Efficient Handling of Obligation Constraints in Synthesis from Omega-Regular Specifications

Saqib bin Sohail

Department of Electrical and Computer Engineering
University of Colorado at Boulder

FMCAD 2013

Outline

1. Introduction: Synthesis from ω-regular properties
2. The Challenges in improving Quality of Results
3. $\mathcal{R}$-Generable languages
4. Experimental Results
5. Conclusions

Outline

1 Introduction: Synthesis from ω-regular properties

2 The Challenges in improving Quality of Results

3 $\mathcal{R}$-Generable languages

4 Experimental Results

5 Conclusions

Realizability of an ω-regular property

Let ϕ be an ω-regular property describing the relation between inputs X_I and outputs X_O where $\Sigma_I = 2^{X_I}$ and $\Sigma_O = 2^{X_O}$.

The realizability problem for ϕ is to decide whether there is a strategy $\tau : \Sigma_I^* \rightarrow \Sigma_O$ which generates an output word $\sigma_O \in \Sigma_O^\omega$ for every input word $\sigma_I \in \Sigma_I^\omega$ such that the input-output word

$$\sigma = (\sigma_I^0, \sigma_O^0), (\sigma_I^1, \sigma_O^1), (\sigma_I^2, \sigma_O^2), \ldots$$

satisfies ϕ.

Realizability and Synthesis

If a specification (set of ω-regular properties) is realizable then from the winning strategy we can generate an implementation (transducer) which guarantees the satisfaction of the specification.

Various approaches of checking Realizability

- **Pnueli and Rosner (POPL'89)**
  Requires determinization
- **"Safraless" approach - Vardi *et al.* (FOCS'05)**
  Same worst case complexity but avoids determinization
- **Reactive(1) Designs - Piterman *et al.* (VMCAI'06)** Subset of ω-regular languages that can be synthesized efficiently
- **SAFETY-FIRST - Sohail *et al.* (VMCAI'08, FMCAD'09)**
  - Two-stage approach improves efficiency
  - Achieved efficiency without sacrificing generality
- **BOUNDED SYNTHESIS and its variants - Ehlers, Raskin *et al.***
  Sequence of safety games

Efficiency and Quality

Current techniques focus on efficiency of the realizability check and overlook the quality of the implementation.

Quality of Results (QoR) - the amount of combinational and sequential logic required by the implementation.

The implementation generated by automatic techniques is not good enough even when compared against an implementation generated by a novice designer.

Efficiency and Quality

Current techniques focus on efficiency of the realizability check and overlook the quality of the implementation.

Quality of Results (QoR) - the amount of combinational and sequential logic required by the implementation.

The implementation generated by automatic techniques is not good enough even when compared against an implementation generated by a novice designer.

Outline

1 Introduction: Synthesis from ω-regular properties

2 The Challenges in improving Quality of Results

3 $\mathcal{R}$-Generable languages

4 Experimental Results

5 Conclusions

Redundancies and Inefficiencies in Symbolic Encodings

Symbolic algorithms have had significant impact on the performance of model checking algorithms.

Symbolic encoding of a game graph plays a significant role in the efficiency of game playing algorithms.

However, finding an efficient encoding of the game graph is not a trivial task.

Redundancies and Inefficiencies in Symbolic Encodings

Symbolic algorithms have had significant impact on the performance of model checking algorithms.

Symbolic encoding of a game graph plays a significant role in the efficiency of game playing algorithms.

However, finding an efficient encoding of the game graph is not a trivial task.

Redundancies and Inefficiencies in Symbolic Encodings

Symbolic algorithms have had significant impact on the performance of model checking algorithms.

Symbolic encoding of a game graph plays a significant role in the efficiency of game playing algorithms.

However, finding an efficient encoding of the game graph is not a trivial task.

Redundancies and Inefficiencies in Symbolic Encodings... (continued)

A common approach of converting the specification to a game graph is:

- obtain a game graph for each property through explicit techniques
- then generate the symbolic representation of the game graph
- then composing the symbolic representation of these game graphs to yield the game graph of the specification.

Redundancies and Inefficiencies in Symbolic Encodings... (continued)

A common approach of converting the specification to a game graph is:

- obtain a game graph for each property through explicit techniques
- then generate the symbolic representation of the game graph
- then composing the symbolic representation of these game graphs to yield the game graph of the specification.

Redundancies and Inefficiencies in Symbolic Encodings... (continued)

This approach often creates game graphs which contain unreachable states, simulation equivalent states and states that can easily be identified as winning/losing.

Once these states have been identified and removed, the challenge is to generate a suitable encoding for the simplified game graph.

Redundancies and Inefficiencies in Symbolic Encodings... (continued)

This approach often creates game graphs which contain unreachable states, simulation equivalent states and states that can easily be identified as winning/losing.

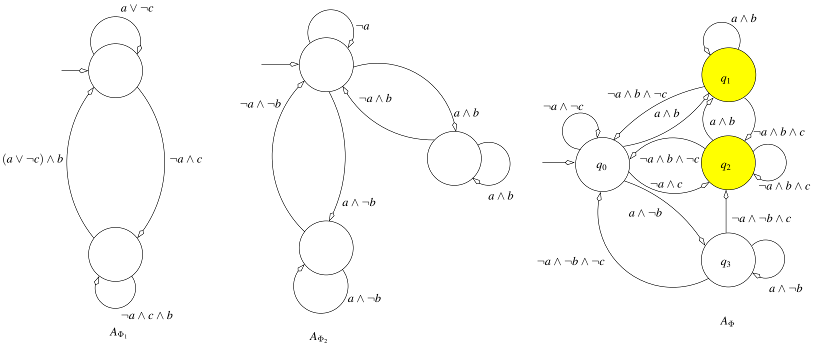
Once these states have been identified and removed, the challenge is to generate a suitable encoding for the simplified game graph.

Unreachable and simulation equivalent states

The composed automaton may contain simulation equivalent states even if the original two automata do not.

Unreachable and simulation equivalent states

The composed automaton may contain simulation equivalent states even if the original two automata do not.



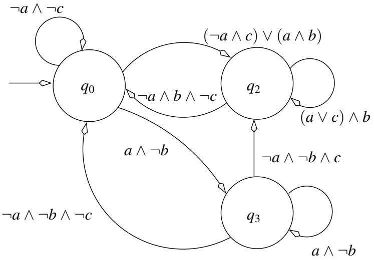
In this example, q_1 and q_2 are simulation equivalent.

Unreachable and simulation equivalent states... (continued)

$$q_0 = s_0 \qquad q_2 = \neg s_0 \wedge \neg s_1 \qquad q_3 = \neg s_0 \wedge s_1$$

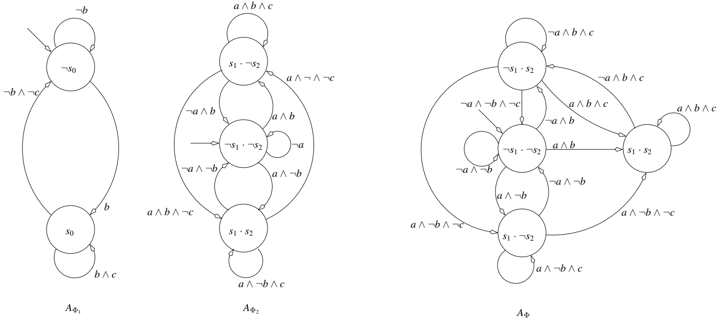
$$\overline{s_0} = (s_0 \vee b) \wedge \neg a \wedge \neg c$$

$$\overline{s_1} = a \wedge \neg b$$



A_Φ

Cyclic Dependencies – bad for BDDs



$$\bar{s}_0 = b, \quad \bar{s}_1 = a, \quad \bar{s}_2 = (a \wedge c \wedge s_2) \vee (a \wedge \neg b \wedge \neg s_1) \vee (a \wedge b \wedge \neg c \wedge s_1)$$

$$\bar{S}_1 = a \vee (\neg S_2 \wedge b)$$

$$\bar{S}_2 = (\neg a \wedge b) \vee (a \wedge \neg b \wedge \neg S_2) \vee (a \wedge c \wedge S_1) \vee (a \wedge \neg c \wedge S_1)$$

Why do Safety Properties exist in a specification?

The safety properties in the specification capture the transition relation of implementations that can satisfy the specification.

Useful pieces of information about the transition relation are scattered accross different properties.

$\{a\} \rightarrow$ is the set of inputs $\{x, y\} \rightarrow$ is the set of outputs
$\{\mathsf{G}(a \rightarrow \mathsf{X}\, x), \mathsf{G}(\neg a \rightarrow \mathsf{X}\, y)\} \rightarrow$ set of safety properties.

Both the outputs depend on the previous value of the input a.

Why do Safety Properties exist in a specification?

The safety properties in the specification capture the transition relation of implementations that can satisfy the specification.

Useful pieces of information about the transition relation are scattered accross different properties.

$\{a\} \rightarrow$ is the set of inputs $\{x, y\} \rightarrow$ is the set of outputs
$\{\mathsf{G}(a \rightarrow \mathsf{X}x), \mathsf{G}(\neg a \rightarrow \mathsf{X}y)\} \rightarrow$ set of safety properties.

Both the outputs depend on the previous value of the input a.

Why do Safety Properties exist in a specification?

The safety properties in the specification capture the transition relation of implementations that can satisfy the specification.

Useful pieces of information about the transition relation are scattered accross different properties.

$\{a\} \rightarrow$ is the set of inputs $\{x, y\} \rightarrow$ is the set of outputs
$\{\mathsf{G}(a \rightarrow \mathsf{X}\, x), \mathsf{G}(\neg a \rightarrow \mathsf{X}\, y)\} \rightarrow$ set of safety properties.

Both the outputs depend on the previous value of the input a.

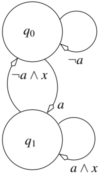
Why do Safety Properties exist in a specification? ...(continued)

The existing approaches are often unable to take advantage of useful information because it is often obscured and hard to recover.

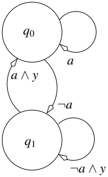
Automata Based conversion

$\{a\} \rightarrow$ is the set of inputs $\{x, y\} \rightarrow$ is the set of outputs
$\{\mathsf{G}(a \rightarrow \mathsf{X}\, x), \mathsf{G}(\neg a \rightarrow \mathsf{X}\, y)\} \rightarrow$ set of safety properties.

The states of the game represent the memory that is required to remember some past event.



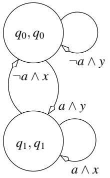
$\mathsf{G}(a \rightarrow \mathsf{X}\, x)$



$\mathsf{G}(\neg a \rightarrow \mathsf{X}\, y)$

The state space of each game is encoded with a single binary variable.

Automata Based conversion . . . (continued)



$\mathsf{G}(a \rightarrow \mathsf{X}\,x) \wedge \mathsf{G}(\neg a \rightarrow \mathsf{X}\,y)$

The composed game has two reachable states. However, it is encoded by two binary variables.

$\mathcal{R}$-Generable languages

An $\mathcal{R}$-generable language L can be generated by a relation such that every two consecutive letters of a word in the language satisfy some relation R.

$$\forall w \in L \,.\, \forall i \geq 0 \,.\, (w_i, w_{i+1}) \in R$$

$\mathcal{R}$-generable languages are accepted by 1-definite safety automata which are initially free.

Not all safety languages are $\mathcal{R}$-generable.

However, every safety language defined over Σ can be embedded in an $\mathcal{R}$-generable language defined over $\hat{\Sigma}$, where $\Sigma \subseteq \hat{\Sigma}$.

$\mathcal{R}$-Generable languages

An $\mathcal{R}$-generable language L can be generated by a relation such that every two consecutive letters of a word in the language satisfy some relation R.

$$\forall w \in L \,.\, \forall i \geq 0 \,.\, (w_i, w_{i+1}) \in R$$

$\mathcal{R}$-generable languages are accepted by 1-definite safety automata which are initially free.

Not all safety languages are $\mathcal{R}$-generable.

However, every safety language defined over Σ can be embedded in an $\mathcal{R}$-generable language defined over $\hat{\Sigma}$, where $\Sigma \subseteq \hat{\Sigma}$.

$\mathcal{R}$-Generable languages

An $\mathcal{R}$-generable language L can be generated by a relation such that every two consecutive letters of a word in the language satisfy some relation R.

$$\forall w \in L \,.\, \forall i \geq 0 \,.\, (w_i, w_{i+1}) \in R$$

$\mathcal{R}$-generable languages are accepted by 1-definite safety automata which are initially free.

Not all safety languages are $\mathcal{R}$-generable.

However, every safety language defined over Σ can be embedded in an $\mathcal{R}$-generable language defined over $\hat{\Sigma}$, where $\Sigma \subseteq \hat{\Sigma}$.

$\mathcal{R}$-Generable languages

An $\mathcal{R}$-generable language L can be generated by a relation such that every two consecutive letters of a word in the language satisfy some relation R.

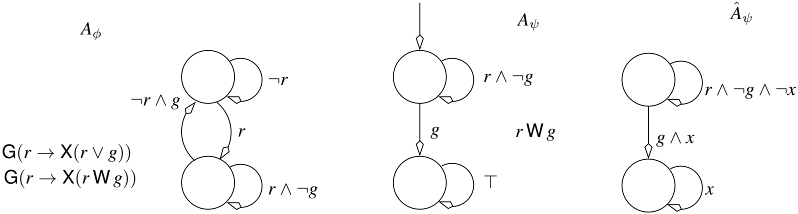
$$\forall w \in L \,.\, \forall i \geq 0 \,.\, (w_i, w_{i+1}) \in R$$

$\mathcal{R}$-generable languages are accepted by 1-definite safety automata which are initially free.

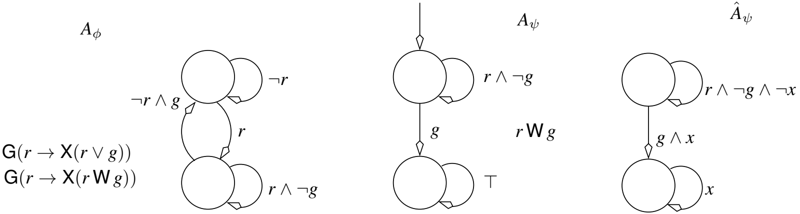
Not all safety languages are $\mathcal{R}$-generable.

However, every safety language defined over Σ can be embedded in an $\mathcal{R}$-generable language defined over $\hat{\Sigma}$, where $\Sigma \subseteq \hat{\Sigma}$.

$\mathcal{R}$-Generable languages... (continued)



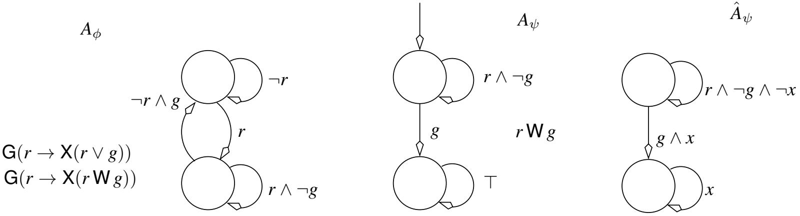
$\mathcal{R}$-Generable languages... (continued)



$$R = \neg r_L \vee r \vee g$$

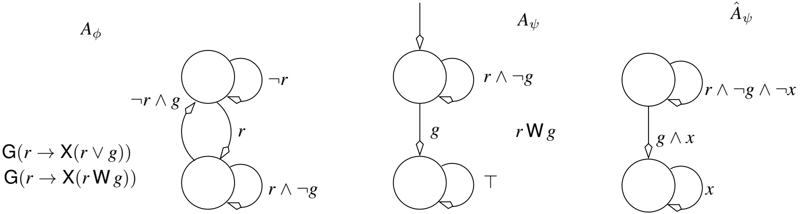
where r_L and g_L represent the previous values of the inputs r and g.

$\mathcal{R}$-Generable languages... (continued)



$$\Gamma : \hat{\Sigma} \to \Sigma \qquad \Gamma : \hat{\Sigma}^\omega \to \Sigma^\omega$$

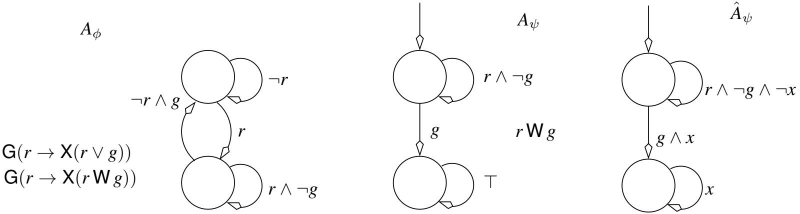
$\mathcal{R}$-Generable languages... (continued)



$$R = (r_L \wedge \neg g_L \wedge \neg x_L) \wedge ((r \wedge \neg g \wedge \neg x) \vee (g \wedge x)) \vee (x_L \wedge x)$$

$$L(A_\phi) \subseteq \Gamma(L(\hat{A}_\phi))$$

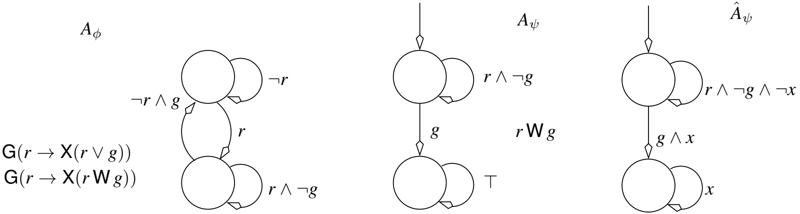
$\mathcal{R}$-Generable languages... (continued)



$$R = (r_L \wedge \neg g_L \wedge \neg x_L) \wedge ((r \wedge \neg g \wedge \neg x) \vee (g \wedge x)) \vee (x_L \wedge x)$$

$$I = (r \wedge \neg g \wedge \neg x) \vee (g \wedge x) \quad .$$

$\mathcal{R}$-Generable languages... (continued)



$$\Gamma(L(\hat{A}_\phi)) = L(A_\phi)$$

The projection function Γ when restricted to $L(A_\phi)$ and $L(\hat{A}_\phi)$ is a bijection.

Relation Based conversion

$\{a\} \rightarrow$ is the set of inputs $\{x, y\} \rightarrow$ is the set of outputs
$\{\mathsf{G}(a \rightarrow \mathsf{X}\, x), \mathsf{G}(\neg a \rightarrow \mathsf{X}\, y)\}$ is the set of safety properties.

$$(\neg a_L \vee x) \wedge (a_L \vee y)$$

The past events that need to be remembered are not abstracted by state variables.

Relation Based conversion

$\{a\} \rightarrow$ is the set of inputs $\{x, y\} \rightarrow$ is the set of outputs
$\{\mathsf{G}(a \rightarrow \mathsf{X}\,x), \mathsf{G}(\neg a \rightarrow \mathsf{X}\,y)\}$ is the set of safety properties.

$$(\neg a_L \vee x) \wedge (a_L \vee y)$$

The past events that need to be remembered are not abstracted by state variables.

Relation Based conversion

$\{a\} \rightarrow$ is the set of inputs $\{x, y\} \rightarrow$ is the set of outputs
$\{\mathsf{G}(a \rightarrow \mathsf{X}\, x), \mathsf{G}(\neg a \rightarrow \mathsf{X}\, y)\}$ is the set of safety properties.

$$(\neg a_L \vee x) \wedge (a_L \vee y)$$

The past events that need to be remembered are not abstracted by state variables.

Checking Realizability

Given $\mathcal{I} = \{r\}$ $\quad \mathcal{O} = \{g, h, m\}$

$$R = (\neg r_L \vee \neg g_L \vee \neg m) \wedge (\neg r_L \vee \neg h_L \vee m)$$

$Z_0 = \exists \mathcal{O} \,.\, \forall I \,.\, R \wedge \top = \neg r_L \vee \neg g_L \vee \neg h_L \qquad T = (\neg r \vee \neg g \vee \neg h)$

$Z_1 = \exists \mathcal{O} \,.\, \forall I \,.\, R \wedge Z = \neg r_L \vee \neg g_L \vee \neg h_L$

It is an SCC computation using R as the transition relation and $\mathcal{O}_L \cup \mathcal{I}_L$ as the current state variables.

The variables $\mathcal{O} \cup \mathcal{I}$ are interpreted both as the input variables and next state variables.

Checking Realizability

Given $\mathcal{I} = \{r\} \quad \mathcal{O} = \{g, h, m\}$

$$R = (\neg r_L \vee \neg g_L \vee \neg m) \wedge (\neg r_L \vee \neg h_L \vee m)$$

$Z_0 = \exists \mathcal{O} \,.\, \forall I \,.\, R \wedge \top = \neg r_L \vee \neg g_L \vee \neg h_L \qquad T = (\neg r \vee \neg g \vee \neg h)$

$Z_1 = \exists \mathcal{O} \,.\, \forall I \,.\, R \wedge Z = \neg r_L \vee \neg g_L \vee \neg h_L$

It is an SCC computation using R as the transition relation and $\mathcal{O}_L \cup \mathcal{I}_L$ as the current state variables.

The variables $\mathcal{O} \cup \mathcal{I}$ are interpreted both as the input variables and next state variables.

Checking Realizability

Given $\mathcal{I} = \{r\} \quad \mathcal{O} = \{g, h, m\}$

$$R = (\neg r_L \vee \neg g_L \vee \neg m) \wedge (\neg r_L \vee \neg h_L \vee m)$$

$Z_0 = \exists \mathcal{O} \,.\, \forall I \,.\, R \wedge \top = \neg r_L \vee \neg g_L \vee \neg h_L \qquad T = (\neg r \vee \neg g \vee \neg h)$

$Z_1 = \exists \mathcal{O} \,.\, \forall I \,.\, R \wedge Z = \neg r_L \vee \neg g_L \vee \neg h_L$

It is an SCC computation using R as the transition relation and $\mathcal{O}_L \cup \mathcal{I}_L$ as the current state variables.

The variables $\mathcal{O} \cup \mathcal{I}$ are interpreted both as the input variables and next state variables.

Boolean Equations and Combinational Synthesis

The equation is

$$R \wedge Z = \top$$

where $\mathcal{O}$ are the unknowns and $\mathcal{O}_L \cup \mathcal{I}_L \cup \mathcal{I}$ are the independant variables.

$$h = h_i$$
$$g = (\neg r \vee \neg h_i) \wedge g_i$$
$$m = h_L \vee (\neg r_L \wedge m_i)$$

Parameterized Transition relation

Parameterized transition relation is essential for the correctness of this SAFETY FIRST approach.

Consider the liveness property $\mathsf{G}\,\mathsf{F}(m) \wedge \mathsf{G}\,\mathsf{F}(\neg m)$.

$h = h_i$
$g = (\neg r \vee \neg h_i) \wedge g_i$
$m = h_L \vee (\neg r_L \wedge m_i)$

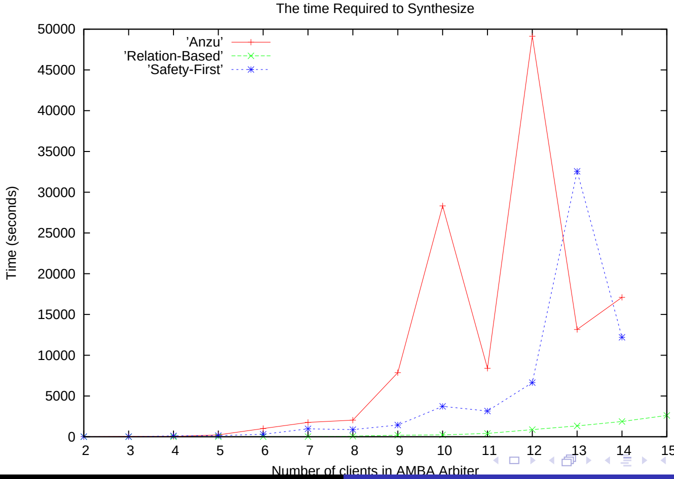
Parameterized Transition relation

Parameterized transition relation is essential for the correctness of this SAFETY FIRST approach.

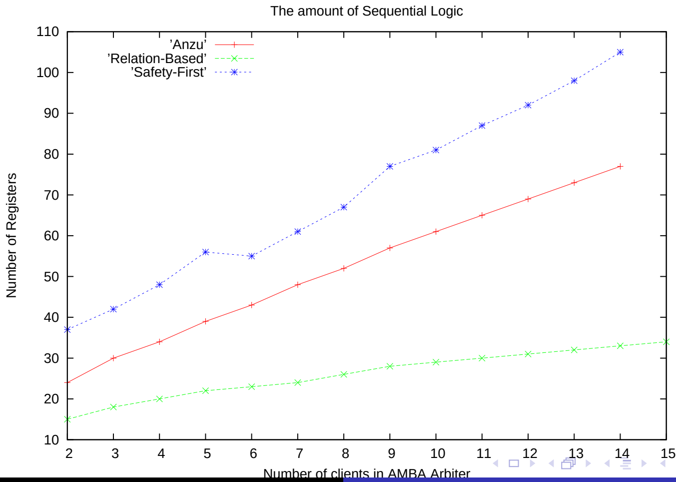
Consider the liveness property $\mathsf{G}\,\mathsf{F}(m) \wedge \mathsf{G}\,\mathsf{F}(\neg m)$.

$$h = h_i$$
$$g = (\neg r \vee \neg h_i) \wedge g_i$$
$$m = h_L \vee (\neg r_L \wedge m_i)$$

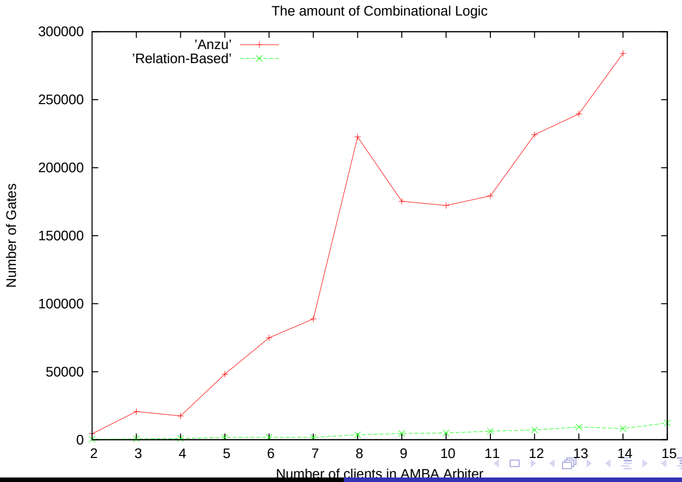
Results - Time



Results - Sequential Logic



Results - Combinational Logic



Advantages of Relation based approach

1. The relation often requires fewer symbolic variables.
2. The relation captures the intent of safety properties in the specification, therefore, debugging is a lot easier.
3. The problem of sequential synthesis is converted to a problem of combinational synthesis.
4. Retiming may improve the parameteric transition relation.
5. This approach has been extended to obligation properties.

Advantages of Relation based approach

1. The relation often requires fewer symbolic variables.
2. The relation captures the intent of safety properties in the specification, therefore, debugging is a lot easier.
3. The problem of sequential synthesis is converted to a problem of combinational synthesis.
4. Retiming may improve the parameteric transition relation.
5. This approach has been extended to obligation properties.

Advantages of Relation based approach

1. The relation often requires fewer symbolic variables.
2. The relation captures the intent of safety properties in the specification, therefore, debugging is a lot easier.
3. The problem of sequential synthesis is converted to a problem of combinational synthesis.
4. Retiming may improve the parameteric transition relation.
5. This approach has been extended to obligation properties.

Advantages of Relation based approach

1. The relation often requires fewer symbolic variables.
2. The relation captures the intent of safety properties in the specification, therefore, debugging is a lot easier.
3. The problem of sequential synthesis is converted to a problem of combinational synthesis.
4. Retiming may improve the parameteric transition relation.
5. This approach has been extended to obligation properties.

Advantages of Relation based approach

1. The relation often requires fewer symbolic variables.
2. The relation captures the intent of safety properties in the specification, therefore, debugging is a lot easier.
3. The problem of sequential synthesis is converted to a problem of combinational synthesis.
4. Retiming may improve the parameteric transition relation.
5. This approach has been extended to obligation properties.

THANK YOU

Parameterized Transition relation

$\{a\} \rightarrow$ set of inputs $\quad \{x, y\} \rightarrow$ set of outputs
$\{\mathsf{G}((a \wedge \neg y) \rightarrow (\mathsf{X}\, x \vee \mathsf{X}\, y)), \mathsf{G}((\neg a \wedge x \wedge \mathsf{X}\, a) \rightarrow \mathsf{X}\, \neg y)\}$
is the set of safety properties

$\{\mathsf{G}(a \rightarrow \mathsf{F}(x \leftrightarrow \neg y))\}$ is the liveness property

$x = x_i$
$y = (a_L \wedge \neg y_L) \wedge x_i \vee (a_L \vee \neg x_L \vee a) \wedge y_i$

Parameterized Transition relation

$\{a\} \rightarrow$ set of inputs $\quad \{x, y\} \rightarrow$ set of outputs
$\{\mathsf{G}((a \wedge \neg y) \rightarrow (\mathsf{X}\, x \vee \mathsf{X}\, y)), \mathsf{G}((\neg a \wedge x \wedge \mathsf{X}\, a) \rightarrow \mathsf{X}\, \neg y)\}$
is the set of safety properties

$\{\mathsf{G}(a \rightarrow \mathsf{F}(x \leftrightarrow \neg y))\}$ is the liveness property

$x = x_i$
$y = (a_L \wedge \neg y_L) \wedge x_i \vee (a_L \vee \neg x_L \vee a) \wedge y_i$

Parameterized Transition relation

$\{a\} \rightarrow$ set of inputs $\{x, y\} \rightarrow$ set of outputs
$\{\mathsf{G}((a \wedge \neg y) \rightarrow (\mathsf{X}\, x \vee \mathsf{X}\, y)), \mathsf{G}((\neg a \wedge x \wedge \mathsf{X}\, a) \rightarrow \mathsf{X}\, \neg y)\}$
is the set of safety properties

$\{\mathsf{G}(a \rightarrow \mathsf{F}(x \leftrightarrow \neg y))\}$ is the liveness property

$x = x_i$
$y = (a_L \wedge \neg y_L) \wedge x_i \vee (a_L \vee \neg x_L \vee a) \wedge y_i$

Boolean Equations and Combinational Synthesis

$\{a\} \rightarrow$ set of inputs $\{x, y\} \rightarrow$ set of outputs
$\{\mathsf{G}((a \wedge \neg y) \rightarrow \mathsf{X}(x \vee y)), \mathsf{G}((\neg a \wedge x \wedge \mathsf{X}\, a) \rightarrow \mathsf{X}\, \neg y)\}$
is the set of safety properties

Boolean Equations and Combinational Synthesis

$\{a\} \rightarrow$ set of inputs $\quad \{x, y\} \rightarrow$ set of outputs
$\{\mathsf{G}((a \wedge \neg y) \rightarrow \mathsf{X}(x \vee y)), \mathsf{G}((\neg a \wedge x \wedge \mathsf{X}\, a) \rightarrow \mathsf{X}\, \neg y)\}$
is the set of safety properties

$$(\neg a_L \vee y_L \vee \; x \; \vee \; y\,) \wedge (a_L \vee \neg x_L \vee \neg a \vee \; \neg y\,) = \top$$

Boolean Equations and Combinational Synthesis

$\{a\} \rightarrow$ set of inputs $\quad \{x, y\} \rightarrow$ set of outputs
$\{\mathsf{G}((a \wedge \neg y) \rightarrow \mathsf{X}(x \vee y)), \mathsf{G}((\neg a \wedge x \wedge \mathsf{X}\, a) \rightarrow \mathsf{X}\, \neg y)\}$
is the set of safety properties

$$R = (\neg a_L \vee y_L \vee x \vee y) \wedge (a_L \vee \neg x_L \vee \neg a \vee \neg y)$$

Boolean Equations and Combinational Synthesis

$\{a\} \rightarrow$ set of inputs $\qquad \{x, y\} \rightarrow$ set of outputs
$\{\mathsf{G}((a \wedge \neg y) \rightarrow \mathsf{X}(x \vee y)), \mathsf{G}((\neg a \wedge x \wedge \mathsf{X}\, a) \rightarrow \mathsf{X}\, \neg y)\}$
is the set of safety properties

$$R = (\neg a_L \vee y_L \vee x \vee y) \wedge (a_L \vee \neg x_L \vee \neg a \vee \neg y)$$

$x = x_i$
$y = (a_L \wedge \neg y_L) \wedge x_i \vee (a_L \vee \neg x_L \vee a) \wedge y_i$

LTL and $\mathcal{R}$-generable Languages

Languages described by certain LTL properties can be identified as $\mathcal{R}$-generable without constructing the corresponding automaton.

E.g. $\mathsf{G}(a \rightarrow \mathsf{X}\, x)$ or $G((a \vee \mathsf{X}\, b) \leftrightarrow \mathsf{X}\, x)$

$\mathsf{G}(a \rightarrow \mathsf{X}\,\mathsf{X}\, y)$ does not describe an $\mathcal{R}$-generable language.

This syntactic characterization is sufficient but not necessary.

E.g. $\mathsf{G}(r \rightarrow (r\, \mathsf{W}\, g))$

$\mathcal{R}$-generable languages are those that only need to remember the previous letter.

LTL and $\mathcal{R}$-generable Languages

Languages described by certain LTL properties can be identified as $\mathcal{R}$-generable without constructing the corresponding automaton.

E.g. $\mathsf{G}(a \rightarrow \mathsf{X}\, x)$ or $G((a \vee \mathsf{X}\, b) \leftrightarrow \mathsf{X}\, x)$

$\mathsf{G}(a \rightarrow \mathsf{X}\,\mathsf{X}\, y)$ does not describe an $\mathcal{R}$-generable language.

This syntactic characterization is sufficient but not necessary.

E.g. $\mathsf{G}(r \rightarrow (r\, \mathsf{W}\, g))$

$\mathcal{R}$-generable languages are those that only need to remember the previous letter.

LTL and $\mathcal{R}$-generable Languages

Languages described by certain LTL properties can be identified as $\mathcal{R}$-generable without constructing the corresponding automaton.

E.g. $\mathsf{G}(a \rightarrow \mathsf{X}\, x)$ or $G((a \vee \mathsf{X}\, b) \leftrightarrow \mathsf{X}\, x)$

$\mathsf{G}(a \rightarrow \mathsf{X}\,\mathsf{X}\, y)$ does not describe an $\mathcal{R}$-generable language.

This syntactic characterization is sufficient but not necessary.

E.g. $\mathsf{G}(r \rightarrow (r\, \mathsf{W}\, g))$

$\mathcal{R}$-generable languages are those that only need to remember the previous letter.

LTL and $\mathcal{R}$-generable Languages

Languages described by certain LTL properties can be identified as $\mathcal{R}$-generable without constructing the corresponding automaton.

E.g. $\mathsf{G}(a \rightarrow \mathsf{X}\,x)$ or $G((a \vee \mathsf{X}\,b) \leftrightarrow \mathsf{X}\,x)$

$\mathsf{G}(a \rightarrow \mathsf{X}\,\mathsf{X}\,y)$ does not describe an $\mathcal{R}$-generable language.

This syntactic characterization is sufficient but not necessary.

E.g. $\mathsf{G}(r \rightarrow (r\,\mathsf{W}\,g))$

$\mathcal{R}$-generable languages are those that only need to remember the previous letter.

LTL and $\mathcal{R}$-generable Languages

Languages described by certain LTL properties can be identified as $\mathcal{R}$-generable without constructing the corresponding automaton.

E.g. $\mathsf{G}(a \rightarrow \mathsf{X}\, x)$ or $G((a \vee \mathsf{X}\, b) \leftrightarrow \mathsf{X}\, x)$

$\mathsf{G}(a \rightarrow \mathsf{X}\,\mathsf{X}\, y)$ does not describe an $\mathcal{R}$-generable language.

This syntactic characterization is sufficient but not necessary.

E.g. $\mathsf{G}(r \rightarrow (r\, \mathsf{W}\, g))$

$\mathcal{R}$-generable languages are those that only need to remember the previous letter.

LTL and $\mathcal{R}$-generable Languages

Languages described by certain LTL properties can be identified as $\mathcal{R}$-generable without constructing the corresponding automaton.

E.g. $\mathsf{G}(a \rightarrow \mathsf{X}\, x)$ or $G((a \vee \mathsf{X}\, b) \leftrightarrow \mathsf{X}\, x)$

$\mathsf{G}(a \rightarrow \mathsf{X}\,\mathsf{X}\, y)$ does not describe an $\mathcal{R}$-generable language.

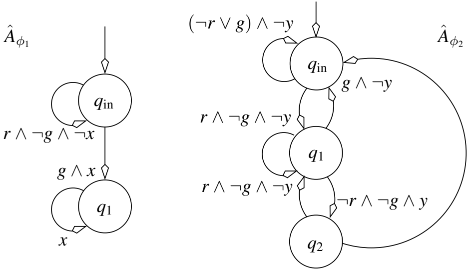
This syntactic characterization is sufficient but not necessary.

E.g. $\mathsf{G}(r \rightarrow (r\,\mathsf{W}\, g))$

$\mathcal{R}$-generable languages are those that only need to remember the previous letter.

Optimal augmentation of the alphabet

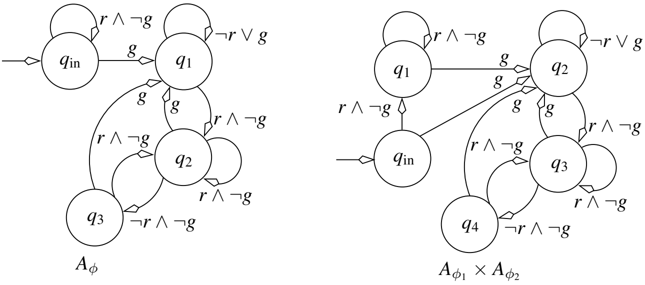
Augmenting the alphabet of individual properties may not be the optimal strategy.



$$\phi_1 = r \,\mathsf{W}\, g$$
$$\phi_2 = \mathsf{G}(r \wedge \neg g \rightarrow \mathsf{X}(r \vee g \vee \mathsf{X}(r \vee g)))$$

Optimal augmentation of the alphabet

Augmenting the alphabet of individual properties may not be the optimal strategy.



After generating the automaton for ϕ or composing the automata $A_{\phi_1} \times A_{\phi_2}$ it becomes clear that the alphabet needed to be augmented by only two letters.

Retiming

$\{a, x_i, y_i\} \rightarrow$ set of inputs
$\{a_L, x_L, y_L\} \rightarrow$ set of memory elements
$\{x, y\} \rightarrow$ set of outputs

Retiming

$\{a, x_i, y_i\} \rightarrow$ set of inputs
$\{a_L, x_L, y_L\} \rightarrow$ set of memory elements
$\{x, y\} \rightarrow$ set of outputs

$x = x_i$
$y = (\ a_L \wedge \neg y_L\ ) \wedge x_i \vee (\ a_L \vee \neg x_L\ \vee a) \wedge y_i$

Retiming

$\{a, x_i, y_i\} \rightarrow$ set of inputs
$\{a_L, x_L, y_L\} \rightarrow$ set of memory elements
$\{x, y\} \rightarrow$ set of outputs

$x = x_i$
$y = (a_L \wedge \neg y_L) \wedge x_i \vee (a_L \vee \neg x_L \vee a) \wedge y_i$

Retiming

$\{a, x_i, y_i\} \rightarrow$ set of inputs
$\{a_L, x_L, y_L\} \rightarrow$ set of memory elements
$\{x, y\} \rightarrow$ set of outputs

$x = x_i$
$y = (a_L \wedge \neg y_L) \wedge x_i \vee (a_L \vee \neg x_L \vee a) \wedge y_i$

$\{m_1, m_2\} \rightarrow$ set of memory elements where

$$m_1 = a \wedge \neg y \qquad m_2 = (a \vee \neg x)$$

$y = m_1 \wedge x_i \vee (m_2 \vee a) \wedge y_i$

Retiming. . . (continued)

The efficiency of retiming heuristic is dependant on the factorization of the function.