What’s new in ltl synt?

Florian Renkin, Philipp Schlehuber, Alexandre Duret-Lutz, Adrien Pommellet

SYNT’21
Basic Workflow of ltl.synt

1. **LTL input**
2. Convert to parity game
   - `--algo=ds|sd|ps|lar`
3. Solve game
4. Encode to AIGER

Ad hoc construction for formulas of the form

- `G(b_1) ∧ (ϕ ↔ G F b_2)`

Inspired from delag improvements over Later Appearance Records

Turn

- `i_1 ∧ i_2 ∧ o_1 ∧ o_2`

Into

- `i_1 ∧ i_2 ∧ o_1 ∧ o_2`

Separate specification as

- `ϕ_1 ∧ ϕ_2 ∧ ϕ_3 ∧ ...`

Where each `ϕ_i` has unique output variables;

Process each `ϕ_i` separately

Merge strategies during encoding (possible sharing of gates above inputs)

Multiple methods:

1. Bisimulation-based (fast, coarse)
2. Bisimulation with output assignments
3. SAT-based (slow, precise)

Experimentation with multiple encodings

Strategy easily obtained if a DBA for `ϕ` is known;

Used in 103/945 cases in SYNTCOMP 2021
Basic Workflow of ltl2synt

1. **LTL input**
2. Translate to DELA
3. Paritize to DPA
4. Split In/Out
5. Solve game
6. Minimize strategy
7. Encode AIGER
8. Convert to parity game

---

--algo=lar

**Inspired from delag**

- **Later Appearance Records**
- Turn
- $i_1 \land i_2 \lor o_1 \land o_2$
- Separate specification as
  - $\phi_1 \land \phi_2 \land \phi_3 \land \ldots$
  - Where each $\phi_i$ has unique output variables.

**Multiple methods:**
1. Bisimulation-based (fast, coarse)
2. Bisimulation with output assignments
3. SAT-based (slow, precise)

**Experimentation with multiple encodings**

**Strategy easily obtained if a DBA for $\phi$ is known:**

- Used in 103/945 cases in SYNTCOMP 2021 by Müller and Sickert. LTL to deterministic Emerson-Lei automata. *GandALF’17*
Basic Workflow of ltl.synt

LTL input

--algo=lar

translate to DELA

paritize to DPA

split In/Out

solve game

encode AIGER

improvements over Later Appearance Records

Basic Workflow of ltlssynt

1. **LTL input**
2. Translate to DELA
3. Paritize to DPA
4. Split In/Out
5. Solve game
6. Encode AIGER
7. --algo=lar
8. AIGER output

---

- **turn** $i_1 \land i_2 \land o_1 \land o_2$ into $i_1 \land i_2$ $\square o_1 \land o_2$
- Ad hoc construction for formulas of the form $G(b_1) \land (\phi \leftrightarrow G F b_2)$
- --algo=lar inspired from delag
- Improvements over Later Appearance Records
- Turn $i_1 \land i_2 \land o_1 \land o_2$ into $i_1 \land i_2$ $\land o_1 \land o_2$
- Separate specification as $\phi_1 \land \phi_2 \land \phi_3 \land \ldots$
- Where each $\phi_i$ has unique output variables
- Process each $\phi_i$ separately
- Merge strategies during encoding (possible sharing of gates above inputs)
- Multiple methods:
  1. Bisimulation-based (fast, coarse) 
  2. Bisimulation with output assignments
  3. SAT-based (slow, precise)
- Experimentation with multiple encodings
- Strategy easily obtained if a DBA for $\phi$ is known
- Used in 103/945 cases in SYNTCOMP 2021
Basic Workflow of ltlisynt

1. **LTL input**
2. **LTL decomp.**
3. **translate to DELA**
4. **paritize to DPA**
5. **split In/Out**
6. **solve game**
7. **encode AIGER**

**AIGER output**

---

**separate specification as** $\varphi_1 \land \varphi_2 \land \varphi_3 \land \cdots$ **where each** $\varphi_i$ **has unique output variables; process each** $\varphi_i$ **separately**

**merge strategies during encoding (possible sharing of gates above inputs)**

---

Finkbeiner, Geier, and Passing. Specification decomposition for reactive synthesis. *NFM'21*
Basic Workflow of ltltsynt

LTL input → LTL decomp. → translate to DELA → paritize to DPA → split In/Out → solve game → ad hoc construction for formulas of the form $G(b_1) \land (\varphi \leftrightarrow GFb_2)$ → encode AIGER

strategy easily obtained if a DBA for $\varphi$ is known; used in 103/945 cases in SYNTCOMP 2021
Strategy Minimization

LTL input → LTL decomp. → translate to DELA → paritize to DPA → split In/Out → solve game → minimize strategy → encode AIGER

ad hoc construction for formulas of the form $G(b_1) \land (\varphi \leftrightarrow GF b_2)$

multiple methods:
1. bisimulation-based (fast, coarse)
2. bisimulation w/ output assignments
3. SAT-based (slow, precise)
Minimizing Incompletely Specified Mealy Machines

The diagram illustrates a state transition system with the following states: 0, 1, 2, 3, 4, 5, and 6. The transitions are labeled with input/output pairs:

- From state 0:
  - Input 0, Output 000
- From state 1:
  - Input 1, Output 0
  - Input 0, Output 0
- From state 2:
  - Input 0, Output 1
- From state 3:
  - Input 0, Output --/---
- From state 4:
  - Input 1, Output 0
- From state 5:
  - Input 0, Output 1
  - Input 1, Output 1
Minimizing ISMM with Bisimulation

![Diagram showing a state transition graph with states labeled 0, 1, 3, 4, 5, and 6. The transitions include arrows labeled with transitions like 0->0, 1->1, and 0->1. The states and transitions are depicted with circles and directed arrows.]
Minimizing ISMM with Bisimulation

\[ \begin{array}{l}
\text{0} \\
\text{1} \\
\text{2} \\
\text{3} \\
\text{4} \\
\text{5} \\
\text{6}
\end{array} \]
let’s assign “don’t care” outputs to improve bisimulation quotient
Minimizing ISMM with Bisimulation and Output Assignment

Let's assign “don’t care” outputs to improve bisimulation quotient.
Minimizing ISMM with Bisimulation and Output Assignment

let's assign "don't care" outputs to improve bisimulation quotient
Minimizing ISMM with Bisimulation and Output Assignment

Let's assign "don't care" outputs to improve bisimulation quotient.
Minimizing ISMM with SAT (Similar to MeMin)

Abel and Reineke. MeMin: SAT-based exact minimization of incompletely specified Mealy machines. *ICCAD’15*
Minimizing ISMM with SAT (Similar to MeMin)

[Diagram of a state transition graph with states and transitions labeled with input-output pairs]

Abel and Reineke. MeMin: SAT-based exact minimization of incompletely specified Mealy machines. *ICCAD’15*
Benchmarks

![Graph showing runtime vs. number of problems]

- # Problems: 530, 535, 540, 545, 550, 555, 560
- Runtime in seconds (sec)
- Log scale for sec
- Log scale for # Problems

- MeMin
- bisim.
- bisim. w/ o.a.
- SAT
- bisim. + SAT
- bisim. w/ o.a. + SAT

Graph indicates the performance of different benchmarking techniques across varying numbers of problems, with MeMin and SAT showing lower runtime compared to others.
<table>
<thead>
<tr>
<th>method</th>
<th>#solved</th>
<th>#minimal</th>
<th>size ratio of non-minimal cases</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>mean</td>
</tr>
<tr>
<td>no reduction</td>
<td>560</td>
<td>63%</td>
<td>9.05</td>
</tr>
<tr>
<td>bisimulation</td>
<td>560</td>
<td>76%</td>
<td>1.73</td>
</tr>
<tr>
<td>bisim. w/ output ass.</td>
<td>559</td>
<td>96%</td>
<td>1.39</td>
</tr>
<tr>
<td>SAT</td>
<td>557</td>
<td>100%</td>
<td></td>
</tr>
<tr>
<td>bisimulation + SAT</td>
<td>558</td>
<td>100%</td>
<td></td>
</tr>
<tr>
<td>bisim. w/ output ass. + SAT</td>
<td>559</td>
<td>97%</td>
<td>1.18</td>
</tr>
<tr>
<td>MeMin</td>
<td>536</td>
<td>100%</td>
<td></td>
</tr>
</tbody>
</table>
Differences to MeMin

- Using BDDs instead of Cubes to label edges
  More expressive edge labels e.g. $1 - \{10, 01\}$
- Improved usage of a priori knowledge about the solution
AIGER Encoding

1. LTL input
2. LTL decomp.
3. translate to DELA
4. paritize to DPA
5. split In/Out
6. solve game
7. minimize strategy
8. encode AIGER

Ad hoc construction for formulas of the form $\text{G}(b_1) \land (\varphi \leftrightarrow \text{G F} b_2)$

- Inspired from delag improvements over Later Appearance Records
- Turn $i_1 \land i_2 \land o_1 \land o_2$ into $i_1 \land i_2 \land o_1 \land o_2$
- Separate specification as $\varphi_1 \land \varphi_2 \land \varphi_3 \land \ldots$
  - Where each $\varphi_i$ has unique output variables;
  - Process each $\varphi_i$ separately;
  - Merge strategies during encoding (possible sharing of gates above inputs)

Multiple methods:
1. Bisimulation-based (fast, coarse)
2. Bisimulation w/ output assignments
3. SAT-based (slow, precise)

Experimentation with multiple encodings

Strategy easily obtained if a DBA for $\varphi$ is known;
- Used in 103/945 cases in SYNTCOMP 2021
Testing Different BDD to Aiger Encodings

Conditions are represented by BDDs and translated into circuits
Conditions are represented by BDDs and translated into circuits

- If-then-else form (ITE):
  \[ f = (i_1 \land f_{i_1}) \lor (\overline{i_1} \land f_{\overline{i_1}}) \]
Conditions are represented by BDDs and translated into circuits

- If-then-else form (ITE):
  \[ f = (i_1 \land f_{i_1}) \lor (\overline{i_1} \land f_{\overline{i_1}}) \]

- Irredundant sum of products (ISOP):
  \[ f = f_1 \lor f_2 \lor f_3 \lor f_4 \]
Testing Different BDD to Aiger Encodings

Conditions are represented by BDDs and translated into circuits

- **If-then-else form (ITE):**
  \[ f = (i_1 \land f_{i_1}) \lor (\bar{i}_1 \land f_{\bar{i}_1}) \]

- **Irredundant sum of products (ISOP):**
  \[ f = f_1 \lor f_2 \lor f_3 \lor f_4 \]

- If \( f_1 \) and \( f_3 \) appear frequently together in the strategy, reorder inputs to improve sharing (OPTIM)
Conditions are represented by BDDs and translated into circuits

- **If-then-else form (ITE):**
  \[ f = (i_1 \land f_{i_1}) \lor (\bar{i}_1 \land f_{\bar{i}_1}) \]

- **Irredundant sum of products (ISOP):**
  \[ f = f_1 \lor f_2 \lor f_3 \lor f_4 \]

- **If** \( f_1 \) and \( f_3 \) appear frequently together in the strategy, reorder inputs to improve sharing (**OPTIM**)

- **Encode** \( f \) and \( \bar{f} \) and keep the smallest circuit (**DUAL**)

---

---

---
Benchmarking Aiger Encodings

BEST = ITE + DUAL | ISOP + DUAL

total encoding time [s]
ITE 1.0
ITE + DUAL 1.6
ISOP 2.7
ISOP + DUAL 5.1
BEST 6.2
OPTIM 4170
Conclusion

LTL input

- LTL decomp.
- Ad hoc construction for formulas of the form $G(b_1) \land (\varphi \leftrightarrow GF b_2)$
- Translate to DELA
- Paritize to DPA
- Split In/Out
- Solve game
- Minimize strategy
- Encode AIGER

AIGER output

- New in 2021
- Improved in 2021

* : new in 2021
\* : improved in 2021