Compositional Algorithms for Succinct Safety Games

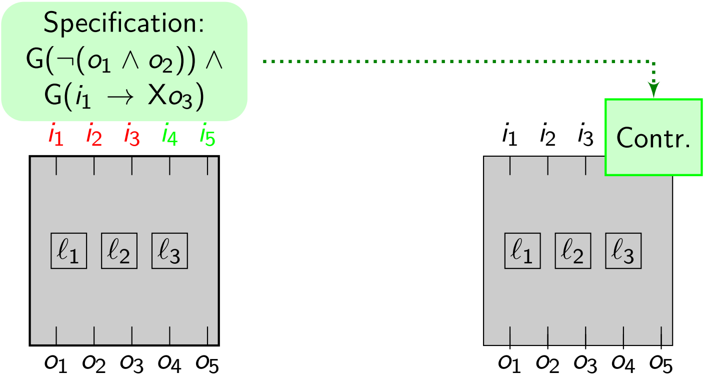
Romain Brenguier, Guillermo A. Pérez,
Jean-François Raskin, Ocan Sankur

UNIVERSITÉ LIBRE DE BRUXELLES
ULB

SYNT'15

Reactive Synthesis for circuits

AbsSynthe `https://github.com/gaperez64/AbsSynthe`

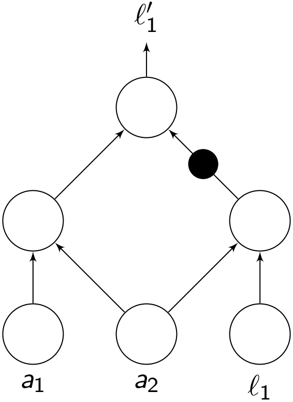


Succinct Safety Games

Safety game: $\langle \mathsf{Stat}, \mathsf{Act}_u, \mathsf{Act}_c, \delta, \mathcal{U} \rangle$

Succinct representation: $\mathsf{Stat} = \{0,1\}^L$, $\mathsf{Act}_u = \{0,1\}^{X_u}$, $\mathsf{Act}_c = \{0,1\}^{X_c}$, δ and $\mathcal{U}$ are given by And-Inverter Graphs (AIG)

- $\rightarrow$ standard file format for sequential synchronous circuits
- $\rightarrow$ used in model checking and synthesis competitions

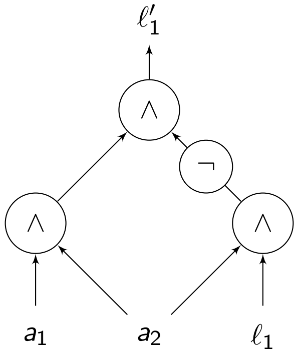


Succinct Safety Games

Safety game: $\langle \mathsf{Stat}, \mathsf{Act}_u, \mathsf{Act}_c, \delta, \mathcal{U} \rangle$

Succinct representation: $\mathsf{Stat} = \{0,1\}^L$, $\mathsf{Act}_u = \{0,1\}^{X_u}$, $\mathsf{Act}_c = \{0,1\}^{X_c}$, δ and $\mathcal{U}$ are given by And-Inverter Graphs (AIG)

- $\rightarrow$ standard file format for sequential synchronous circuits
- $\rightarrow$ used in model checking and synthesis competitions



The classical algorithm: attractor computation

For the safety game $\langle \mathrm{Stat}, \mathrm{Act}_u, \mathrm{Act}_c, \delta, \mathcal{U} \rangle$:

1. uncontrollable predecessors: states where environment can force S in 1 step: $\mathrm{UPRE}(S) = \{s \mid \exists a_u, \forall a_c, \delta(s, a_u, a_c) \in S\}$
2. Compute the least fixpoint of UPRE starting from the error states $\mathcal{U}$.

$\rightarrow$ if $s_0 \in \mathrm{Stat} \setminus \mathrm{UPRE}^*(\mathcal{U})$, controller has a winning strategy

The classical algorithm: attractor computation

For the safety game $\langle \mathrm{Stat}, \mathrm{Act}_u, \mathrm{Act}_c, \delta, \mathcal{U} \rangle$:

1. uncontrollable predecessors: states where environment can force S in 1 step: $\mathrm{UPRE}(S) = \{s \mid \exists a_u, \forall a_c, \delta(s, a_u, a_c) \in S\}$
2. Compute the least fixpoint of UPRE starting from the error states $\mathcal{U}$.

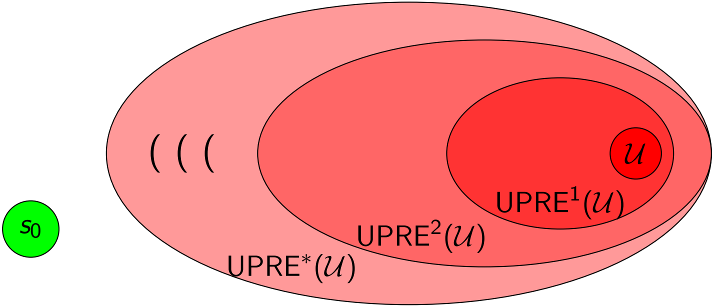
$\rightarrow$ if $s_0 \in \mathrm{Stat} \setminus \mathrm{UPRE}^*(\mathcal{U})$, controller has a winning strategy

The classical algorithm: attractor computation

For the safety game $\langle \mathsf{Stat}, \mathsf{Act}_u, \mathsf{Act}_c, \delta, \mathcal{U} \rangle$:

1. uncontrollable predecessors: states where environment can force S in 1 step: $\mathsf{UPRE}(S) = \{s \mid \exists a_u, \forall a_c, \delta(s, a_u, a_c) \in S\}$
2. Compute the least fixpoint of UPRE starting from the error states $\mathcal{U}$.

$\rightarrow$ if $s_0 \in \mathsf{Stat} \setminus \mathsf{UPRE}^*(\mathcal{U})$, controller has a winning strategy



Implementation with BDDs

We use Binary Decision Diagrams (BDDs):

- data structure to represent Boolean functions
- efficient Boolean operations ($\wedge$, $\vee$, $\forall$, $\exists$,...) and equality test

2 basic approaches:

1. Compute a transition relation

$$T(L, X_u, X_c, L') = \bigwedge_{\ell \in L} \ell' \Leftrightarrow f_\ell(L, X_u, X_c)$$

   and then set $\mathrm{UPRE}(S) = \exists X_u, \forall X_c, \exists L'.\ T(L, X_u, X_c, L') \wedge S(L')$.

   (solved approximately 150 out of 530 benchmarks from last year's competition)

2. Keep a partitioned transition relation, and substitute f_ℓ for each ℓ in S

$$\mathrm{UPRE}(S) = \exists X_u, \forall X_c : S(L')[\ell' \leftarrow f_\ell(X_u, X_c, L)]_{\ell \in L}.$$

   (solved approximately 500 benchmarks in 500 seconds)

Idea of the decomposition

Often: specifications are big conjunctions of smaller specifications

Example from amba2b9

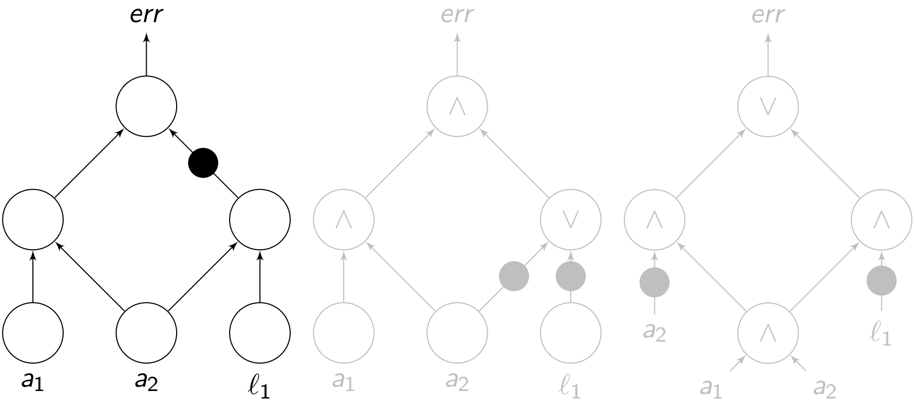
```
assign sys_safe_err = sys_safe_err0 | sys_safe_err1 | sys_safe_err2
                              | ...| sys_safe_err19;
assign o_err = ~env_safe_err & ~env_safe_err_happened &
                              (sys_safe_err | fair_err);
```

- `o_err` can be rewritten:
  `(~env_safe_err & ~env_safe_err_happened & fair_err)` | ϕ_0 | ...| ϕ_{19}
  where ϕ_i = `~env_safe_err & ~env_safe_err_happened & sys_safe_err`i
- we define a game G_i for each formula ϕ_i
- to win the "big" game, we must win each "small" game G_i

Decomposition of AIGs

We must recover the structure of the specifications from the AIG

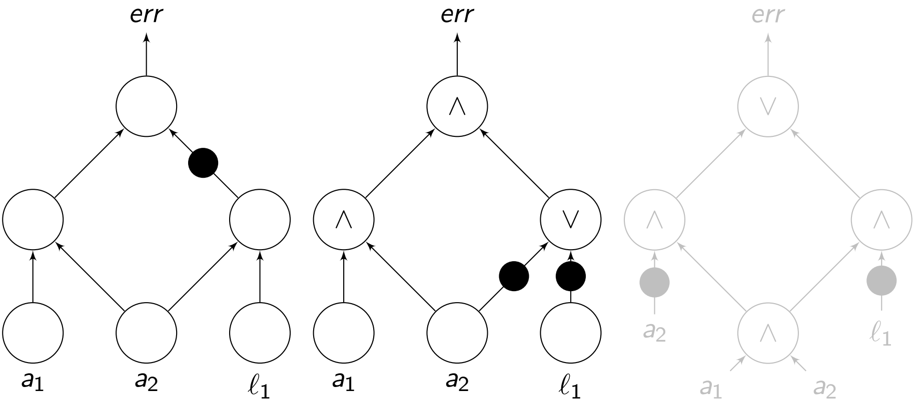
- Explore the graph until encountering a negation
- This corresponds to a disjunction, and it can be distributed over



Decomposition of AIGs

We must recover the structure of the specifications from the AIG

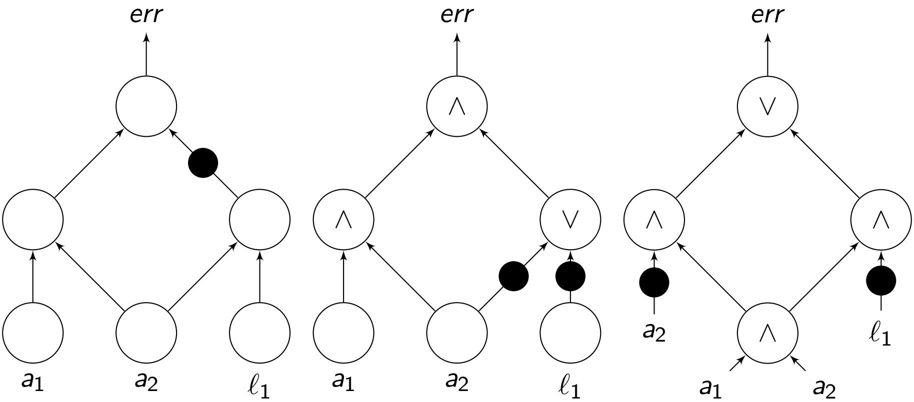
- Explore the graph until encountering a negation
- This corresponds to a disjunction, and it can be distributed over



Decomposition of AIGs

We must recover the structure of the specifications from the AIG

- Explore the graph until encountering a negation
- This corresponds to a disjunction, and it can be distributed over



Subgames

We obtain a decomposition $err = e_1 \vee e_2 \vee \cdots \vee e_n$

If formula e_i does not depend on all latches, solving the game for e_i can be more efficient

Cone of influence

$\mathtt{cone}(e_i)$: set of variables on which e_i depends (directly or indirectly)
$\rightarrow$ can be over-approximated efficiently by exploring the AIG

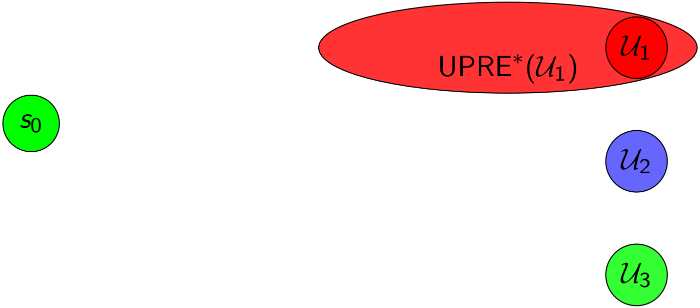
We consider the game G_i where the error function is given by e_i and we only consider variables in $\mathtt{cone}(e_i)$

Compositional algorithm 1: Global aggregation

- Compute the winning region of each subgame
- If the intersection does not contain the initial state, then there is no controller
- Otherwise compute the fixpoint starting from the intersection

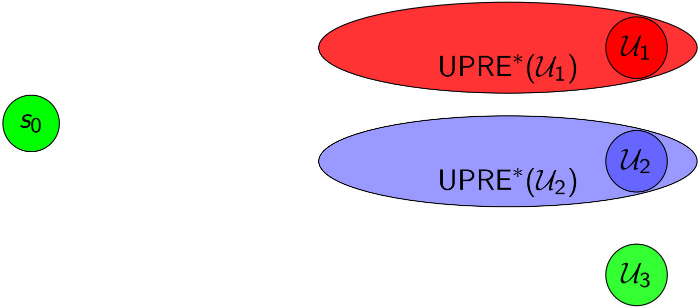
Compositional algorithm 1: Global aggregation

- Compute the winning region of each subgame
- If the intersection does not contain the initial state, then there is no controller
- Otherwise compute the fixpoint starting from the intersection



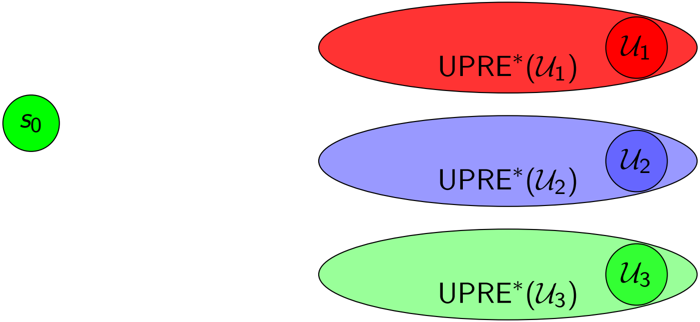
Compositional algorithm 1: Global aggregation

- Compute the winning region of each subgame
- If the intersection does not contain the initial state, then there is no controller
- Otherwise compute the fixpoint starting from the intersection



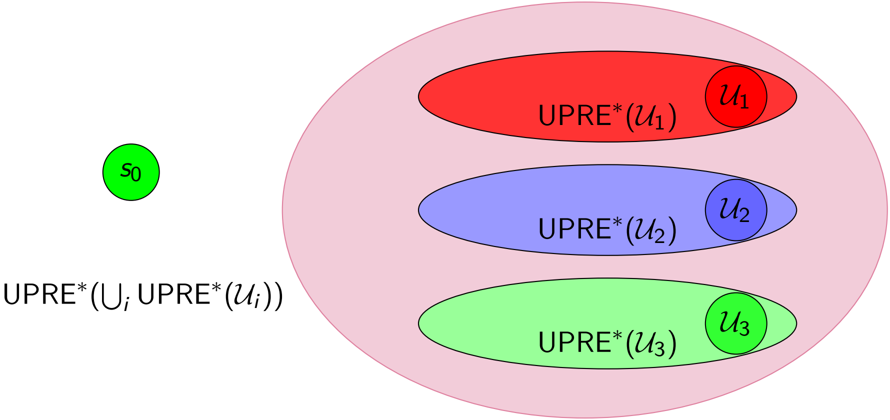
Compositional algorithm 1: Global aggregation

- Compute the winning region of each subgame
- If the intersection does not contain the initial state, then there is no controller
- Otherwise compute the fixpoint starting from the intersection



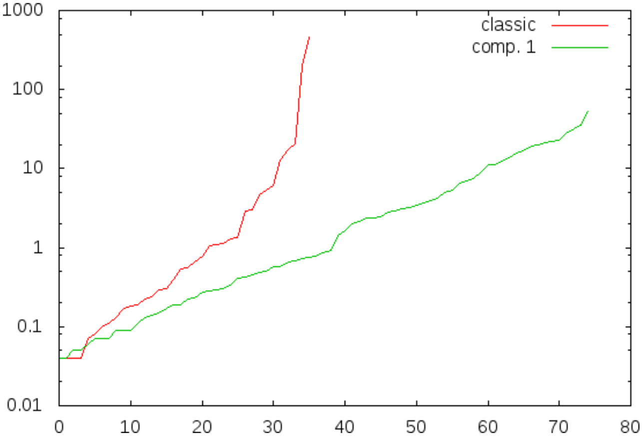
Compositional algorithm 1: Global aggregation

- Compute the winning region of each subgame
- If the intersection does not contain the initial state, then there is no controller
- Otherwise compute the fixpoint starting from the intersection



Matrix multiplication benchmarks

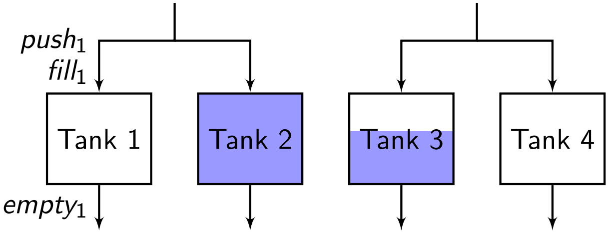
$$err \equiv \begin{pmatrix} u_{1,1} & u_{1,2} \\ u_{2,1} & u_{2,2} \end{pmatrix} \cdot \begin{pmatrix} u'_{1,1} & u'_{1,2} \\ u'_{2,1} & u'_{2,2} \end{pmatrix} \neq \begin{pmatrix} c_{1,1} & c_{1,2} \\ c_{2,1} & c_{2,2} \end{pmatrix}$$

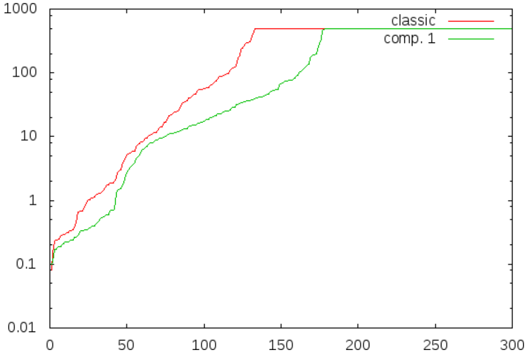


y-axis: time limit (in seconds),

f *x*-axis: number of benchmarks that are solvable within the time limit

Washing system benchmarks



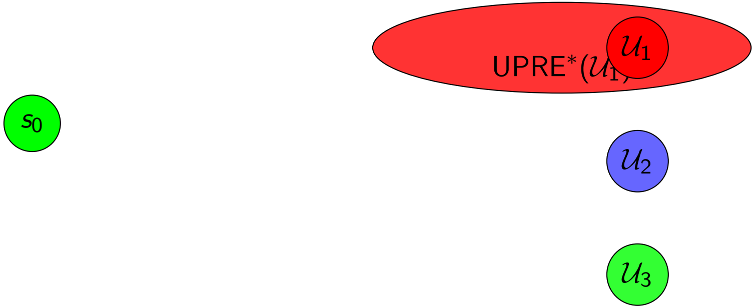


Compositional algorithm 2: Incremental aggregation

While there are several subgames: join two of them and solve the new sub-game that is obtained

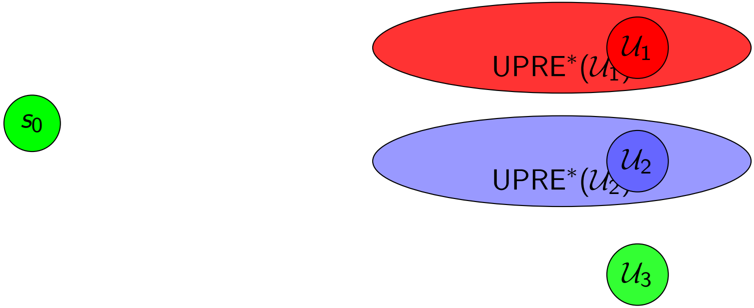
Compositional algorithm 2: Incremental aggregation

While there are several subgames: join two of them and solve the new sub-game that is obtained



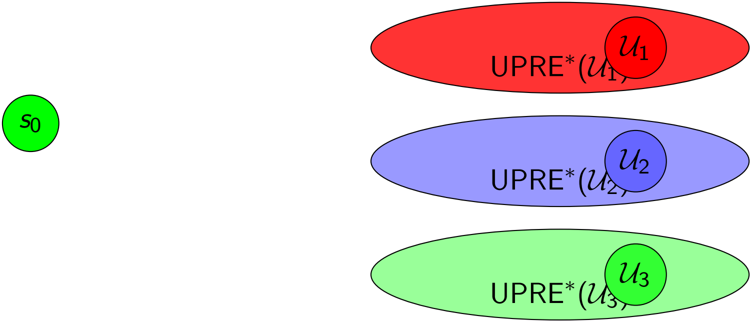
Compositional algorithm 2: Incremental aggregation

While there are several subgames: join two of them and solve the new sub-game that is obtained



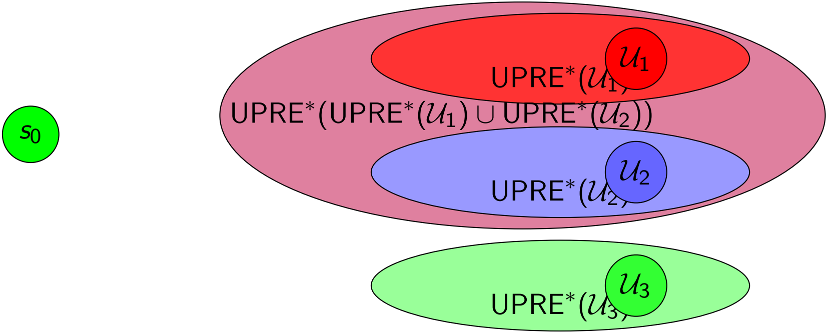
Compositional algorithm 2: Incremental aggregation

While there are several subgames: join two of them and solve the new sub-game that is obtained



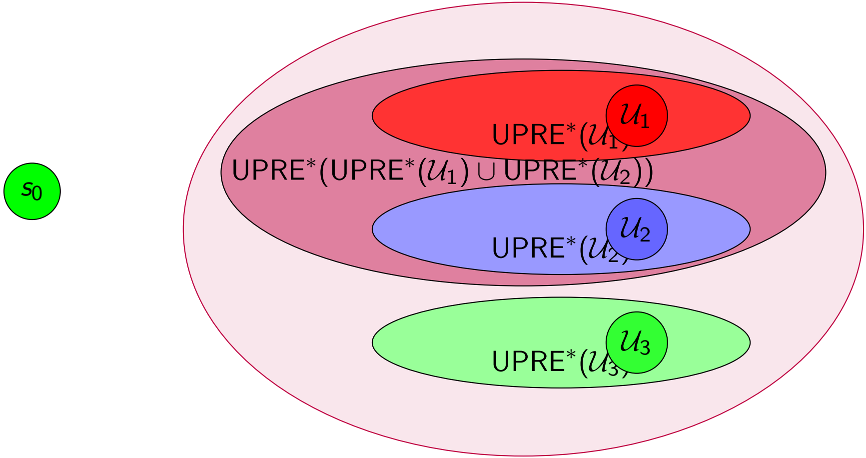
Compositional algorithm 2: Incremental aggregation

While there are several subgames: join two of them and solve the new sub-game that is obtained



Compositional algorithm 2: Incremental aggregation

While there are several subgames: join two of them and solve the new sub-game that is obtained



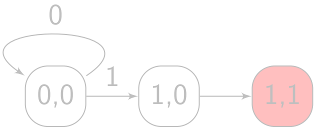
Compositional algorithm 3: Back and forth

After the computation in each subgame, project the union of unsafe states in the subgames, and repeat until stabilized

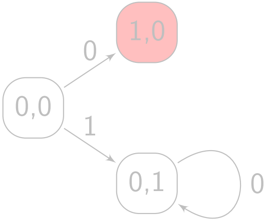
$\rightarrow$ A similar idea was used in [FJR10, Compositional Algorithms for LTL Synthesis]

Example:

- $err = (\ell_1 \wedge \ell_2) \vee (\neg \ell_1 \wedge \ell_3)$
- $\ell'_1 = c \vee \ell_1$;
- $\ell'_2 = \ell_1$;
- $\ell'_3 = \neg \ell_1 \wedge \neg c$;



Subgame ℓ_1, ℓ_2



Subgame ℓ_1, ℓ_3

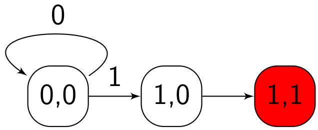
Compositional algorithm 3: Back and forth

After the computation in each subgame, project the union of unsafe states in the subgames, and repeat until stabilized

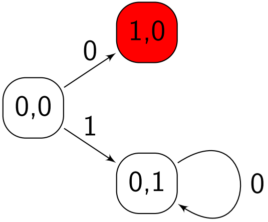
→ A similar idea was used in [FJR10, Compositional Algorithms for LTL Synthesis]

Example:

- $err = (\ell_1 \wedge \ell_2) \vee (\neg\ell_1 \wedge \ell_3)$
- $\ell_1' = c \vee \ell_1$;
- $\ell_2' = \ell_1$;
- $\ell_3' = \neg\ell_1 \wedge \neg c$;



Subgame ℓ_1, ℓ_2



Subgame ℓ_1, ℓ_3

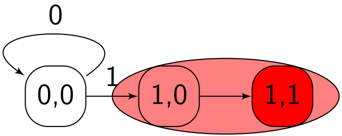
Compositional algorithm 3: Back and forth

After the computation in each subgame, project the union of unsafe states in the subgames, and repeat until stabilized

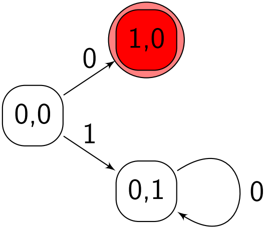
$\rightarrow$ A similar idea was used in [FJR10, Compositional Algorithms for LTL Synthesis]

Example:

- $err = (\ell_1 \wedge \ell_2) \vee (\neg\ell_1 \wedge \ell_3)$
- $\ell_1' = c \vee \ell_1$;
- $\ell_2' = \ell_1$;
- $\ell_3' = \neg\ell_1 \wedge \neg c$;



Subgame ℓ_1, ℓ_2



Subgame ℓ_1, ℓ_3

$\mathrm{UPRE}_1 \cup \mathrm{UPRE}_2 = \ell_1 \vee \ell_3$

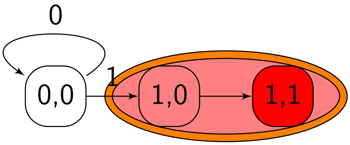
Compositional algorithm 3: Back and forth

After the computation in each subgame, project the union of unsafe states in the subgames, and repeat until stabilized

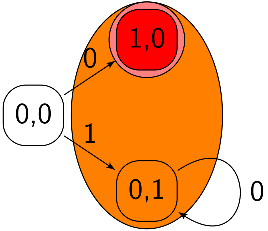
→ A similar idea was used in [FJR10, Compositional Algorithms for LTL Synthesis]

Example:

- $err = (\ell_1 \wedge \ell_2) \vee (\neg\ell_1 \wedge \ell_3)$
- $\ell_1' = c \vee \ell_1$;
- $\ell_2' = \ell_1$;
- $\ell_3' = \neg\ell_1 \wedge \neg c$;



Subgame ℓ_1, ℓ_2



Subgame ℓ_1, ℓ_3

$\mathrm{UPRE}_1 \cup \mathrm{UPRE}_2 = \ell_1 \vee \ell_3$

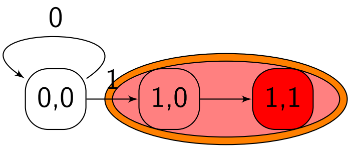
Compositional algorithm 3: Back and forth

After the computation in each subgame, project the union of unsafe states in the subgames, and repeat until stabilized

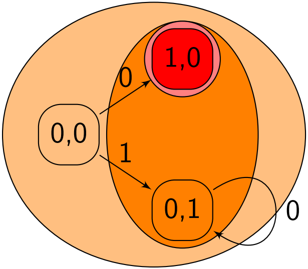
→ A similar idea was used in [FJR10, Compositional Algorithms for LTL Synthesis]

Example:

- $err = (\ell_1 \wedge \ell_2) \vee (\neg\ell_1 \wedge \ell_3)$
- $\ell_1' = c \vee \ell_1$;
- $\ell_2' = \ell_1$;
- $\ell_3' = \neg\ell_1 \wedge \neg c$;



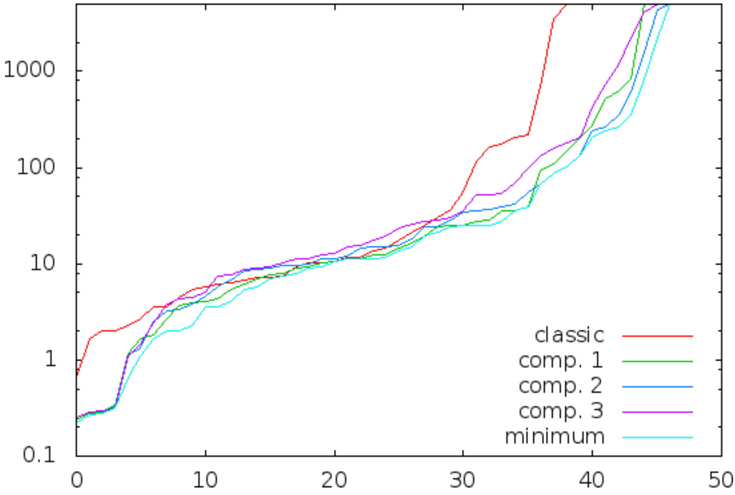
Subgame ℓ_1, ℓ_2



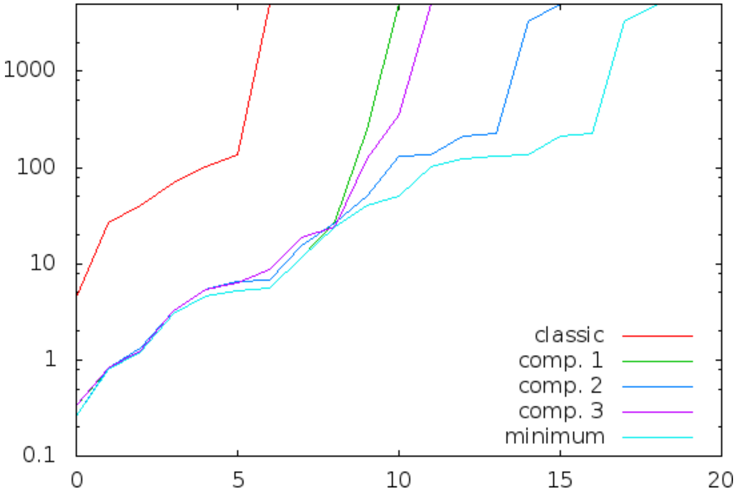
Subgame ℓ_1, ℓ_3

$\mathrm{UPRE}_1 \cup \mathrm{UPRE}_2 = \ell_1 \vee \ell_3$

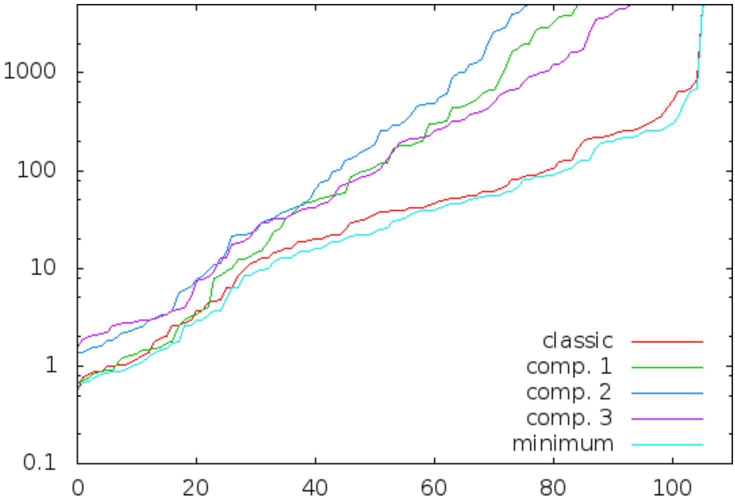
Benchmarks translated from LTL specifications / Load Balancing



Benchmarks translated from LTL specifications / Generalized Buffer



AMBA Benchmarks



Conclusion

- Application of a compositional approach to monolithic AIG specifications
- Can solve problems not handled by the classical algorithm
- Sometimes much more efficient
- Applying the different algorithms in parallel works well in practice

Conclusion

- Application of a compositional approach to monolithic AIG specifications
- Can solve problems not handled by the classical algorithm
- Sometimes much more efficient
- Applying the different algorithms in parallel works well in practice

Thank you