NO strategy for environment to win!

Cinderella-Stepmother Game

Play of the game: $s \to s' \to s'' \to s''' \ \dots$

Example: Let bucket overflow limit C be <2.

Can Cinderella ALWAYS win against any move of the Environment?



For C<2, Cinderella can never win! For any starting state!

Cinderella-Stepmother Game

Cinderella

Con(s,s')

"Empty any two adjacent buckets"

Stepmother

Env(s,s')

"Arbitrarily distribute one unit of water across the 5 buckets"

Wins for C >= 2.0

Wins for 0.0 <= C < 2.0

Game

State space: *s = {b1, b2, b3, b4, b5}*
Safe region: *G(s) = {b1 <= C and ... b5 <= C}*

Cinderella-Stepmother Game

Cinderella

Con(s,s')

"Empty any two adjacent buckets"

Stepmother

Env(s,s')

"Arbitrarily distribute one unit of water across the 5 buckets"

Wins for C >= 2.0

Wins for 0.0 <= C < 2.0

Game

State space: *s = {b1, b2, b3, b4, b5}*

Safe region: *G(s) = {b1 <= C and ... b5 <= C}*

Challenging range: 1.5 < C < 2.0

GenSys: Tool Architecture

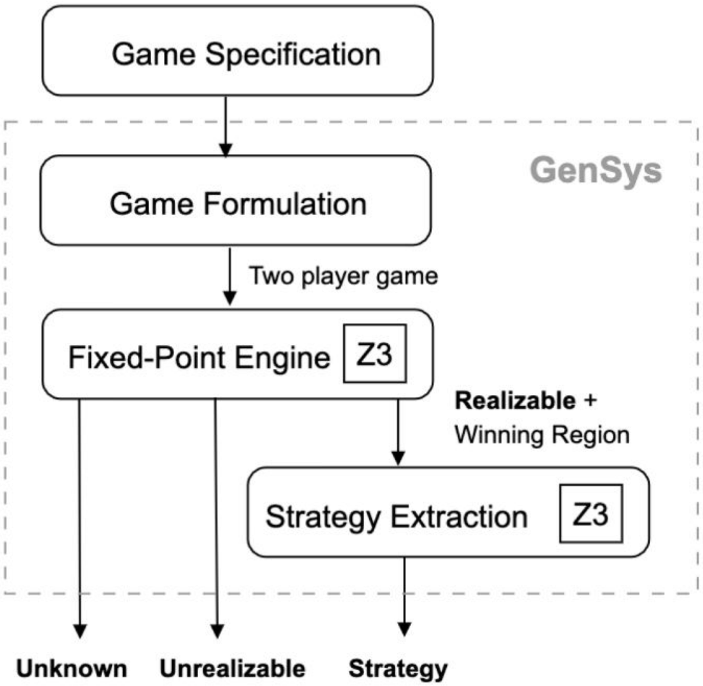


Figure 1: GenSys Tool Architecture

GenSys: Cinderella Game Specification

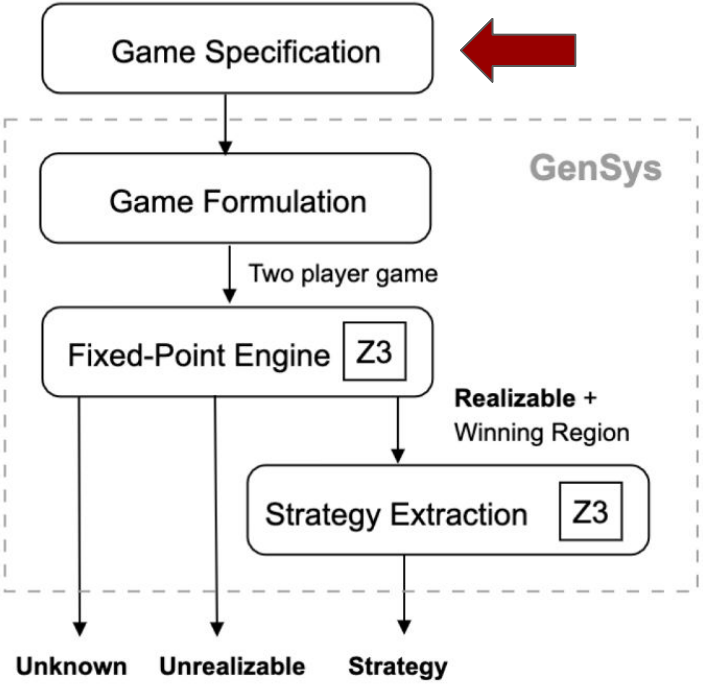


Figure 1: GenSys Tool Architecture

```
from  gensys.helper import *
from  gensys.fixpoints import *
from z3 import *

#1. Environment moves
def environment(b1, b2, b3, b1_, b2_, b3_):
  return And(b1_ + b2_ + b3_  == b1 + b2 + b3  + 1,
     b1_>=b1, b2_>=b2, b3_>=b3)

#2.  Controller moves
def move1(b1, b2, b3, b1_, b2_, b3_):
  return And(b1_ == 0, b2_ == 0, b3_ == b3)

def move2(b1, b2, b3,  b1_, b2_, b3_):
    return And( b2_ == 0, b3_ == 0, b1_ == b1)

def move3(b1, b2, b3, b1_, b2_, b3_):
    return And( b3_ == 0,  b1_ == 0, b2_ == b2)

controller_moves = [move1, move2, move3]

#3. Safe set
C = sys.argv[1]

def guarantee(b1, b2, b3):
    return And(b1 <= C, b2 <= C, b3 <= C, b1 >= 0, b2
      >= 0, b3 >= 0)

safety_fixedpoint(controller_moves,   environment,
     guarantee)
```

Figure 2: Cinderella Game Specification in GenSys

GenSys: Core Computation Engine

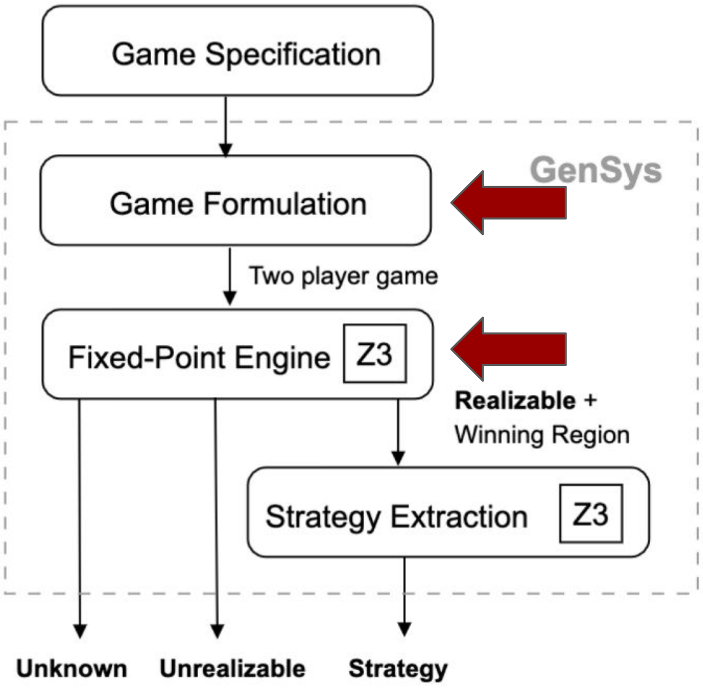


Figure 1: GenSys Tool Architecture

3.2 Game Formulation

From the given game specification, this module of our tool formulates one step of the game. The formulation is as follows:

$$
\begin{aligned}
WP(X) \quad \equiv \quad & \exists s'.(Con(s,s') \;\; \wedge G(s') \;\; \wedge \\
& \forall s''.(Env(s',s'') \implies (G(s'') \wedge X(s''))))
\end{aligned}
$$

A step consists of a move of the controller followed by a move of the environment. The formula above has the state variable s as the free variable. The solution to this formula is the set of states starting from which the controller has a move such that if the environment subsequently makes a move, both moves end in a state that satisfies the given winning condition G, and the environment's move ends in a state that is in a given set of states X. The formula above resembles the weakest pre-condition computation in programming languages. Note that the controller makes the first move [2].

3.3 Fixed-Point Engine

The *winning region* of the game is the solution to the following greatest fixed-point equation:

$$\nu X.\ WP(X)$$

GenSys: Winning Region

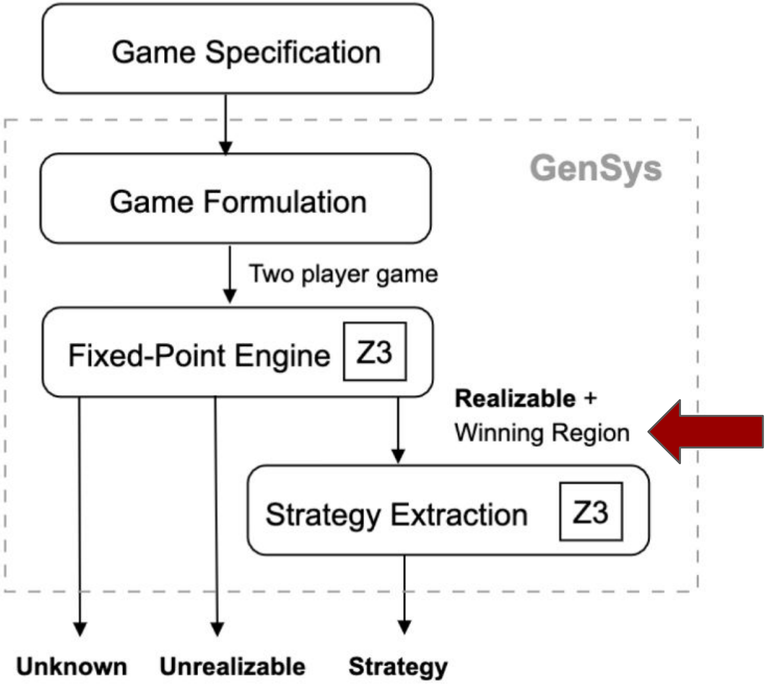


Figure 1: GenSys Tool Architecture

$$
\begin{array}{l}
0 \le b_1, b_2 \le 3 \wedge 0 \le b_3, b_4, b_5 \le 2 \wedge b_3 + b_5 \le 3 \quad \vee \\
0 \le b_2, b_3 \le 3 \wedge 0 \le b_4, b_5, b_1 \le 2 \wedge b_4 + b_1 \le 3 \quad \vee \\
0 \le b_3, b_4 \le 3 \wedge 0 \le b_5, b_1, b_2 \le 2 \wedge b_5 + b_2 \le 3 \quad \vee \\
0 \le b_4, b_5 \le 3 \wedge 0 \le b_1, b_2, b_3 \le 2 \wedge b_1 + b_3 \le 3 \quad \vee \\
0 \le b_5, b_1 \le 3 \wedge 0 \le b_2, b_3, b_4 \le 2 \wedge b_2 + b_4 \le 3 \quad \vee
\end{array}
$$

GenSys: Extracted Controller

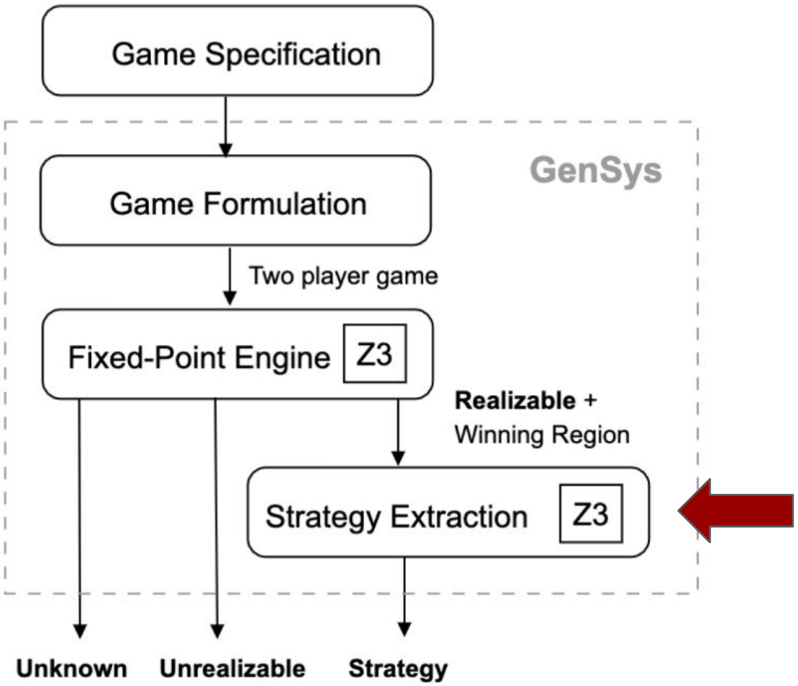


Figure 1: GenSys Tool Architecture

Table 1: Strategy Synthesized by GenSys for the Cindrella game with bucket size 3

Condition	Move
$0 \leq b_1, b_2 \leq 3 \wedge 0 \leq b_3, b_4, b_5 \leq 2 \wedge b_3 + b_5 \leq 3$	$b_1\_, b_2\_ = 0$
$0 \leq b_2, b_3 \leq 3 \wedge 0 \leq b_4, b_5, b_1 \leq 2 \wedge b_4 + b_1 \leq 3$	$b_2\_, b_3\_ = 0$
$0 \leq b_3, b_4 \leq 3 \wedge 0 \leq b_5, b_1, b_2 \leq 2 \wedge b_5 + b_2 \leq 3$	$b_3\_, b_4\_ = 0$
$0 \leq b_4, b_5 \leq 3 \wedge 0 \leq b_1, b_2, b_3 \leq 2 \wedge b_1 + b_3 \leq 3$	$b_4\_, b_5\_ = 0$
$0 \leq b_5, b_1 \leq 3 \wedge 0 \leq b_2, b_3, b_4 \leq 2 \wedge b_2 + b_4 \leq 3$	$b_5\_, b_1\_ = 0$

GenSys: Extracted Strategy

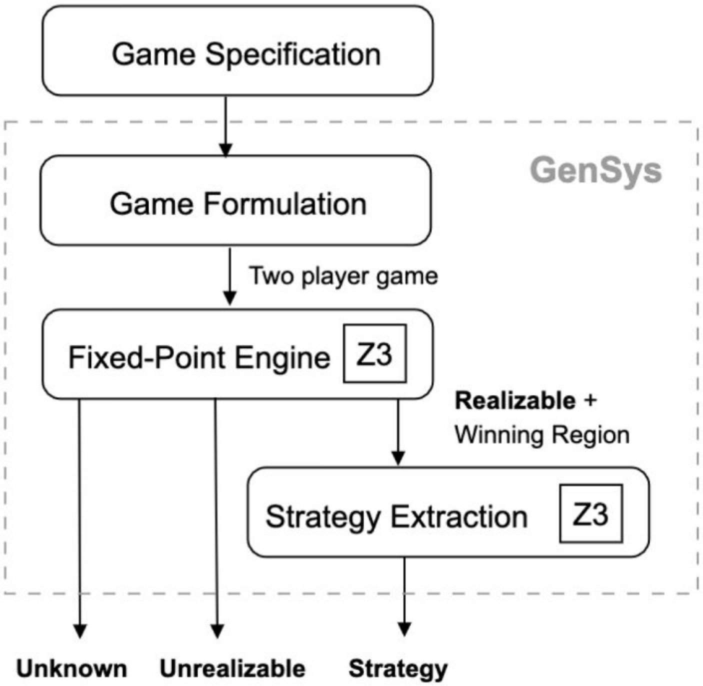


Figure 1: GenSys Tool Architecture

Table 1: Strategy Synthesized by GenSys for the Cindrella game with bucket size 3

Condition	Move
$0 \leq b_1, b_2 \leq 3 \wedge 0 \leq b_3, b_4, b_5 \leq 2 \wedge b_3 + b_5 \leq 3$	$b_1\_, b_2\_ = 0$
$0 \leq b_2, b_3 \leq 3 \wedge 0 \leq b_4, b_5, b_1 \leq 2 \wedge b_4 + b_1 \leq 3$	$b_2\_, b_3\_ = 0$
$0 \leq b_3, b_4 \leq 3 \wedge 0 \leq b_5, b_1, b_2 \leq 2 \wedge b_5 + b_2 \leq 3$	$b_3\_, b_4\_ = 0$
$0 \leq b_4, b_5 \leq 3 \wedge 0 \leq b_1, b_2, b_3 \leq 2 \wedge b_1 + b_3 \leq 3$	$b_4\_, b_5\_ = 0$
$0 \leq b_5, b_1 \leq 3 \wedge 0 \leq b_2, b_3, b_4 \leq 2 \wedge b_2 + b_4 \leq 3$	$b_5\_, b_1\_ = 0$

Theorem: GenSys is guaranteed to synthesize a **sound** and **maximal** controller, if it terminates.

Soundness: Controller can never lose starting from the states in the strategy.

Maximality: No states from where the controller can win upon initiation, is missed.

GenSys: Maximality guarantee

3.2 Game Formulation

From the given game specification, this module of our tool formulates one step of the game. The formulation is as follows:

$$\begin{aligned} WP(X) \quad \equiv \quad & \exists s'.(Con(s,s') \;\wedge G(s') \;\wedge \\ & \forall s''.(Env(s',s'') \implies (G(s'') \wedge X(s'')))) \end{aligned}$$

A step consists of a move of the controller followed by a move of the environment. The formula above has the state variable s as the free variable. The solution to this formula is the set of states starting from which the controller has a move such that if the environment subsequently makes a move, both moves end in a state that satisfies the given winning condition G, and the environment's move ends in a state that is in a given set of states X. The formula above resembles the weakest pre-condition computation in programming languages. Note that the controller makes the first move [2].

3.3 Fixed-Point Engine

The *winning region* of the game is the solution to the following greatest fixed-point equation:

$$\nu X.\ WP(X)$$

8 APPENDIX

8.1 Safety Algorithm

Algorithm 2 computes the greatest solution to the equation in Section 3.2.

Algorithm 1: Safety fixed-point

Input : Game formulation WP, which includes the safe region G

Output : Winning region X, if algorithm terminates

X := True ;

W := $Proj(WP(X))$;

while $(X \wedge G) \not\Rightarrow (W \wedge G)$ **do**

  X := W;

  W := $Proj(WP(X))$

end

return $(X \wedge G)$;

Experimental Evaluation

Table 2: Running times for the Cinderella game for various values of bucket size C. "-" indicates unavailability of data, while ">xm" denotes a timeout after x minutes. R denotes Realizable and U denotes Unrealizable.

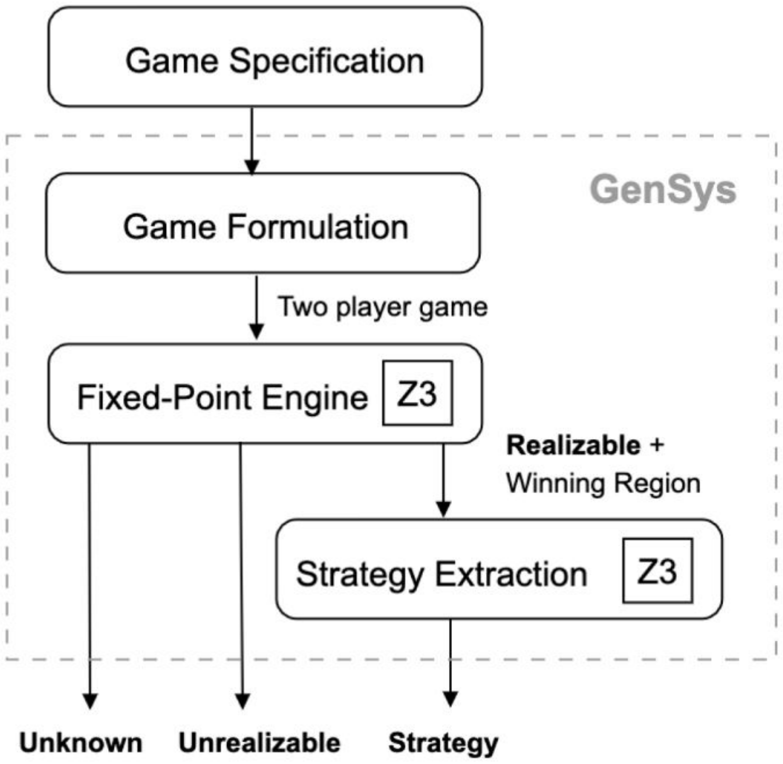
C	Out	SimSynth	ConSynth	JSyn-VG	GenSys	
					Time	Iter
3.0	R	2.2s	12m45s	1m26s	0.6s	3
2.5	R	53.8s	**>15m**	1m19s	0.7s	3
2.0	R	68.9s	-	1m6s	0.6s	3
1.9(20)	U	-	-	**>16m**	31.0s	69
1.8	U	**>10m**	-	**>16m**	0.6s	5
1.6	U	1.5s	-	**>16m**	0.4s	4
1.5	U	1.4s	-	14m34s	0.3s	4
1.4	U	0.2s	-	17s	0.2s	3

Experimental Evaluation

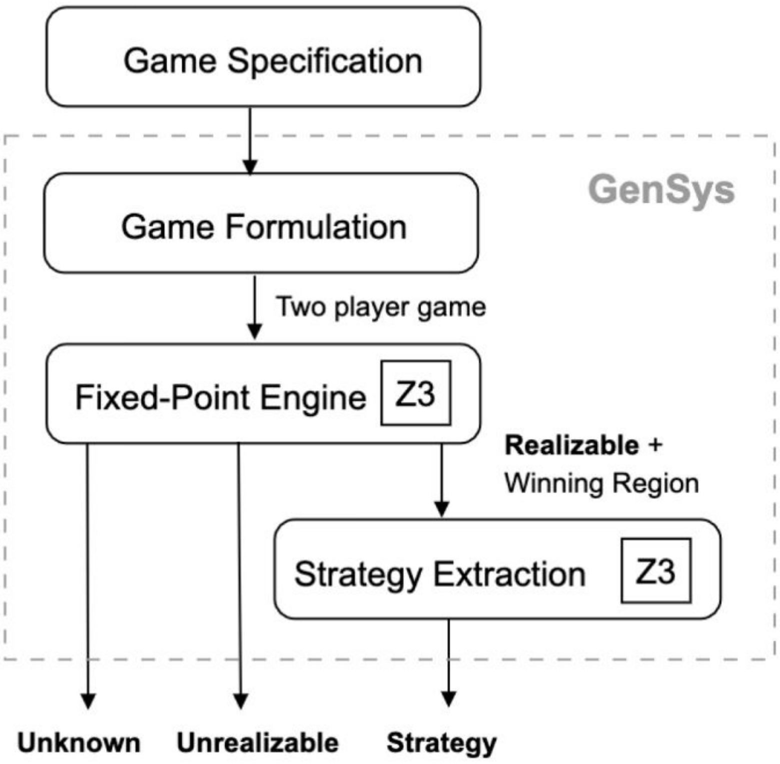
Table 3: Results on remaining benchmarks. Times are in seconds. >15m denotes a timeout after 15 minutes. Tool name abbreviations: C for ConSynth, J for JSyn-VG, D for DT-Synth, S for SAT-Synth, R for RPI-Synth, G for GenSys.

Benchmark	C	J	D	S	R	G
Repair-Lock	2.5	1.5	0.5	0.6	0.2	0.3
Box	3.7	0.6	0.3	0.3	0.1	0.3
Box Limited	0.4	1.7	0.1	0.4	0.5	0.2
Diagonal	1.9	4.0	2.4	1.34	0.5	0.2
Evasion	1.5	0.5	0.2	81	0.1	0.7
Follow	**>15m**	1.2	0.3	88.9	**>15m**	0.7
Solitary Box	0.4	0.9	0.1	0.3	0.1	0.3
Square 5x5	**>15m**	6.5	2.5	0.6	0.2	0.3

Work in Progress: GenSys - ω (GenSys for ω-regular specifications, including LTL)

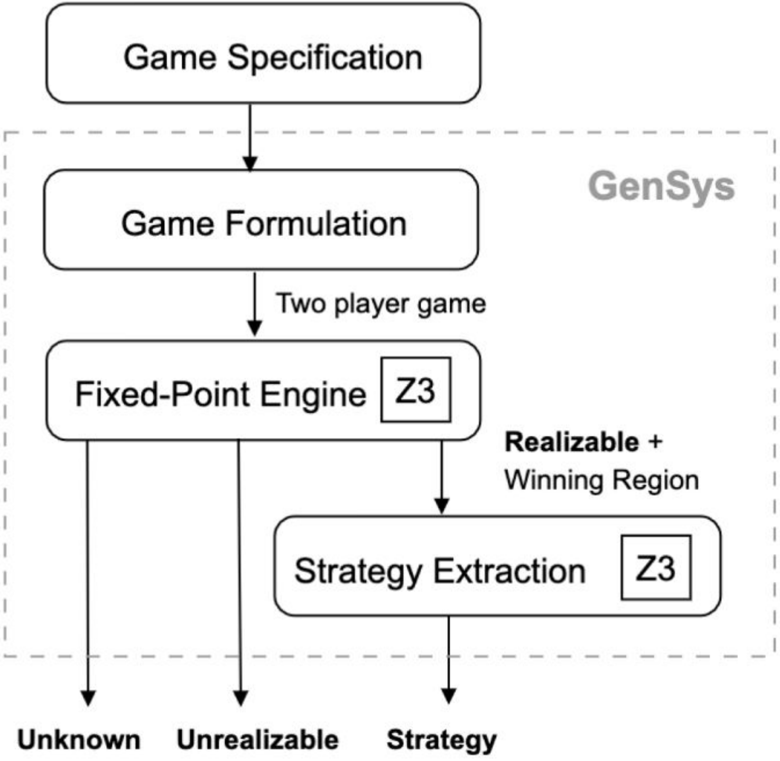


Work in Progress: GenSys - ω (GenSys for ω-regular specifications, including LTL)



Includes extension for Linear Temporal Logic or Universal Co-Buchi Automaton specifications.
E.g. "buckets do not overflow infinitely often"

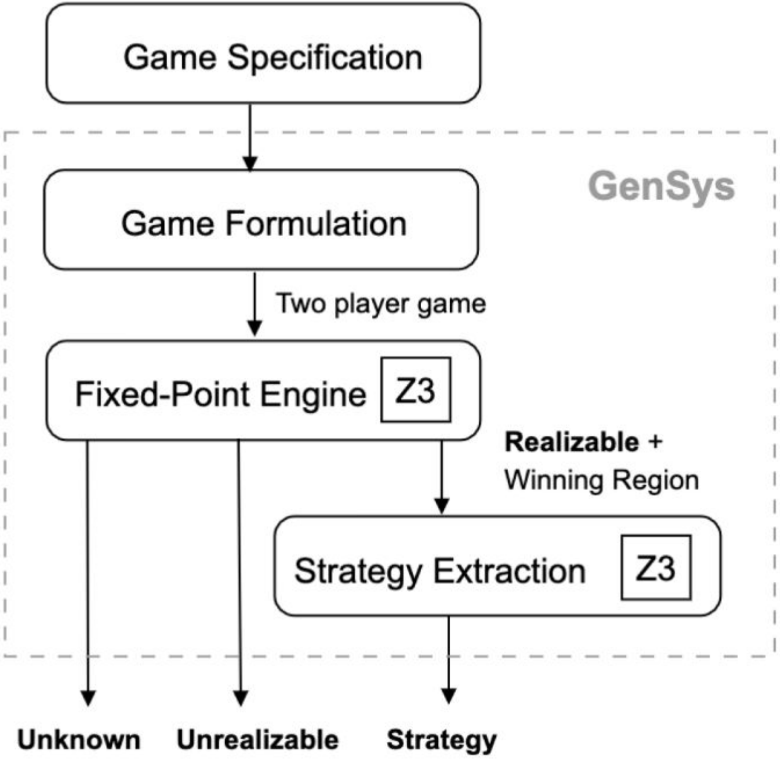
Work in Progress: GenSys - ω (GenSys for ω-regular specifications, including LTL)



Includes extension for Linear Temporal Logic or Universal Co-Buchi Automaton specifications.
E.g. “buckets do not overflow infinitely often”

Extends **WP(X)** with a novel On The Fly (OTF) Determinization Technique baked into the Fixed-Point computation process.

Work in Progress: GenSys - ω (GenSys for ω-regular specifications, including LTL)

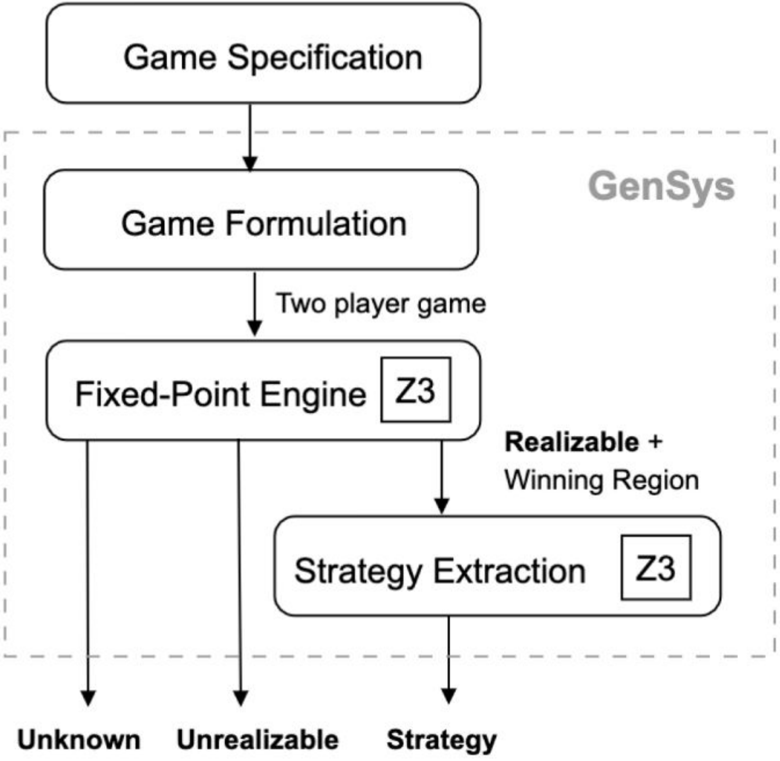


Includes extension for Linear Temporal Logic or Universal Co-Buchi Automaton specifications. **E.g. “buckets do not overflow infinitely often”**

Extends **WP(X)** with a novel On The Fly (OTF) Determinization Technique baked into the Fixed-Point computation process.

Generates a memory-full strategy in terms of the game states which is k-maximal. Procedure is sound and incomplete, as before.

Work in Progress: GenSys - ω (GenSys for ω-regular specifications, including LTL)



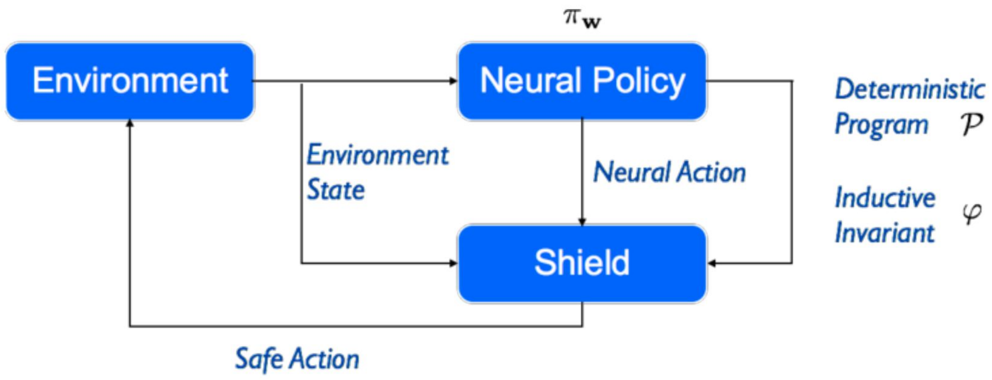
Includes extension for Linear Temporal Logic or Universal Co-Buchi Automaton specifications. **E.g. "buckets do not overflow infinitely often"**

Extends **WP(X)** with a novel On The Fly (OTF) Determinization Technique baked into the Fixed-Point computation process.

Generates a memory-full strategy in terms of the game states which is k-maximal. Procedure is sound and incomplete, as before.

Current Challenge: Scalability due to the inherent dependency on Z3's projection operator.

Why do we care about **Maximality?**



Zhu et. al., PLDI 2019

The Framework of Neural Network Shielding.

Use Case 1: As a **run-time shield** for Neural Network based controllers that may be efficient but not sound.

Use Case 2: For verification of controllers designed by humans, by checking for **containment**.

Use Case 3: In the case of multiple controllers where there exists a supervisory controller that decides which controller's "advice" to take. Maximality allows the **most permissive controller** allowing room for more behaviours

Summary: GenSys - A Scalable Fixed-Point Engine for Maximal Controller Synthesis over Infinite State Spaces

- **System Model:** Logical constraints
- **Specifications:** Safety, ω-regular
- **State Space:** Infinite
- No external templates required, unlike ConSynth.
- Elegant fixed-point procedure
- No dedicated solver required, unlike SimSynth, JSyn-VG and ConSynth
- Scalable, unlike other tools [Safety].
- Controller size is smaller as compared to other tools [Safety].
- **Future Work:** Scalability for ω-regular specs, Specification Language.
- **RQ) Games over uncountable graphs?**
- **GenSys** tool link: https://github.com/stanlysamuel/gensys.git