Formalization of natural language requirements into temporal logics: a survey

Igor Buzhinsky

igor.buzhinskii@aalto.fi

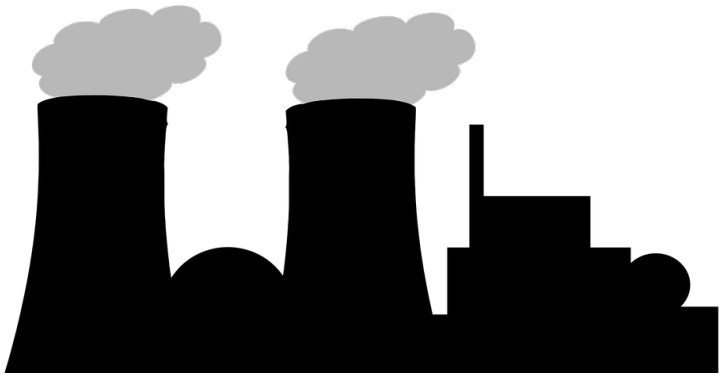
INDIN 2019 conference

Aalto University
School of Electrical Engineering

ITMO UNIVERSITY

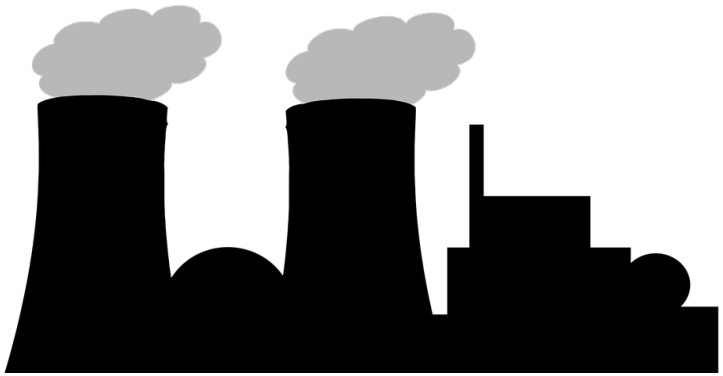
24.07.2019, Espoo

Motivation: nuclear power plant I&C verification in Finland

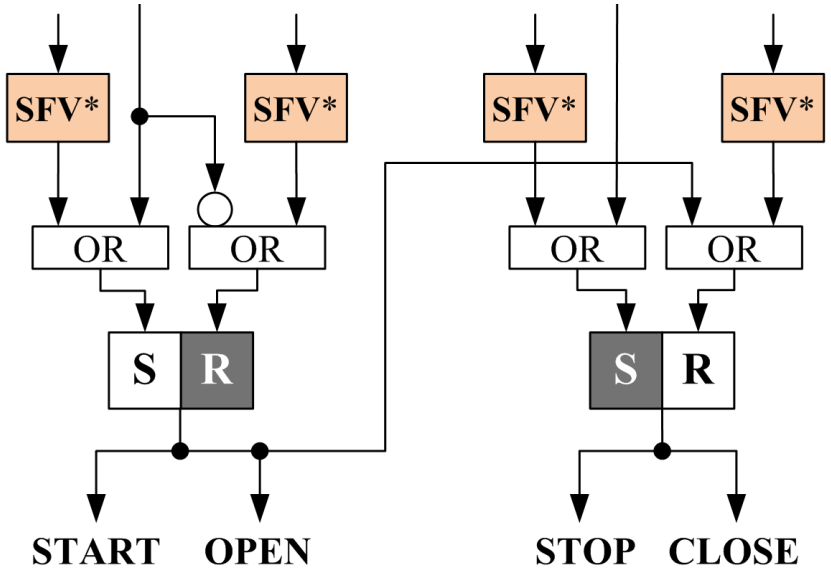


- In Finland, a **formal verification** method called **model checking** is used to ensure the safety of nuclear power plants

Motivation: nuclear power plant I&C verification in Finland

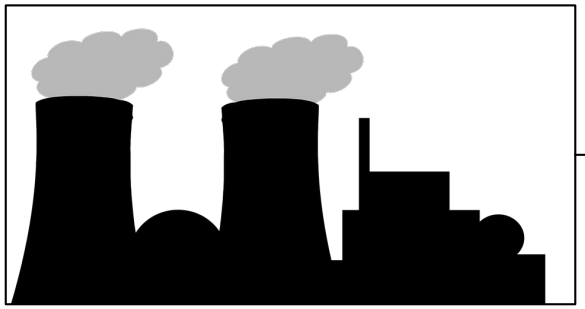


- In Finland, a **formal verification** method called **model checking** is used to ensure the safety of nuclear power plants
- Specifically, VTT model-checks **instrumentation and control (I&C) systems** for Finnish power utilities



Applying model checking

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



Formal model (e.g. in NuSMV)

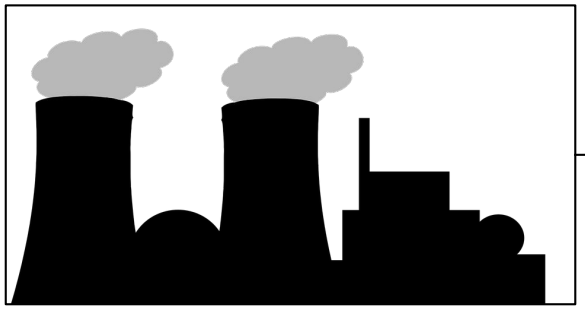
```
SW: BINARY_SWITCH(SWITCH, SWITCH_F
MINUS.ADDER_OUTPUT_SIGN, MINUS.ADD
PLUS: ADDER_3(0, FALSE, FALSE, IN0
MINUS: ADDER_3(0, FALSE, FALSE, IN
OUT := SW.BSWITCH_OUTP_SIGN;
```

Real-world artifacts

Formal artifacts

Applying model checking

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



"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"

Natural language (NL) requirement

Formal model (e.g. in NuSMV)

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

$$\mathbf{G}((l > l_0) \rightarrow (t \vee \mathbf{X}\, t))$$

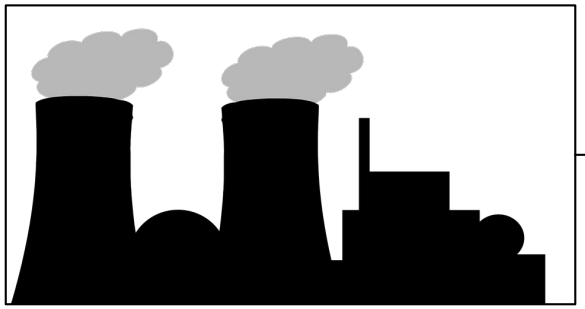
Formal requirement (e.g. in LTL logic)

Real-world artifacts

Formal artifacts

Applying model checking

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



"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"

Natural language (NL) requirement

Real-world artifacts

Formal model (e.g. in NuSMV)

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

$$\mathbf{G}((l > l_0) \rightarrow (t \vee \mathbf{X}\, t))$$

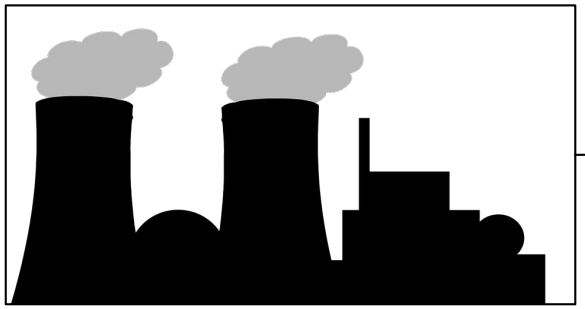
Formal requirement (e.g. in LTL logic)

Formal artifacts

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

Applying model checking

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.ADD
PLUS: ADDER_3(0, FALSE, FALSE, ING
MINUS: ADDER_3(0, FALSE, FALSE, IN
OUT := SW.BSWITCH_OUTP_SIGN;
```

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

"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"

Natural language (NL) requirement

$$\mathbf{G}((l > l_0) \rightarrow (t \vee \mathbf{X}\, t))$$

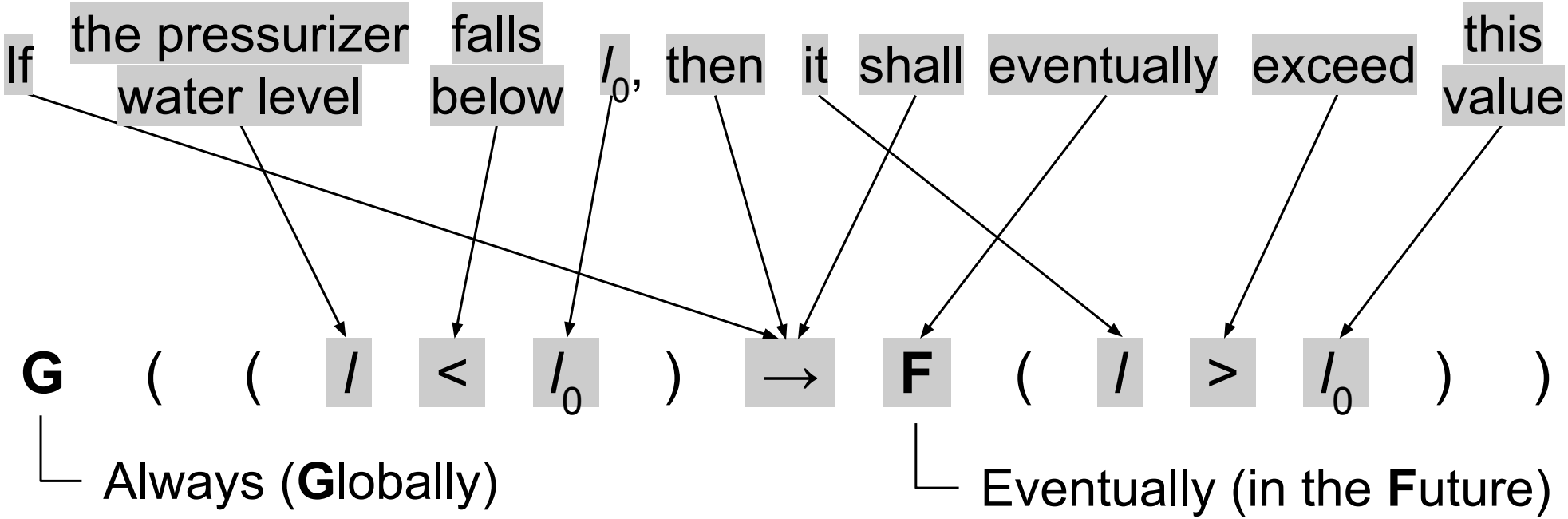
Formal requirement (e.g. in LTL logic)

Our focus

Real-world artifacts

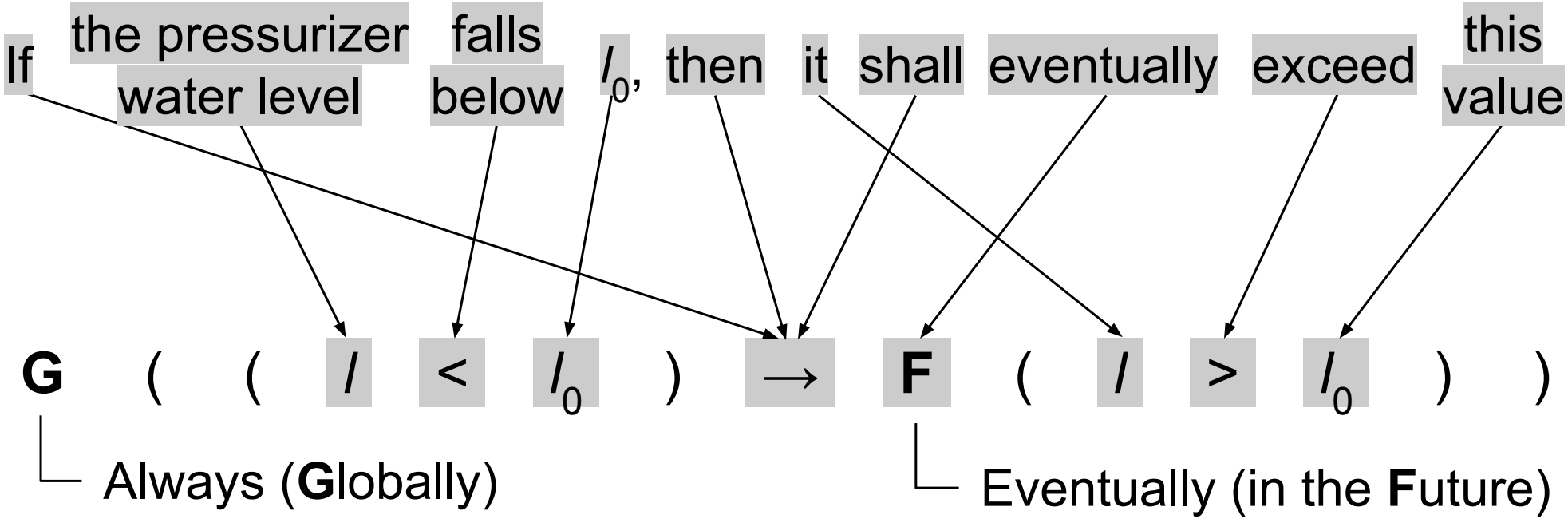
Formal artifacts

Example: linear temporal logic (LTL)



- Extension of the Boolean propositional logic
- Can inquire the state of the model at different time instants
- Temporal operators
  - **G**: always (**G**lobally)
  - **F**: eventually, at some time instant (in the **F**uture)
  - **X**: at the ne**X**t time instant

Example: linear temporal logic (LTL)



- Extension of the Boolean propositional logic
- Can inquire the state of the model at different time instants
- Temporal operators
  - **G**: always (**G**lobally)
  - **F**: eventually, at some time instant (in the **F**uture)
  - **X**: at the ne**X**t 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": $\mathbf{G}(p \rightarrow \mathbf{F}q)$

Example: "**If** the pressurizer water level falls below l_0, **then** it shall **eventually** exceed this value".

(Partially adopted from M.B. Dwyer, G.S. Avrunin, J.C. Corbett (1998). Property specification patterns for finite-state verification. *2nd workshop on Formal methods in software practice*, pp. 7–15. ACM)

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

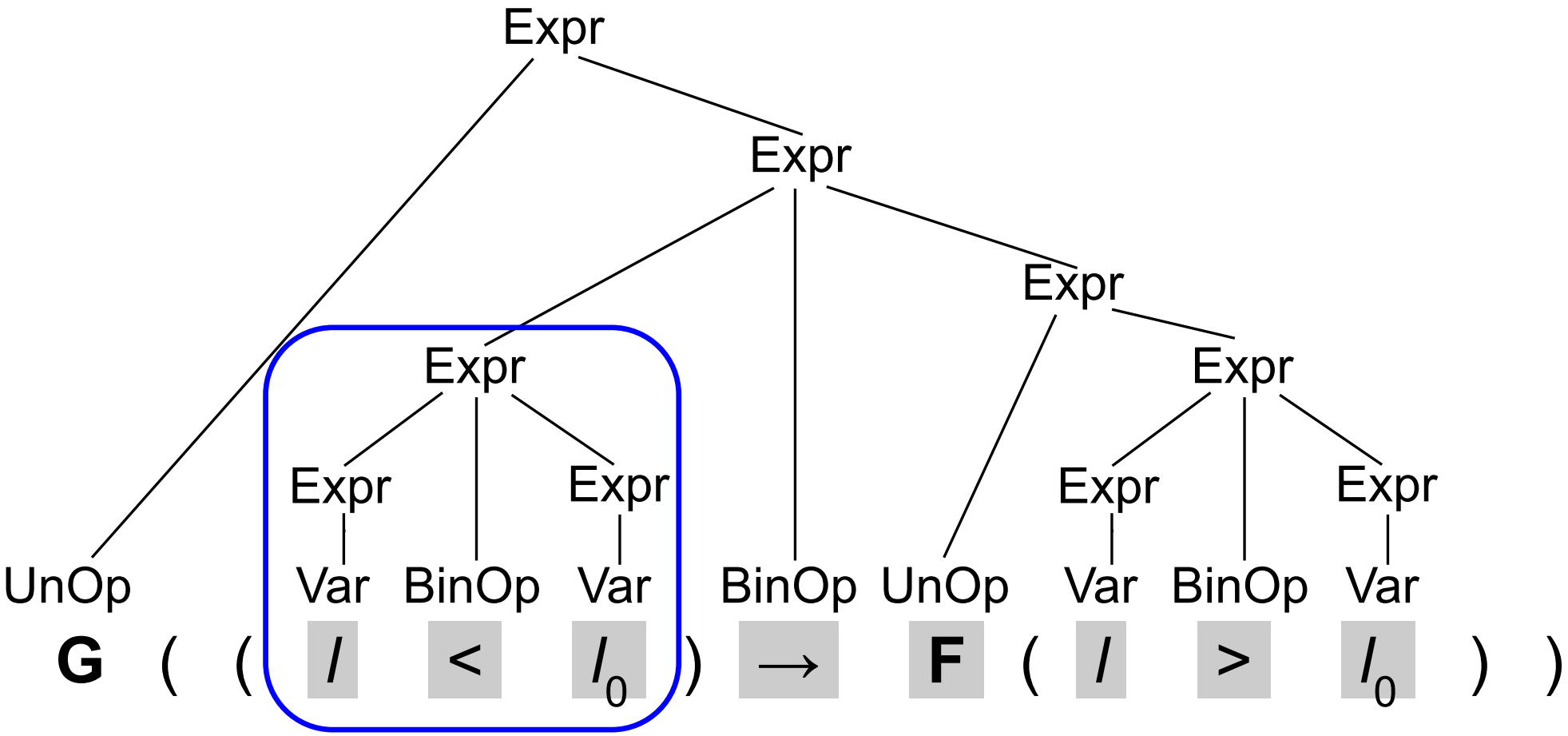
Structured English grammars [1, 2] can **recognize a predefined set of frequent patterns**

- 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

[1] S. Flake, W. Müller, J. Ruf. Structured English for model checking specification. *MBMV*, 2000, pp. 99–108
[2] S. Konrad, B.H. Cheng. Real-time specification patterns. *27th International conference on Software Engineering*. ACM, 2005, pp. 372–381

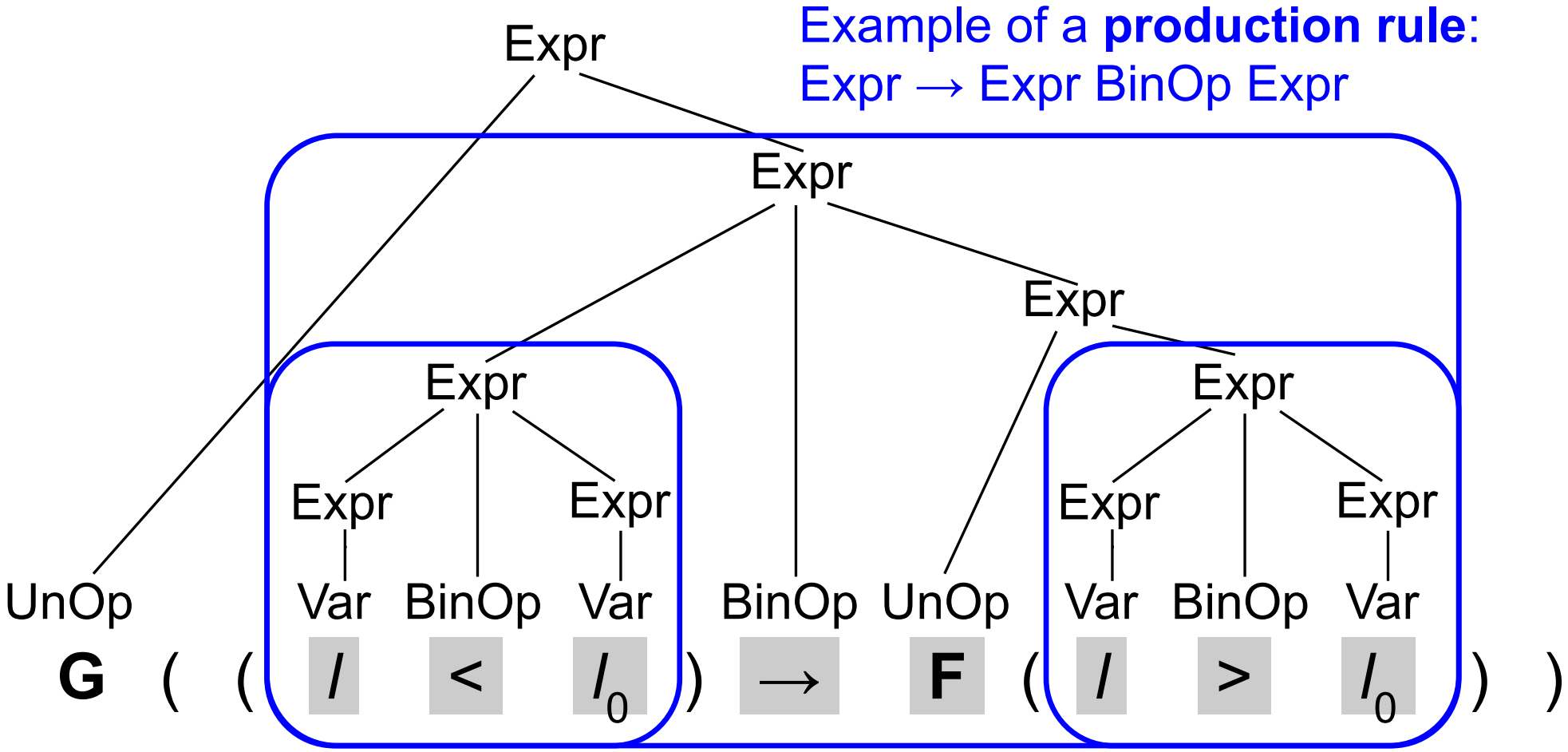
Translation based on formal grammars

Formal grammars and parse trees



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

Formal grammars and parse trees



- **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

A. Fantechi, S. Gnesi, G. Ristori, M. Carenini, M. Vanocchi, P. Moreschini. Assisting requirement formalization by means of natural language translation. *Formal Methods in System Design* 4 (3), pp. 243–263, 1994

State formulas

```
p  --> true                          "any behaviour"
   --> false                         "no behaviour"
   --> ~p                            "not p"
   --> p & p                         "p and p"
   --> p | p                         "p or p"
   --> E q                           "there exists a path in which q"
   --> A q                           "for all paths q"
   --> <ACTION> p                    "there exists a next state reachable
                                      with ACTION, in which p"
   --> [ACTION] p                    "for all next states reachable
                                      with ACTION, p"
```

Path formulas

```
q  --> [p {r} U {r'} p']             "there exists a state in the path, reached
                                      by an action satisfying r', in which p'
                                      is true, and all the previous states in
                                      the path satisfy p and are reached by
                                      an action satisfying r"
   --> G p                           "p is true for all the states in the path"
```

Action formulas

```
r  --> true                          "any action"
   --> false                         "no action"
   --> ACTION
   --> ~p                            "not p"
   --> r | r                         "r or r"
```

Approach: recursive temporal logic representation in NL

A. Fantechi, S. Gnesi, G. Ristori, M. Carenini, M. Vanocchi, P. Moreschini. Assisting requirement formalization by means of natural language translation. *Formal Methods in System Design* 4 (3), pp. 243–263, 1994

One-to-one mapping between temporal logic ACTL and predefined NL text

State formulas

```
p  --> true                          "any behaviour"
   --> false                         "no behaviour"
   --> ~p                            "not p"
   --> p & p                         "p and p"
   --> p | p                         "p or p"
   --> E q                           "there exists a path in which q"
   --> A q                           "for all paths q"
   --> <ACTION> p                    "there exists a next state reachable
                                      with ACTION, in which p"
   --> [ACTION] p                    "for all next states reachable
                                      with ACTION, p"
```

Path formulas

```
q  --> [p {r} U {r'} p']             "there exists a state in the path, reached
                                      by an action satisfying r', in which p'
                                      is true, and all the previous states in
                                      the path satisfy p and are reached by
                                      an action satisfying r"
   --> G p                           "p is true for all the states in the path"
```

Action formulas

```
r  --> true                          "any action"
   --> false                         "no action"
   --> ACTION
   --> ~p                            "not p"
   --> r | r                         "r or r"
```

Approach: a grammar for robot control tasks

H. Kress-Gazit, G.E. Fainekos, G.J. Pappas. Translating structured English to robot controllers. *Advanced Robotics* 22 (12), pp. 1343–1359, 2008.

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

H. Kress-Gazit, G.E. Fainekos, G.J. Pappas. Translating structured English to robot controllers. *Advanced Robotics* 22 (12), pp. 1343–1359, 2008.

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]'

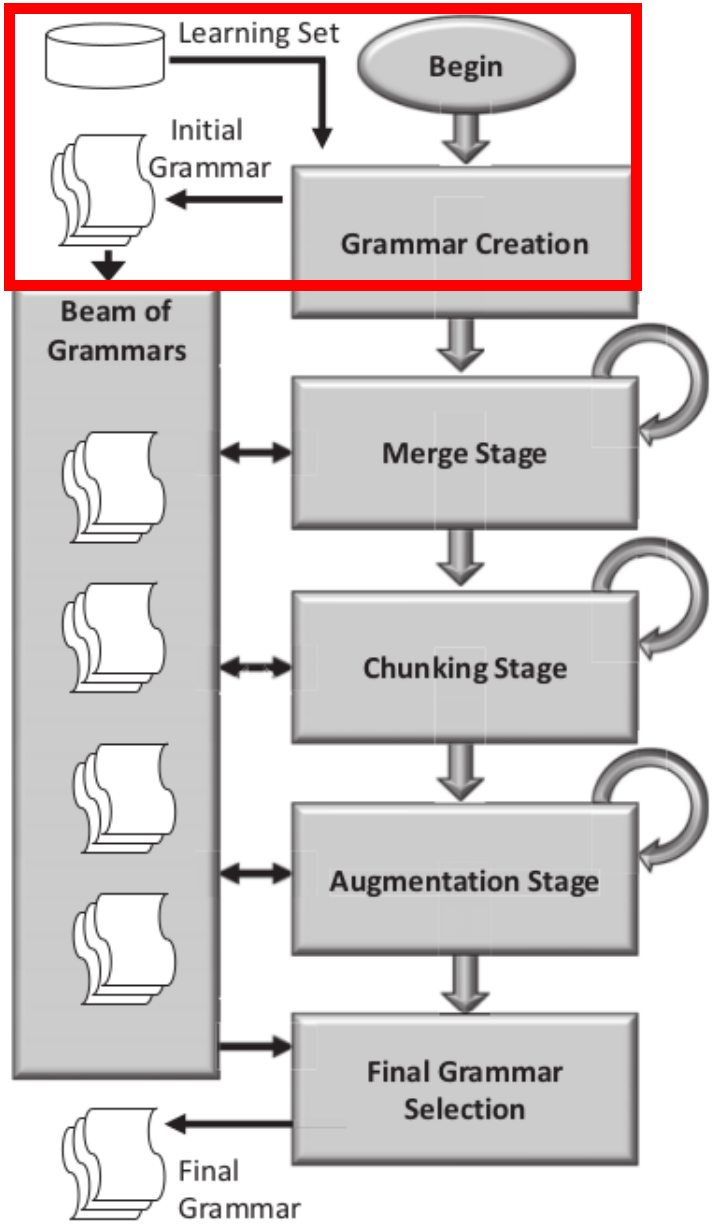
Conditional ::= 'If *Condition* then *Requirement*' | '*Requirement* unless *Condition*' | '*Requirement* if and only if *Condition*'

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]'

- **Concrete domain**
- **Limited recursion**
- **Mapped to LTL**

Approach: grammar learning based on a corpus

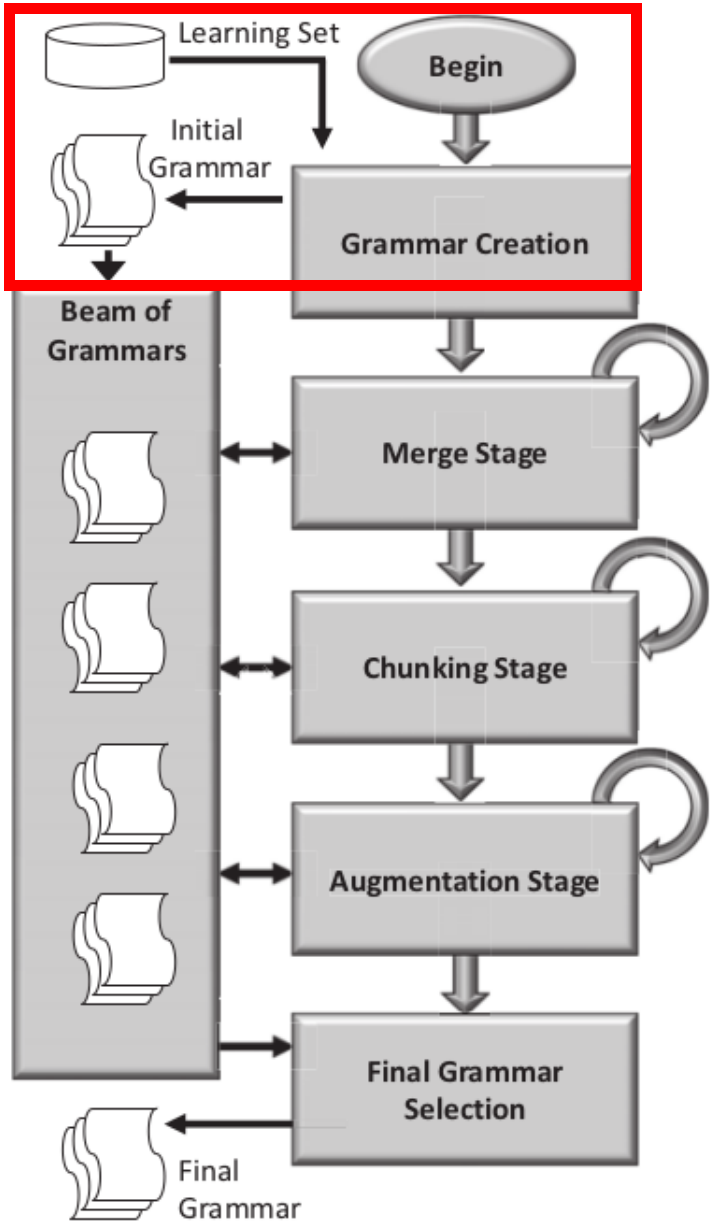


C.B. Harris, I.G. Harris. GLAsT: Learning formal grammars to translate natural language specifications into hardware assertions. *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2016, pp. 966–971

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)

nlp.stanford.edu:8080/parser/index.jsp 140%

Stanford Parser

Please enter a sentence to be parsed:

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

Language: English Sample Sentence Parse

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:

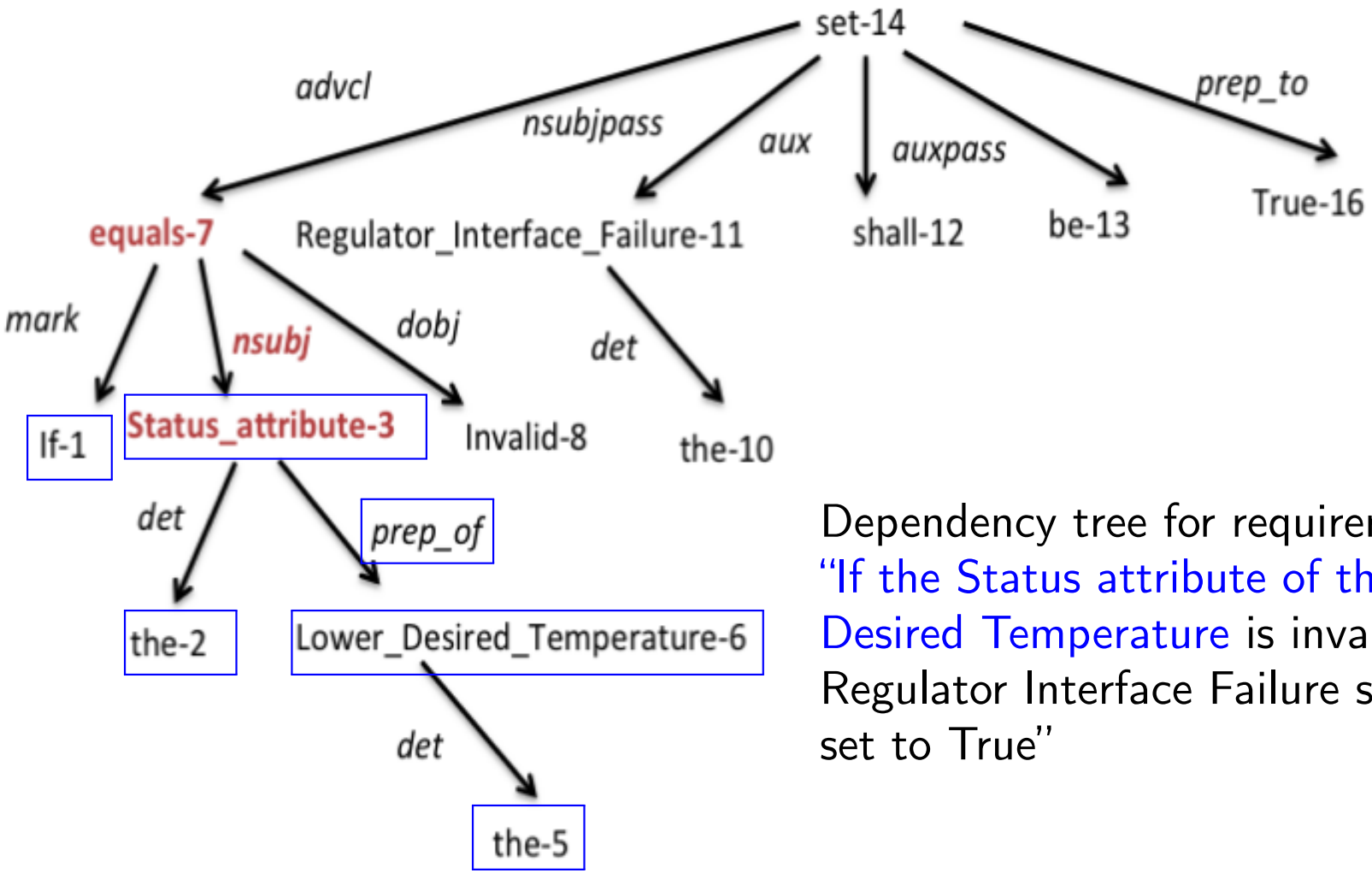
```
(ROOT
  (S
    (SBAR (IN If)
      (S
        (NP (NN water) (NN level))
        (VP (VBZ becomes)
          (ADJP (JJ high)))))
    (, ,)
    (NP (RB then) (DT the) (NN reactor))
    (VP (MD shall)
      (VP (VB be)
        (VP (VBN tripped))))))
```

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)
```

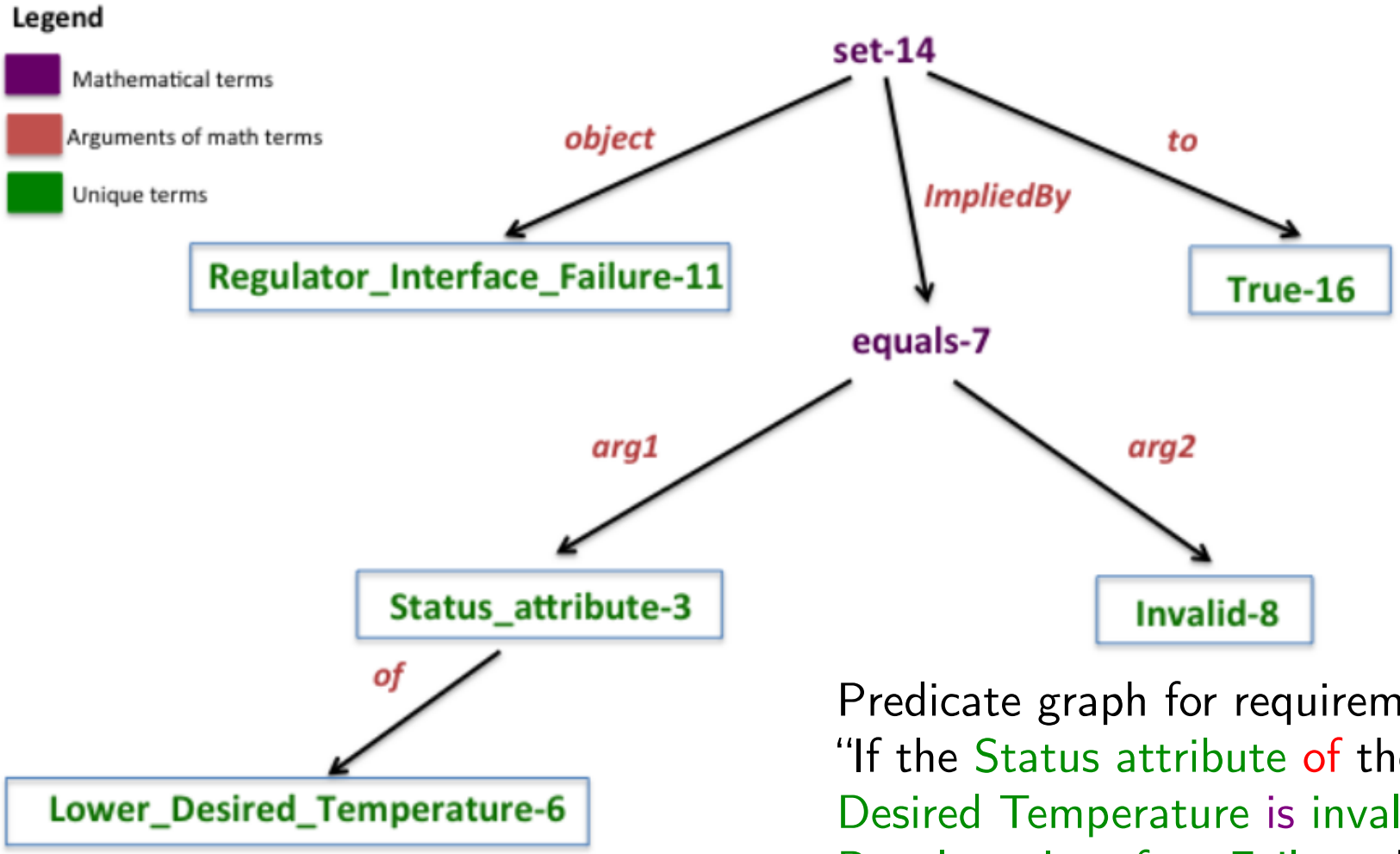
Approach: ARSENAL, stage 1: get word dependencies



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

S. Ghosh, D. Elenius, W. Li, P. Lincoln, N. Shankar, W. Steiner. ARSENAL: automatic requirements specification extraction from natural language. *NASA Formal Methods Symposium*. Springer, 2016, pp. 41–46

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:
  $\mathbf{G}((\text{Lower\_Desired\_Temperature.Status\_attribute} = \text{Invalid}) \rightarrow (\text{RegulatorInterfaceFailure} = \text{TRUE}))$

Statistical machine translation

Statistical machine translation applied to our problem (1)

- Translate language f (NL requirements) to e (temporal logic)
- Bayes rule:

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

- $p(e)$ is the **language model**
- $p(f|e)$ is the **translation model**
- $p(f)$ can be ignored since we search $\text{argmax}_e\ p(e|f)$

Statistical machine translation applied to our problem (1)

- Translate language f (NL requirements) to e (temporal logic)
- Bayes rule:

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

- $p(e)$ 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 → ...	"y"?	p("y" | "x", "→")	=	**0.25**
	"**F**"?	p("**F**" | "x", "→")	=	0.15
	")"?	p(")" | "x", "→")	=	0.00

Statistical machine translation applied to our problem (2)

Basic improvements

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

Statistical machine translation applied to our problem (2)

Basic improvements

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

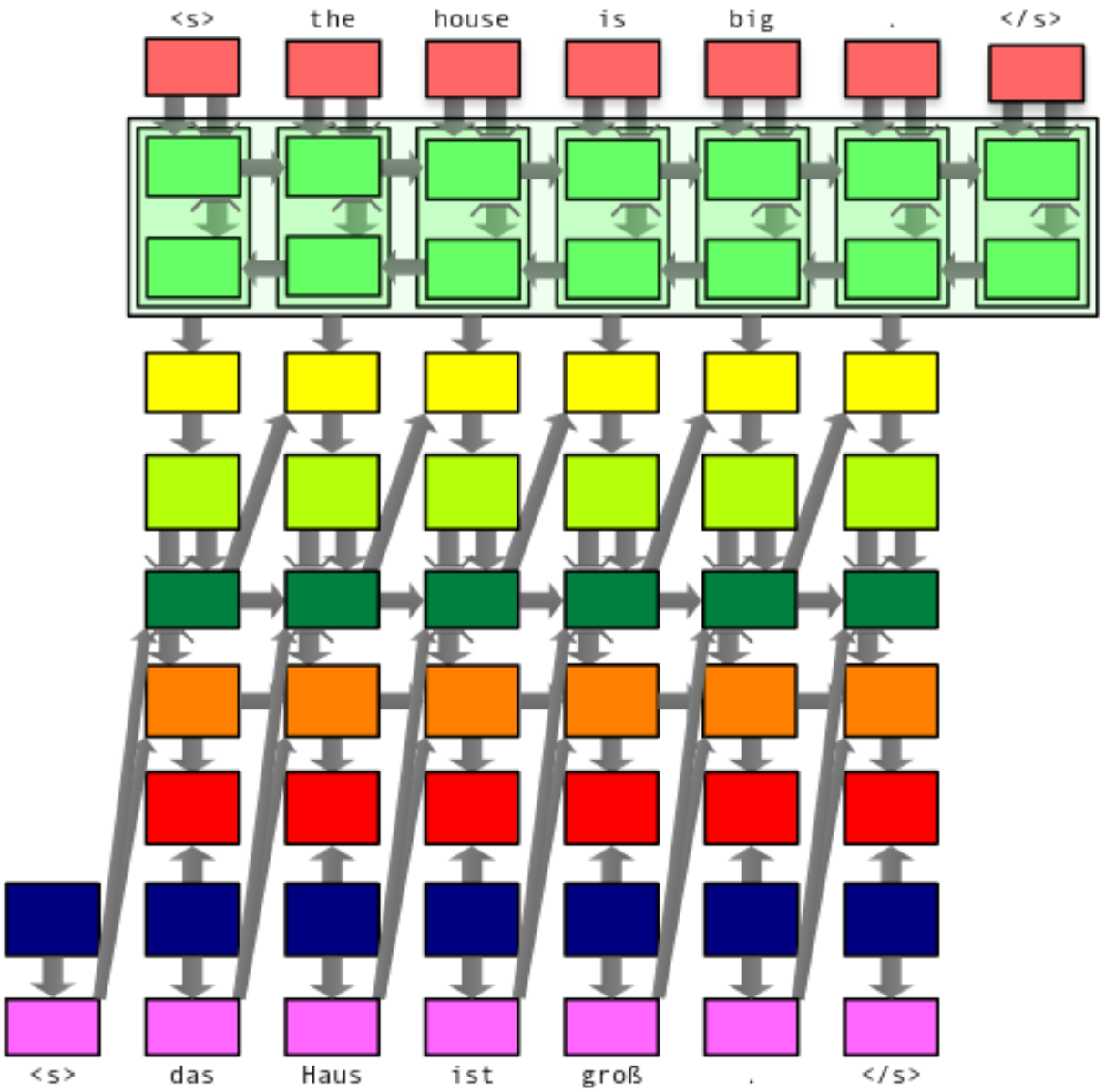
Difficulties in our case

- 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

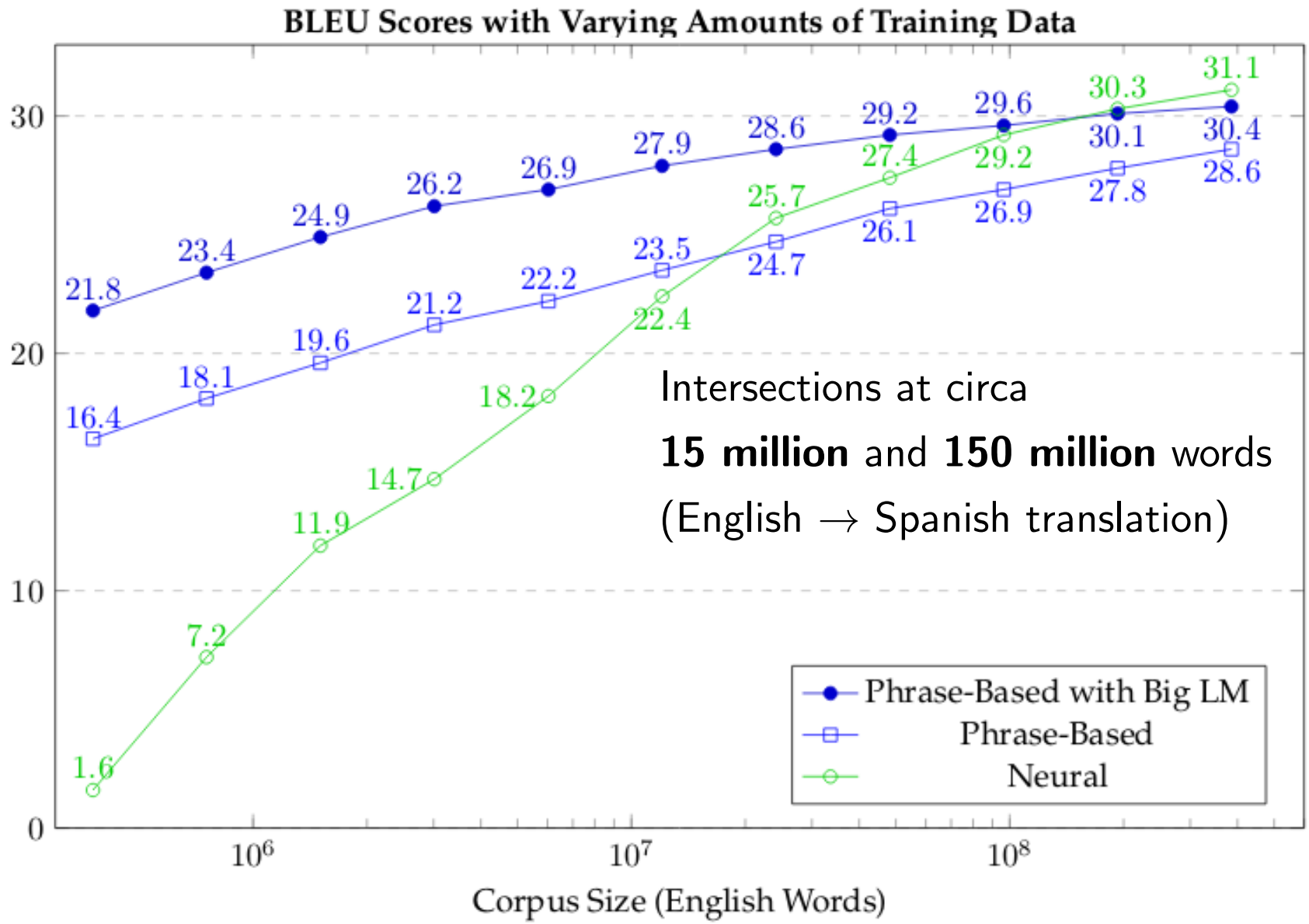
Error

Given output word

Output word embeddings

P. Koehn. Neural machine translation. arXiv:1709.07809, 2017

Comparison of neural and statistical phrase-based translation



Intersections at circa

15 million and **150 million** words

(English → Spanish translation)

P. Koehn. Neural machine translation. arXiv:1709.07809, 2017

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
  - S. Ghosh, D. Elenius, W. Li, P. Lincoln, N. Shankar, W. Steiner. ARSENAL: automatic requirements specification extraction from natural language. *NASA Formal Methods Symposium*. Springer, 2016, pp. 41–46
- 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