Formalization of natural language requirements into temporal logics: a survey

Igor Buzhinsky
igor.buzhinskii@aalto.fi

INDIN 2019 conference

24.07.2019, Espoo
Motivation: nuclear power plant I&C verification in Finland

- In Finland, a formal verification method called **model checking** is used to ensure the safety of nuclear power plants.
Motivation: nuclear power plant I&C verification in Finland

- In Finland, a **formal verification** method called **model checking** is used to ensure the safety of nuclear power plants.
- Specifically, VTT model-checks instrumentation and control (I&C) systems for Finnish power utilities.
Applying model checking

System (e.g. power plant I&C system)

Formal model (e.g. in NuSMV)

Real-world artifacts

Real-world artifacts

Formal artifacts

```
SW: BINARY_SWITCH(SWITCH, SWITCH ? MINUS.ADDER_OUTPUT_SIGN, MINUS.ADI
PLUS: ADDER_3(0, FALSE, FALSE, ING
MINUS: ADDER_3(0, FALSE, FALSE, IN
OUT := SW.BSWITCH_OUTP_SIGN;
```
Applying model checking

"If the pressurizer water level rises above $l_0$, then the reactor shall be tripped (i.e., shut down) on the next cycle at latest"

System (e.g. power plant I&C system)

Formal model (e.g. in NuSMV)

```
SW: BINARY_SWITCH(SWITCH, SWITCH_F
MINUS.ADDER_OUTPUT_SIGN, MINUS.ADI
PLUS: ADDER_3(0, FALSE, FALSE, ING
MINUS: ADDER_3(0, FALSE, FALSE, IN
OUT := SW.BSWITCH_OUTP_SIGN;
```

"G(($l > l_0$) → ($t \lor X t$))"

Formal requirement (e.g. in LTL logic)

Natural language (NL) requirement

Real-world artifacts

Formal artifacts
Applying model checking

“If the pressurizer water level rises above \( l_0 \), then the reactor shall be tripped (i.e., shut down) on the next cycle at latest”

\[ G((l > l_0) \rightarrow (t \lor X t)) \]

Automatic, thorough model checking in a model checker (e.g. NuSMV)

System (e.g. power plant I&C system)

Formal model (e.g. in NuSMV)

Natual language (NL) requirement

Formal requirement (e.g. in LTL logic)

Real-world artifacts

Formal artifacts
Applying model checking

“If the pressurizer water level rises above $l_0$, then the reactor shall be tripped (i.e., shut down) on the next cycle at latest”

G((l > l_0) → (t ∨ X t))

Our focus

Automatic, thorough model checking in a model checker (e.g. NuSMV)

Real-world artifacts

Formal artifacts

System (e.g. power plant I&C system)

Formal model (e.g. in NuSMV)

Natural language (NL) requirement

Formal requirement (e.g. in LTL logic)
Example: linear temporal logic (LTL)

If the pressurizer water level falls below $l_0$, then it shall eventually exceed this value.

Always (Globally) Eventually (in the Future)

- Extension of the Boolean propositional logic
- Can inquire the state of the model at different time instants
- Temporal operators
  - $G$: always (Globally)
  - $F$: eventually, at some time instant (in the Future)
  - $X$: at the next time instant
Example: linear temporal logic (LTL)

If the pressurizer water level falls below \( l_0 \), then it shall eventually exceed this value.

Always (Globally) Eventually (in the Future)

\[
G \left( ( l < l_0 ) \rightarrow F \left( l > l_0 \right) \right)
\]

- Extension of the Boolean propositional logic
- Can inquire the state of the model at different time instants
- Temporal operators
  - \( G \): always (Globally)
  - \( F \): eventually, at some time instant (in the Future)
  - \( X \): at the next time instant
- Not obvious how to generate from NL requirements
Pattern-based approaches
What is a (temporal) requirement pattern

**Pattern:** Response

**Intent:** To describe *cause-effect relationships* between a pair of events/states. An occurrence of the first, *the cause*, must be followed by an occurrence of the second, *the effect*, within a defined portion of a system’s execution. Also known as *Follows* and *Leads-to*.

**LTL** (the simplest case):
“Globally, q responds to p”: \(G(p \rightarrow Fq)\)

**Example:** “If the pressurizer water level falls below \(l_0\), then it shall eventually exceed this value”.

Patterns in practice

Boilerplates are NL templates with placeholders, such as:

- Example: “In subsystem _____, condition _____ always leads to action _____”
- Can directly map to LTL
Patterns in practice

Boilerplates are NL templates with placeholders, such as:

- Example: “In subsystem _____, condition _____ always leads to action _____”
- Can directly map to LTL

Structured English grammars [1, 2] can recognize a predefined set of frequent patterns

- Example [1]: “… It is at all times possible that all elements of set A occur simultaneously within the interval [x, y]…”
- Again, can directly map to LTL

Translation based on formal grammars
Formal grammars and parse trees

- **Terminals** = actual characters of a string
- **Non-terminals** are more high-level concepts (like *binary operator*)

G ( ( l < l₀ ) → F ( l > l₀ ) )
Formal grammars and parse trees

Example of a production rule:
\[ \text{Expr} \rightarrow \text{Expr} \text{ BinOp } \text{Expr} \]

- **Terminals** = actual characters of a string
- **Non-terminals** are more high-level concepts (like *binary operator*)
- **A formal grammar** is essentially a set of production rules
General ideas behind grammar-based approaches

General approach

- Represent NL as a grammar
- Define how this parse tree is converted to the one of a temporal logic
- Alternatively, can just define rules to extract fixed temporal property patterns (already covered)
General ideas behind grammar-based approaches

General approach

- Represent NL as a grammar
- Define how this parse tree is converted to the one of a temporal logic
- Alternatively, can just define rules to extract fixed temporal property patterns (already covered)

Problems

- **Ambiguity** of NL
- **Variability** of NL: potentially many ways to express the same requirement
- Need to **restructure** NL sentences to generate temporal formulas
Approach: recursive temporal logic representation in NL

State formulas

\[ p \rightarrow \text{true} \]
\[ \rightarrow \text{false} \]
\[ \rightarrow \neg p \]
\[ \rightarrow p \land p \]
\[ \rightarrow p \lor p \]
\[ \rightarrow \exists q \]
\[ \rightarrow \forall q \]
\[ \rightarrow <\text{ACTION}> p \]
\[ \rightarrow [\text{ACTION}] p \]

Path formulas

\[ q \rightarrow [p \cup \{t\} \cup \{t'\} p'] \]
\[ \rightarrow G p \]

Action formulas

\[ r \rightarrow \text{true} \]
\[ \rightarrow \text{false} \]
\[ \rightarrow \text{ACTION} \]
\[ \rightarrow \neg p \]
\[ \rightarrow r \mid r \]
### Approach: recursive temporal logic representation in NL


<table>
<thead>
<tr>
<th>State formulas</th>
<th>Path formulas</th>
<th>Action formulas</th>
</tr>
</thead>
<tbody>
<tr>
<td>p → true</td>
<td>q → [p {r} U {r'} p']</td>
<td>r → true</td>
</tr>
<tr>
<td>→ false</td>
<td>&quot;there exists a path in which q&quot;</td>
<td>&quot;any action&quot;</td>
</tr>
<tr>
<td>→ ¬p</td>
<td>&quot;there exists a next state reachable with ACTION, in which p&quot;</td>
<td>&quot;no action&quot;</td>
</tr>
<tr>
<td>→ p &amp; p</td>
<td>&quot;for all next states reachable with ACTION, p&quot;</td>
<td>&quot;not p&quot;</td>
</tr>
<tr>
<td>→ p</td>
<td>&quot;there exists a path in which q&quot;</td>
<td>&quot;r or p&quot;</td>
</tr>
<tr>
<td>→ E q</td>
<td>&quot;for all paths q&quot;</td>
<td>&quot;r or p&quot;</td>
</tr>
<tr>
<td>→ A q</td>
<td>&quot;there exists a next state reachable with ACTION, in which p&quot;</td>
<td>&quot;r or p&quot;</td>
</tr>
</tbody>
</table>

One-to-one mapping between temporal logic ACTL and predefined NL text
Approach: a grammar for robot control tasks


- **EnvInit** ::= ‘Environment starts with (ϕ_{env} | false | true)’
- **RobotInit** ::= ‘Robot starts [in ϕ_{region}] [with ϕ_{action} | with false | with true]’
- **EnvSafety** ::= ‘Always ϕ_{env}’
- **RobotSafety** ::= ‘(Always | Always do | Do) ϕ_{robot}’
- **EnvLiveness** ::= ‘Infinitely often ϕ_{env}’
- **RobotLiveness** ::= ‘(Go to | Visit | Infinitely often do) ϕ_{robot}’
- **RobotGoStay** ::= ‘Go to ϕ_{region} and stay [there]’
- **Conditional** ::= ‘If Condition then Requirement ’ | ‘Requirement unless Condition’ | ‘Requirement if and only if Condition’
- **Condition** ::= ‘Condition and Condition’ | ‘Condition or Condition’ | ‘you (were | are) [not] in ϕ_{region}’ | ‘you (sensed | did not sense | are [not] sensing) ϕ_{env}’ | ‘you (activated | did not activate | are [not] activating) ϕ_{action}’
- **Requirement** ::= EnvSafety | RobotSafety | EnvLiveness | RobotLiveness | ‘stay [there]’
Approach: a grammar for robot control tasks


\[
\text{EnvInit} ::= \text{‘Environment starts with } (\phi_{env} \text{ | false | true)}’ \\
\text{RobotInit} ::= \text{‘Robot starts [in } \phi_{region} \text{] [with } \phi_{action} \text{ | with false | with true]’} \\
\text{EnvSafety} ::= \text{‘Always } \phi_{env}’ \\
\text{RobotSafety} ::= \text{‘(Always | Always do | Do) } \phi_{robot}’ \\
\text{EnvLiveness} ::= \text{‘Infinitely often } \phi_{env}’ \\
\text{RobotLiveness} ::= \text{‘(Go to | Visit | Infinitely often do) } \phi_{robot}’ \\
\text{RobotGoStay} ::= \text{‘Go to } \phi_{region} \text{ and stay [there]’} \\
\text{Conditional} ::= \text{‘If Condition then Requirement’ | ‘Requirement unless Condition’ | ‘Requirement if and only if Condition’} \\
\text{Condition} ::= \text{‘Condition and Condition’ | ‘Condition or Condition’ | ‘you (were | are) [not] in } \phi_{region}’ | \text{‘you (sensed | did not sense | are [not] sensing) } \phi_{env}’ | \text{‘you (activated | did not activate | are [not] activating) } \phi_{action}’ | \\
\text{Requirement} ::= \text{EnvSafety | RobotSafety | EnvLiveness | RobotLiveness | ‘stay [there]’} \]

- Concrete domain
- Limited recursion
- Mapped to LTL
Approach: grammar learning based on a corpus

C.B. Harris, I.G. Harris. GLAsT: Learning formal grammars to translate natural language specifications into hardware assertions. *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2016, pp. 966–971

Target notation: SystemVerilog Assertions

- `assert property(@(posedge clk) module1.rst == 1);` — “Reset should be high”
Approach: grammar learning based on a corpus

C.B. Harris, I.G. Harris. GLAsT: Learning formal grammars to translate natural language specifications into hardware assertions. *Design, Automation & Test in Europe Conference & Exhibition (DATE).* IEEE, 2016, pp. 966–971

Target notation: SystemVerilog Assertions

- `assert property(@(posedge clk) 
  module1.rst == 1);` – “Reset should be high”

Search in the space of formal grammars

- **Smaller** grammars are preferred
- Initially a verbose grammar based on the learning set
- Beam search to find the smallest grammars
Approaches based on parse trees
(start from NL, not formal grammars)
Example: Stanford NL parser (1)

Stanford Parser

Please enter a sentence to be parsed:
If water level becomes high, then the reactor shall be tripped

Language: English  Sample Sentence

Your query

If water level becomes high, then the reactor shall be tripped

Tagging

If/IN water/NN level/NN becomes/VBZ high/JJ ,/, then/RB the/DT reactor/NN shall/MD be/VB tripped/VBN
Example: Stanford NL parser (2)

Sentence: “If water level becomes high, then the reactor shall be tripped”

Parse tree:

(Root
(S
(SBAR (IN If)
  (S
    (NP (NN water) (NN level))
    (VP (VBZ becomes)
        (ADJP (JJ high)))))
  (,) )
(NP (RB then) (DT the) (NN reactor))
(VP (MD shall)
  (VP (VB be)
    (VP (VBN tripped))))
Example: Stanford NL parser (3)

Sentence: “If water level becomes high, then the reactor shall be tripped”

Universal dependencies (enhanced):

mark(becomes-4, If-1)
compound(level-3, water-2)
nsubj(becomes-4, level-3)
advcl(tripped-12, becomes-4)
xcomp(becomes-4, high-5)
advmod(reactor-9, then-7)
det(reactor-9, the-8)
nsubjpass(tripped-12, reactor-9)
aux(tripped-12, shall-10)
auxpass(tripped-12, be-11)
root(ROOT-0, tripped-12)
Approach: ARSENAL, stage 1: get word dependencies

Dependency tree for requirement:
“If the Status attribute of the Lower Desired Temperature is invalid, the Regulator Interface Failure shall be set to True”

Approach: ARSENAL, stage 2: apply customizable type rules

Predicate graph for requirement:
“If the Status attribute of the Lower Desired Temperature is invalid, the Regulator Interface Failure shall be set to True”
Approach: ARSENAL, stage 3: formula generation

- Formula generation based on the predicate graph
- **Output adapters** for different logics
- Our requirement: “If the Status attribute of the Lower Desired Temperature is invalid, the Regulator Interface Failure shall be set to True”
- Generated LTL formula:
  \[ G((\text{Lower} \_\text{Desired} \_\text{Temperature}.\text{Status} \_\text{attribute} = \text{Invalid}) \rightarrow (\text{RegulatorInterfaceFailure} = \text{TRUE})) \]
Statistical machine translation
Statistical machine translation applied to our problem (1)

- Translate language $f$ (NL requirements) to $e$ (temporal logic)
- Bayes rule:
  \[ p(e|f) = \frac{p(f|e)p(e)}{p(f)} \]

- $p(e)$ is the **language model**
- $p(f|e)$ is the **translation model**
- $p(f)$ can be ignored since we search $\text{argmax}_e p(e|f)$
• Translate language \( f \) (NL requirements) to \( e \) (temporal logic)

• Bayes rule:

\[
p(e|f) = \frac{p(f|e)p(e)}{p(f)}, \text{ where}
\]

• \( p(e) \) is the **language model**

• \( p(f|e) \) is the **translation model**

• \( p(f) \) can be ignored since we search \( \text{argmax}_e p(e|f) \)

• \( n \)-gram approach: estimate token probabilities based on last \( n - 1 \) words, using a corpus

• For example, 3-gram language model \( p(e) \):

\[
G \ (x \rightarrow \ldots \text{“y”}? \quad p(\text{“y”} \mid \text{“x”, “→”}) = 0.25
\]

\[
\text{“F”}? \quad p(\text{“F”} \mid \text{“x”, “→”}) = 0.15
\]

\[
\text{“)”}? \quad p(\text{“)”} \mid \text{“x”, “→”}) = 0.00
\]
Basic improvements

- No need for a parallel corpus to train a language model!
- Phrase-based translation
- Alignment (word reordering) models
Basic improvements

- No need for a parallel corpus to train a language model!
- Phrase-based translation
- Alignment (word reordering) models

Difficulties in our case

- Need a large parallel corpus to train the translation model
- We can tolerate imperfect NL translation, but temporal formulas need to be precise
- This means that much manual work will be needed to fix the generated temporal formulas
- But we intended to avoid it!
Neural machine translation
(everybody uses deep learning, why not try?)
Example of a neural translation architecture

Input word embeddings
Left → Right Recurrent NN
Right → Left Recurrent NN
Attention
Input context
Hidden state
Output word predictions
Error
Given output word
Output word embeddings

Comparison of neural and statistical phrase-based translation

Intersections at circa 15 million and 150 million words
(English → Spanish translation)

Conclusions
Conclusions

- So far, only partial solutions to the problem of requirements formalization into temporal logics exist
- Promising approaches
  - C.B. Harris, I.G. Harris. GLAsT: Learning formal grammars to translate natural language specifications into hardware assertions. *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2016, pp. 966–971
- Statistical and neural (deep learning) translation approaches won’t produce adequate results due to the lack of training data
Questions?

Thank you for your attention!

Igor Buzhinsky
igor.buzhinskii@aalto.fi

Formalization of natural language requirements into temporal logics: a survey

24.07.2019, Espoo