
JPL
California Institute of Technology

# Developing Formal Correctness Properties from Natural Language Requirements

Allen P. Nikora
Jet Propulsion Laboratory,
California Institute of Technology
Pasadena, CA
Allen.P.Nikora@jpl.nasa.gov

**The work described in this presentation was carried out at the Jet Propulsion Laboratory, California Institute of Technology. This work is sponsored by the National Aeronautics and Space Administration's Independent Verification and Validation Facility's Research Program. This activity is managed locally at JPL through the Assurance Technology Program Office (ATPO).**


# Agenda

JPL
California Institute of Technology

- Problem/Approach
- Relevance to NASA
- Potential Applications
- Accomplishments and/or Tech Transfer Potential
- Next steps

# Problem/Approach

California Institute of Technology

- Goal - transform natural language specifications into formal notation. Specifically, automate generation of Linear Temporal Logic (LTL) correctness properties from natural language temporal specifications.
- Why?
  - Model-based techniques becoming more widely accepted
  - Analytical verification techniques (e.g., model checking, theorem proving) significantly more effective at detecting types of spec. design errors (e.g., race conditions, deadlock) than manual inspection
  - Many requirements still written in natural language
    - High learning curve for specification languages, associated tools
    - Increased schedule and budget pressure on projects reduce training opportunities for engineers
  - Formulation of correctness properties for system models can be a difficult problem

# Problem/Approach (cont'd)

- Develop more accurate classifiers to discriminate between temporal, non-temporal natural language requirements
  - Extend results of recently completed CI, reported in ISSRE2005[1]
  - Improve probability of detection, reduce false positive rate
- Use pattern matching/natural language processing techniques to map identified natural language temporal requirements to LTL patterns
  - Correctness properties can often be specified as LTL expressions
- Extract semantic information to populate LTL pattern
  - Existing techniques (e.g., "-f" option for SPIN, LTL2BA[2]) can transform LTL expression into a "never" clause for model checkers (e.g., SPIN)

Details

Example

1. A. Nikora, "Classifying Requirements: Towards a More Rigorous Analysis of Natural-Language Specifications", Proceedings of the 16th International Symposium on Software Reliability Engineering, Chicago, IL, Nov 8-11, 2005.
2. P. Gastin, D. Oddoux "Fast LTL to Büchi Automata Translation", CAV'01, LNCS 2102, p. 53-65. Available at http://www.liafa.jussieu.fr/~oddoux/

# Problem/Approach (cont'd)

Schedule of Activities and Deliverables

- Year 1
  - Initial (manual) identification of temporal requirements within requirements documents of collaborating projects.
    - These requirements will be used as training sets for the classifiers and transformation tools developed for this task.
    - On-going throughout first year as additional collaborating projects contribute requirements
  - High-performance specification classifiers (temporal vs. non-temporal requirements)
  - Initial specification and design for tool to transform natural language temporal requirements to LTL expressions
  - End of first year report
- Year 2
  - Final specification and design for tool to transform natural language temporal requirements to LTL expressions ; initial tool implementation.
  - Final toolset for transforming natural language temporal requirements into LTL expressions
  - Final Report

# Relevance to NASA

JPL
California Institute of Technology

- Simplify development of formal correctness properties
- More widespread use of model-based specification, design techniques

$\Rightarrow$

- Earlier identification of defects
- Reduce residual defect content for space mission software systems

# Potential Applications



- Model-based assurance techniques can
  - Find defects earlier in the development process
  - Find types of defects that cannot easily be found by test (e.g., race conditions, deadlocks, lack of progress cycles/starvation)
- Correctness properties
  - Manual specification of correctness properties can be difficult
  - Goal: Automated/assisted transformation of correctness properties written in natural language will simplify application of model-based techniques, encourage greater use
- More widespread use of model-based techniques will result in more reliable flight software.

CLAIM

# Accomplishments and/or Tech Transfer Potential

- New task, work just starting
- Gathering requirements from collaborating projects – projected data availability high
  - Requirements for one project already in hand
  - Working with additional on-going projects to collect requirements for analysis
- Acquiring relevant classification, natural language processing tools. Several tools already in hand:
  - Link Grammar natural language parser
  - TnT parts-of-speech tagger
  - Weka
  - SPIN
- Projected Technology Transfer Level:
  - Year 1 (by June 07): 3 (“Experimental demonstration of critical function &/or proof of concept”)
  - Year 2 (by June 08): 4 (“Validation in a lab environment”) on collaborating flight projects

# Next steps



- Focus areas for next year
  - Develop High Performance Classifiers
    - Improve ability to discriminate between temporal, non-temporal requirements
      - Try more classifiers – only a subset of classifiers in WEKA has been applied in previous work
      - Add more structural information to requirements being classified (e.g., parse tree information – part of speech, level in tree)
      - “Bagging” – apply more than one classifier, develop meta-classifier in which individual components are weighted according to how well they perform (e.g., pd, pf)
      - Direct pattern matching – use natural language parsing/transformation techniques to match structure of requirement to LTL pattern
  - Map Temporal Requirements Structure to LTL Patterns
  - Populate LTL patterns with semantic information
    - Work by Jane Malin, JSC, et al. (“Reconciler”) may be applicable

# Discriminating Between Requirement Types (cont'd)



Machine Learning/Natural Language Processing (cont'd)

- Developing input to classifiers
  - Apply TnT POS tagger to requirement text
  - Form list of tags – n'th list element corresponds to n'th word in requirements text
  - Map each word in requirement to unique numerical ID
  - Map each POS tag to unique numerical ID
  - Concatenate requirements text word list, POS tag list
    - Use only first 200 elements of word list, POS tag list
  - Apply supervised discretization[1]
    - Many classifiers require discrete data input
- Created training data sets (252 temporal requirements, 252 nontemporal requirements) that included the attributes accounting for the first 60%, 80%, and 90% of the classification merit
- Other input representations yielded poorer classifiers

1. U.M. Fayyad, K.B. Irani,. "Multi-Interval Discretization Of Continuous-Valued Attributes For Classification Learning", Proc. 13th Int. Joint Conf. AI (IJCAI-93), Chamberry, France, Aug./ Sep. 1993.

Return to Approach

# Discriminating Between Requirement Types (cont'd)



Machine Learning/Natural Language Processing (cont'd)

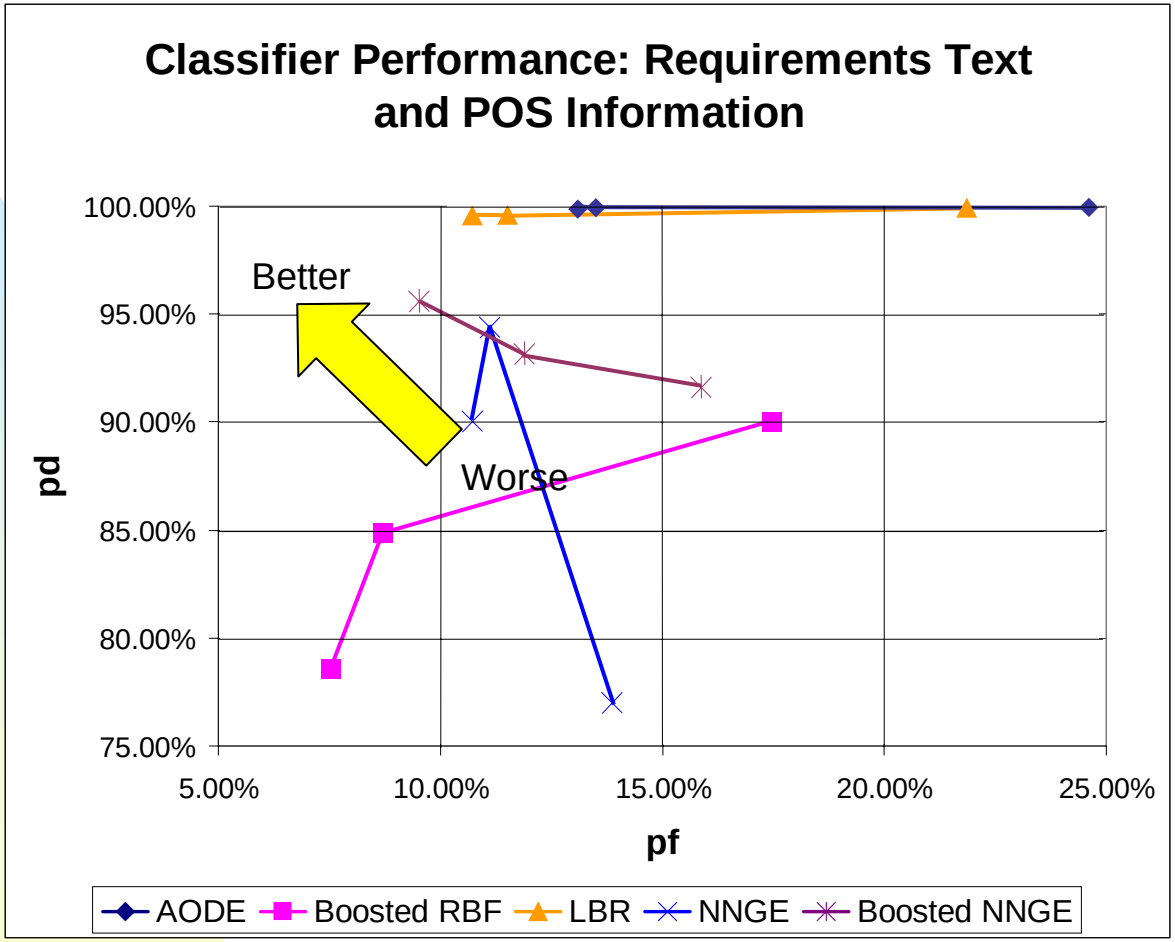
- Five well-performing classifiers
  - AODE[1]
  - RBF Network[2] boosted with AdaBoostM1[3]
  - Lazy Bayesian Rules (LBR)[4]
  - NNGE[4, 5]
  - NNGE boosted with AdaBoostM1
- Classifiers evaluated according to four criteria:
  - ***pd*** (probability of detection) - a/(a+b)
  - ***pf*** (probability of false detection, or "false positives") - c/(c+d)
  - ***accuracy*** - c/(a+c)
  - ***precision*** - (a+d)/(a+b+c+d)

| Detected as temporal | Detected as nontemporal | |
|---|---|---|
| a | b | Really temporal |
| c | d | Really nontemporal |

Return to Approach


1. G. I. Webb, J. Boughton, Z. Wang, "Not So Naive Bayes: Aggregating One-Dependence Estimators", Machine Learning, 58(1), pp. 5-24, 2005
2. D.S. Broomhead, D. Lowe, "Multivariate Functional Interpolation And Adaptive Networks", Complex Systems, 2:321-355, 1988.
3. Y. Freund, R. E. Schapire. "Experiments With A New Boosting Algorithm". Proc International Conference on Machine Learning, Morgan Kaufmann, San Francisco, 1996.
4. B. Martin, "Instance-Based Learning: Nearest Neighbor With Generalization", Master Thesis, University of Waikato, Hamilton, New Zealand, 1995
5. Z. Zheng, G. Webb, "Lazy Learning Of Bayesian Rules", Machine Learning, Vol 41, No 1, pp. 53-84, Kluwer Academic Publishers, 2000

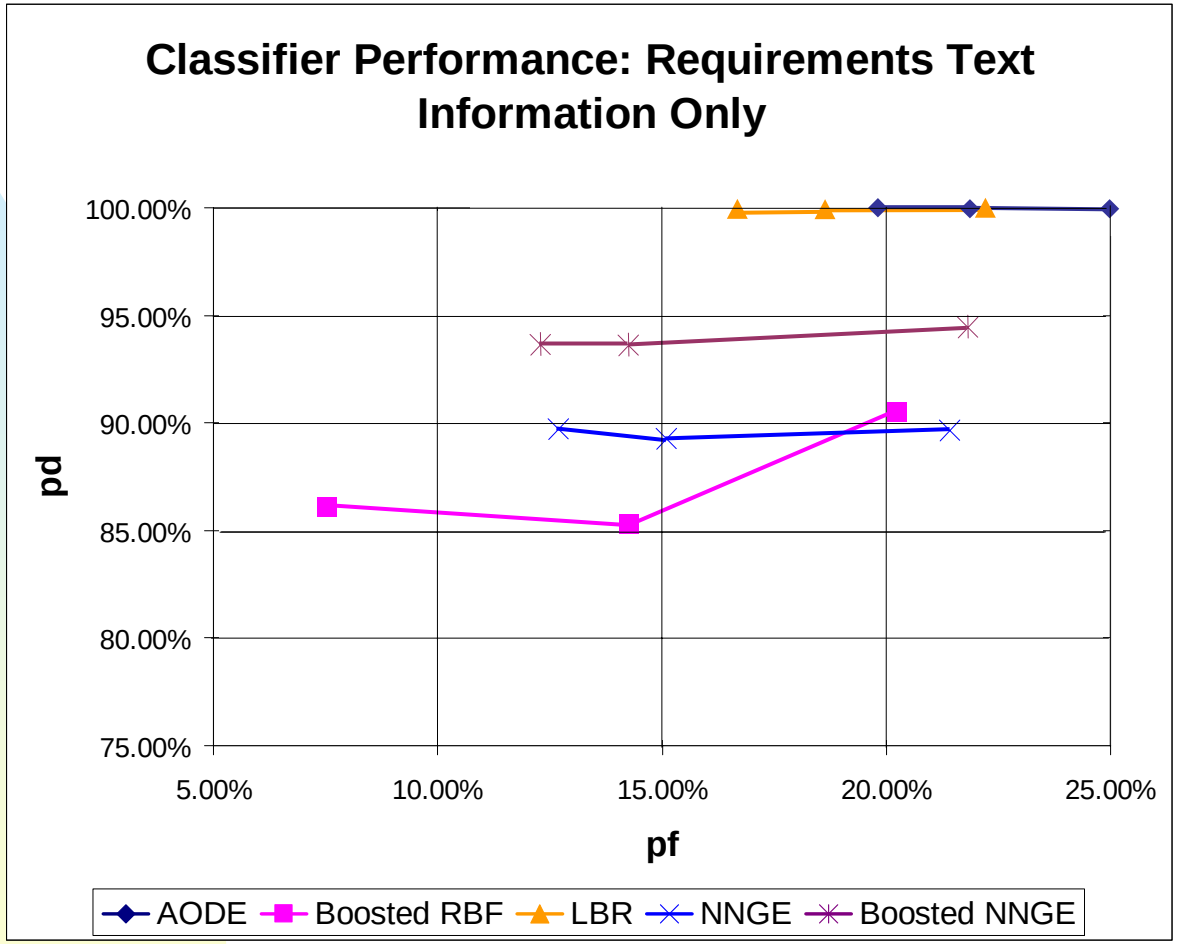
# Discriminating Between Requirement Types (cont'd)




Classifier Performance – Requirements Text and POS Information

Return to Approach  Next Slide

# Discriminating Between Requirement Types (cont'd)




Classifier Performance – Requirements Text Only

Return to Approach | Previous Slide

# Example

- Natural language requirement text
  - “Electrical interfaces passing through cable cutter separation devices shall be deadfaced prior to actuation of the device”
- Requirements text parsed with Link Grammar parser[1, 2]

```
(S (NP (NP Electrical interfaces)
       (VP passing
           (PP through
               (NP cable cutter separation devices))))
   (VP shall
       (VP be
           (VP deadfaced
               (PP prior to
                   (NP (NP actuation)
                       (PP of
                           (NP the device))))))))
```

Return to Approach  Continue Example


1. D. Sleator, D. Temperley, “Parsing English with a Link Grammar”, Third International Workshop on Parsing Technologies, 1993
2. Link Grammar, http://bobo.link.cs.cmu.edu/link/

# Example (cont'd)

- Matching LTL Pattern (patterns developed at KSU CIS Dep't[1])

S precedes P:

(*) Globally `!P W S`

(*) Before **R** `<>R -> (!P U (S | R))`

(*) After **Q** `[]!Q | <>(Q & (!P W S))`

(*) Between **Q** and **R** `[]((Q & !R & <>R) -> (!P U (S | R)))`

(*) After **Q** until **R** `[](Q & !R -> (!P W (S | R)))`

Where "W" represents the "weak until operator" and "U" represents the "strong until operator", which are related as shown below:

```
p W q = ([]p) | (p U q)
      = <>(!p) -> (p U q)
      = p U (q | []p)
```

Return to Approach | Continue Example

1. Kansas State University CIS Department, Laboratory for Specification, Analysis, and Transformation of Software (SanToS Laboratory), Specification Patterns Project, http://patterns.projects.cis.ksu.edu/

# Example (cont'd)

- Extract semantic information
  - Event "P" represented by noun phrase "(NP (NP actuation) (PP of (NP the device)))"
  - Event "S" represented by clause "(S (NP (NP Electrical interfaces) (VP passing (PP through (NP cable cutter separation devices)))) (VP shall (VP be (VP deadfaced))))".
  - Corresponding LTL specification
    - !cable_cut W deactive_electrical_interface
    - Equivalent to: <>(cut_cable) -> (!cut_cable U deactivate_electrical_interface)

Return to Approach

# Linear Temporal Logic

California Institute of Technology

- Linear Temporal Logic (LTL) is a way of reasoning about a system's desired properties
  - p is invariantly true
  - eventually p becomes invariantly true
  - p always implies not q
  - p always implies eventually not q
- Interesting in model checking context because an LTL expression corresponds to an automaton that can become part of the model being checked
- Introduced by Amir Pneuli in late 1970s
- Based on "tense logics" developed in 1950s

Return to Approach

# Linear Temporal Logic (cont'd)

California Institute of Technology

- LTL can specify both safety and liveness properties
- LTL is propositional logic plus following temporal operators:
  - []p: always p
  - <>q: eventually q
  - p U q: p until q

Return to Approach

# Linear Temporal Logic (cont'd)



## Common LTL Expressions

| | | |
|---|---|---|
| [] p | always p | invariance |
| <> p | eventually p | guarantee |
| p -> <> q | p implies eventually q | response |
| p->q U r | p implies q until r | precedence |
| [] <> p | always eventually p | recurrence (progress) |
| <> [] p | eventually always p | stability (non-progress) |
| <> p -> <> q | eventually p implies eventually q | correlation |

Return to Approach

# Linear Temporal Logic (cont'd)

California Institute of Technology

- Common LTL Rules
  - ![]p ⇔ <>!p
  - !<>p ⇔ []!p
  - !(p W q) ⇔ (!q) U (!p ∧ !q)
  - !(p U q) ⇔ (!q) W (!p ∧ !q)
  - [] (p ∧ q) ⇔ []p ∧ []q
  - <> (p ∨ q) ⇔ <>p ∨ <>q
  - p U (q ∨ r) ⇔ (p U q) ∨ (p U r)
  - (p ∧ q) U r ⇔ (p U r) ∧ (q U r)
  - p W (q ∨ r) ⇔ (p W q) ∨ (p W r)
  - (p ∧ q) W r ⇔ (p W r) ∧ (q W r)
  - [] <> (p ∨ q) ⇔ []<>p ∨ []<>q
  - <> [] (p ∧ q) ⇔ <>[]p ∧ <>[]q

Return to Approach



# Linear Temporal Logic (cont'd)

- Relationship between never claims and LTL
  - Desired property can be expressed as an LTL formula.
    - Requirement: "The electrical interfaces between the probe and the orbiter shall be deadfaced prior to activation of the cable cutting device"
    - Corresponding LTL formula: "Not p until s", written as "!p U s"
      - "p": activation of cable cutting device
      - "s" deadfacing of electrical interfaces
  - LTL formula is then negated (the negated property should **NEVER** occur)
    - Example: "!(!p U s)"

Return to Approach

# Linear Temporal Logic (cont'd)

- Relationship between never claims and LTL (cont'd)
  - Negated formula can then automatically be converted to a never claim using one of the spin execution options
  - Example – produce never claim to see if property "not p until s" can be violated by a model
    - spin –f '!(!p U s)' [> text file]

```
never {    /* !(!p U s) */
accept_init:
T0_init:
   if
   :: (! ((s))) -> goto T0_init
   :: (! ((s)) && (p)) -> goto accept_all
   fi;
accept_all:
   skip
}
```

Return to Approach

# Propositional Logic Operators

California Institute of Technology

- Unary
  - !: negation
- Binary
  - &&: logical and
  - ||: logical or
  - ->: logical implication
  - <->: logical equivalence

Return to Linear Temporal Logic

# Strong vs. Weak Until

California Institute of Technology

- Weak until (p W q)
  - A state $s_i$ satisfies p W q iff
    - $s_i$ satisfies q, or $s_i$ satisfies p and $s_{i+1}$ satisfies (p W q)
    - Formally: $s_i \models p \, W \, q$ iff $s_i \models q \vee (s_i \models p \wedge s_{i+1} \models (p \, W \, q))$
    - Never actually requires q to become true
- Strong until (p U q)
  - A state $s_i$ satisfies p U q iff
    - For some value of j, j >= 1, $s_j$ satisfies q, and for all values of k, i <= k < j, $s_k$ satisfies p
    - Formally: $s_i \models p \, U \, q$ iff $\exists \, j,(j >= i)$: $s_j \models f$ and $\forall \, k,(i <= k < j)$: $s_k \models p$
    - Requires that q eventually become true

Return to Linear Temporal Logic