Karlsruher Institut für Technologie

Text Understanding for Programming in Natural Language

Control Structures

Mathias Landhäußer, Ronny Hug

IPD Tichy, Fakultät für Informatik

Problem Solved?
A Quote from a Textbook

End-user development is

"a set of methods [. . .] that allow

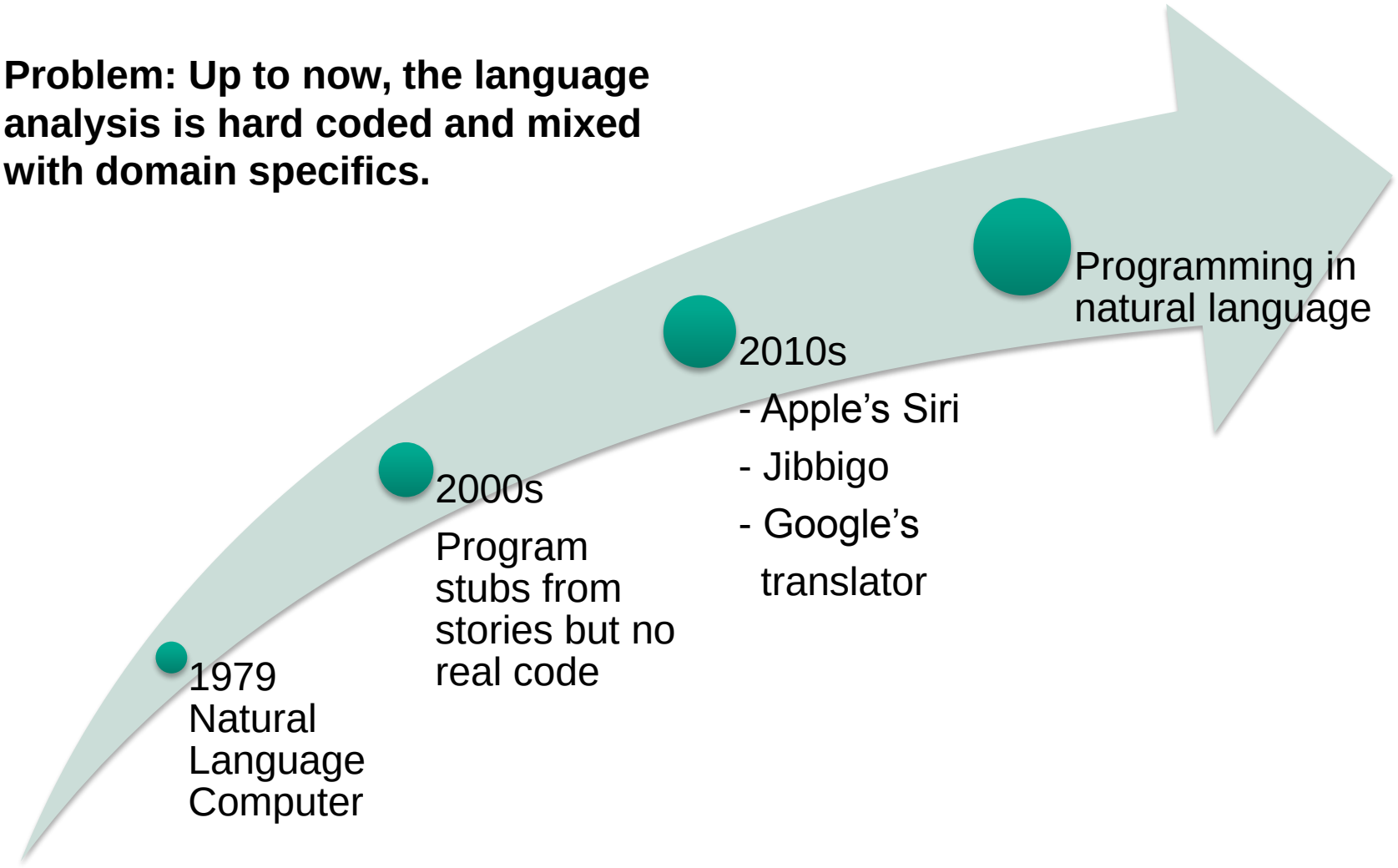
users of software systems [. . .]

to **create**, **modify**, or **extend** a **software artifact**."

Lieberman et al., 2006

A Little Bit of History

Problem: Up to now, the language analysis is hard coded and mixed with domain specifics.

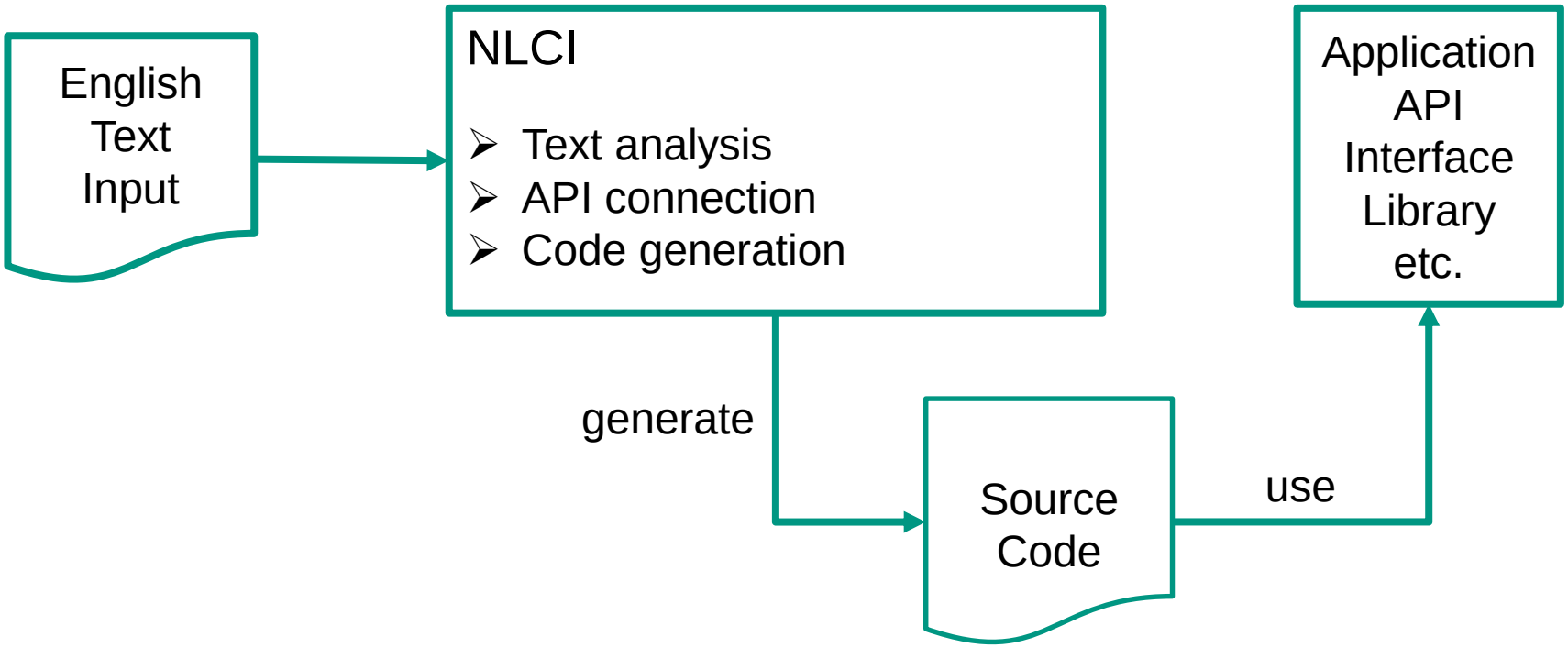


Different Points of View

- Prior work thinks about *programming* in natural language
  - How ~~would~~ **should** users express a loop, a condition, etc.?
  - How can we **provide** “simple” commands?
  - Can we use domain knowledge to **boost** performance?

- Our project thinks about programming *in natural language*
  - How **do users express** repetition, alternatives, parallelism, etc.?
  - How can we **identify** these phenomena in text?
  - How can we **map** that to programing structures?

Natural Language Command Interpreter (NLCI) General Architecture



Application domain: Alice
Input: Animation script
Output: Code to produce that animation

Approach

- Same approach for all control structures:
  Use signal phrases to trigger action search

- Configuration
  - What's an action? → verbs
  - How to find them? → depends on control structure
  - What to ignore? → action blacklist

- Not covered
  - References to distant parts of the text

Approach

- Pre-processing
  - Parse the text (Stanford)
  - Result: part-of-speech tags & typed dependency graphs
- Processing
  - Traverse graph to identify actions in control structures
  - Check nesting of control structures
  - Remove unnecessary structures
- Record results as annotations in the text

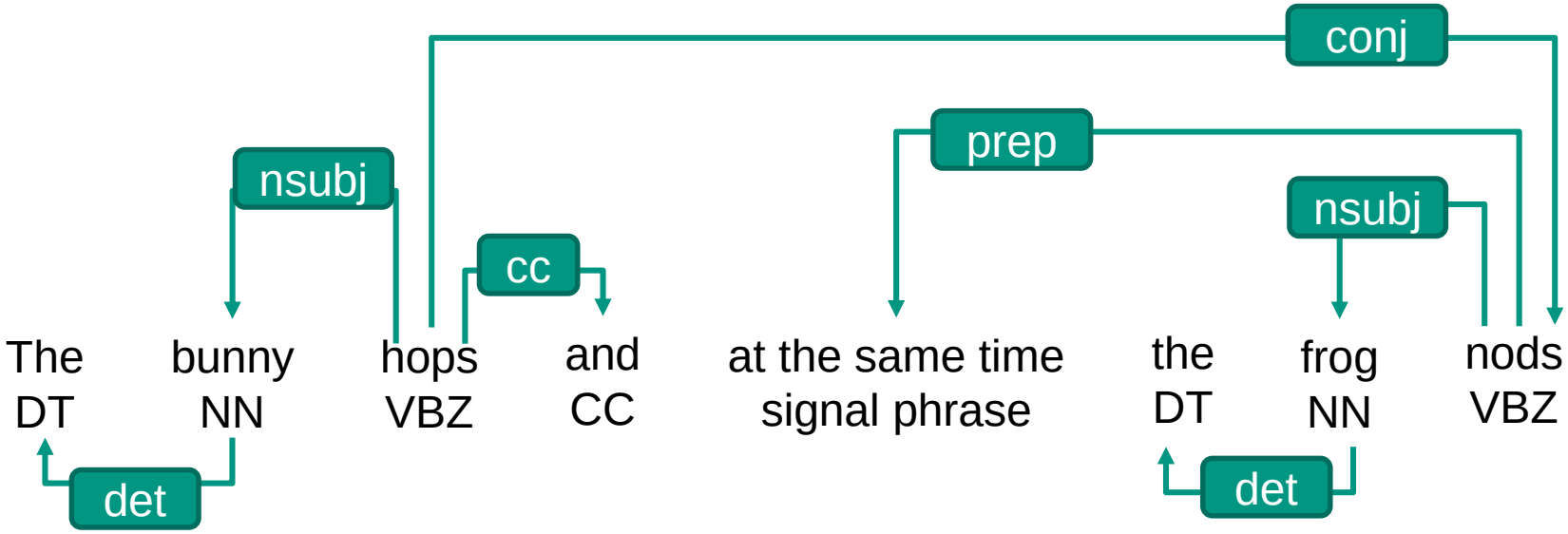
Control Structures

- Sequential blocks
  - *do in order* (first do x then do y)
- Repetition
  - *loop* (do x n times)
  - while (do x until y is finished)
  - for all in order (A, B, and C do x, y, and z)
- Parallelism
  - *do together* (do x and y simultaneously)
  - *for all together*
- Alternatives (not yet implemented)
  - *If … else* (sub phrases that start with a condition)

Dependency Graph Traversal

Trace profile for *do together*
Deps: conj, prep
POS tags: *VB**
Ignored words: times
Signal phrases: at the same time

- Same idea for all control structures
  - Starting point: signal word/phrase
  - Goal: Identify actions that are connected to the key phrase
  - Which edges to visit: list of edge types
  - Stopping criterion: all reachable relevant edges visited?



Evaluation
What and how?

- Text corpus (with expected solutions)
  - Written by different people, with and without programming knowledge
  - Different animations (i.e. programs)
  - Control structures are known
- Evaluation covers different input sets
  - Erroneous input (with parser errors) → “real world”
  - Corrected input (without parser errors) → “ideal world”
- Evaluation covers different configurations
  - Default configuration (linguistic considerations)
  - Tuned configuration (counteract parser errors)

Evaluation Results

Evaluation Config	Expected Annotations	Correct Ann.	Incorrect Ann.	Missing Ann.	Incorrect Nesting
Erroneous Default	795	649 82 %	77 10 %	140 18 %	6 1 %
Erroneous Tuned	795	705 89 %	78 10 %	79 10 %	11 1 %
Corrected Default	529	513 97 %	5 1 %	16 3 %	0 0 %

Erroneous Test Corpus: 52 Texts
Corrected Test Corpus: 28 Texts

Conclusion

- Control structures are essential for programming
  - Respective phenomena must be identified in text
- We use Stanford's typed dependencies to identify them
  - Approach is sensitive to parser errors
  - We can remedy some of them