On Teaching Programming with Nondeterminism

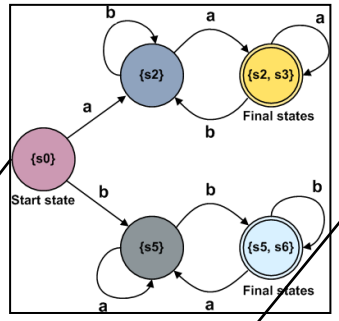
Giora Alexandron, Michal Armoni,
Michal Gordon, David Harel

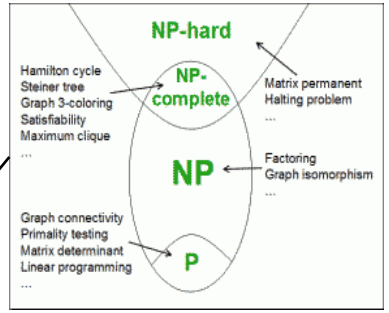
Nondeterminism (ND)

- A fundamental idea of CS
  - First introduced into CS by Rabin and Scott *
- Appears in various contexts:
  - Automata theory *
  - Nondeterministic programming (Dijkstra's guarded commands, Logic Programming, LSC, etc.)
  - Concurrent and asynchronous systems
  - …
- In the curriculum:
  - CC2001: Elective unit on automata theory
  - CC2013: "Given the vastly increased importance of parallel and distributed computing…identify essential concepts… **promote those topics to the core**."

* D. Scott and M. Rabin (1959). "Finite Automata and Their Decision Problems".

Teaching ND

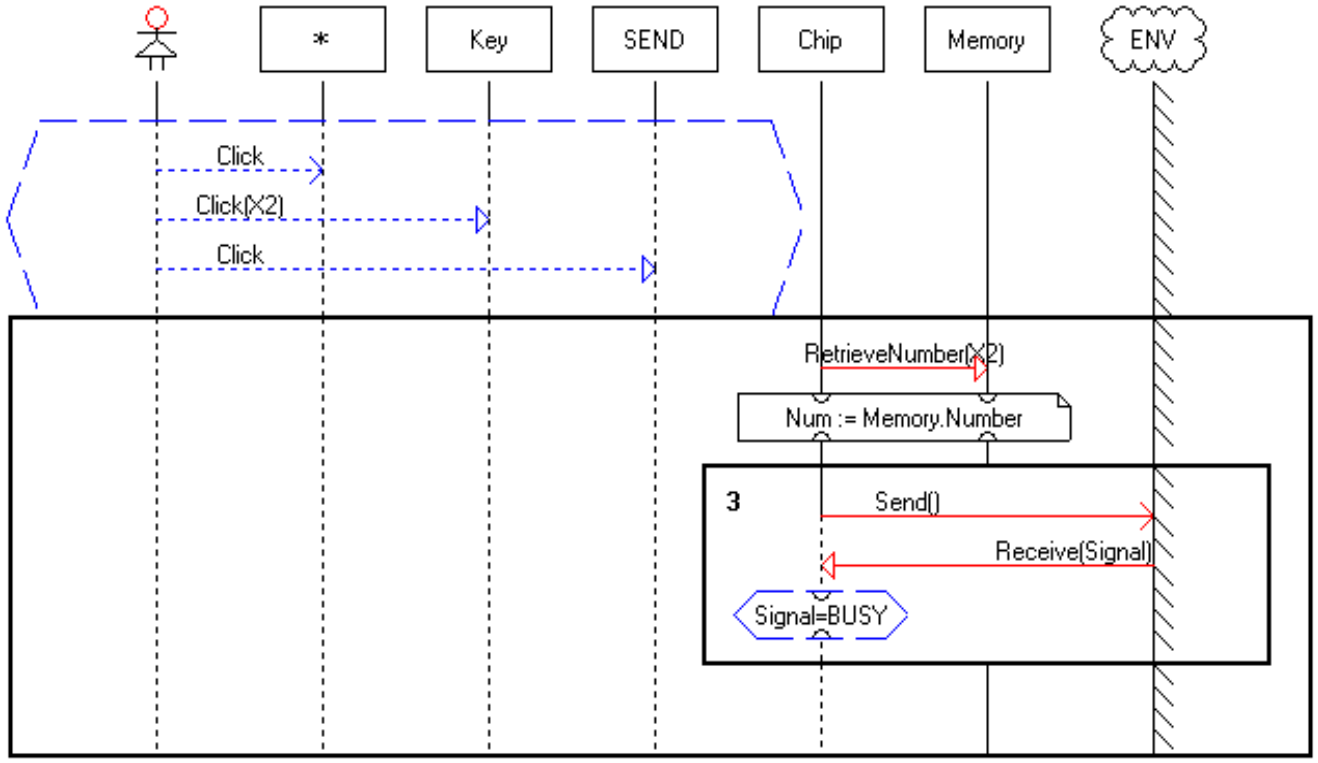




- ND is a complex concept
  - Cognitively, but also psychologically:
    - Dijkstra: "I myself had to overcome a considerable mental resistance before I found myself willing to consider non-deterministic programs seriously"
  - Known to be hard to teach and learn
- What kind of ND is usually taught?
  - Automata theory, Class NP…
    - Hige level abstraction, existential semantics, mathematical context…
- What we suggest:
  - Teach the kind of ND that appears in non-deterministic programming (*operative* ND)
  - Teach it in the context of a programming course
  - Using a nondeterministic language such as LSC

LIVE SEQUENCE CHARTS (LSC*)

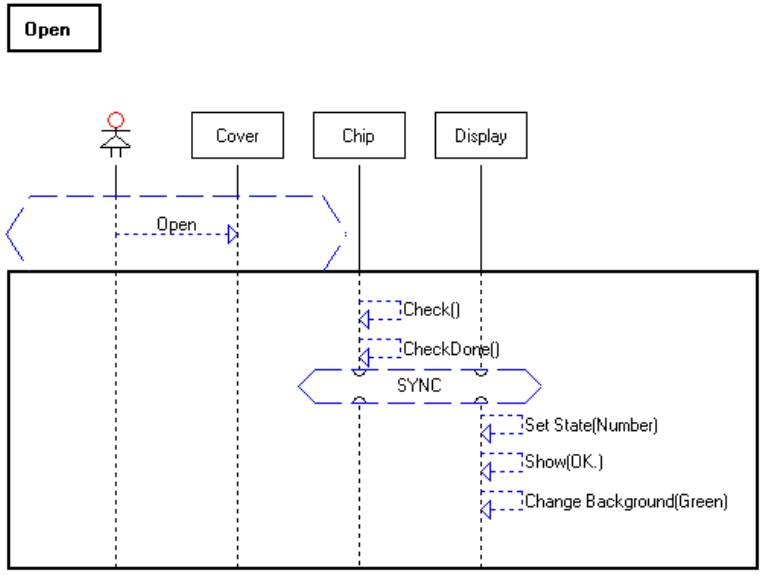
- Visual
- Scenario-based

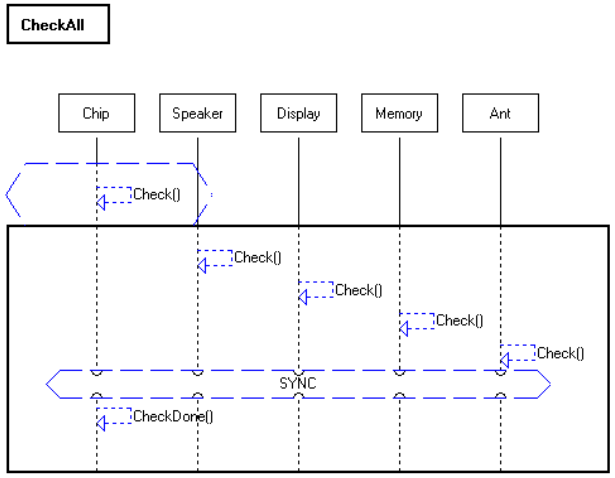


* W. Damm and D. Harel (2001). “LSCs: Breathing Life into Message Sequence Charts”.

ND IN LSC – SOME EXAMPLES

- Nondeterministic order:
  - within a chart
  - between charts

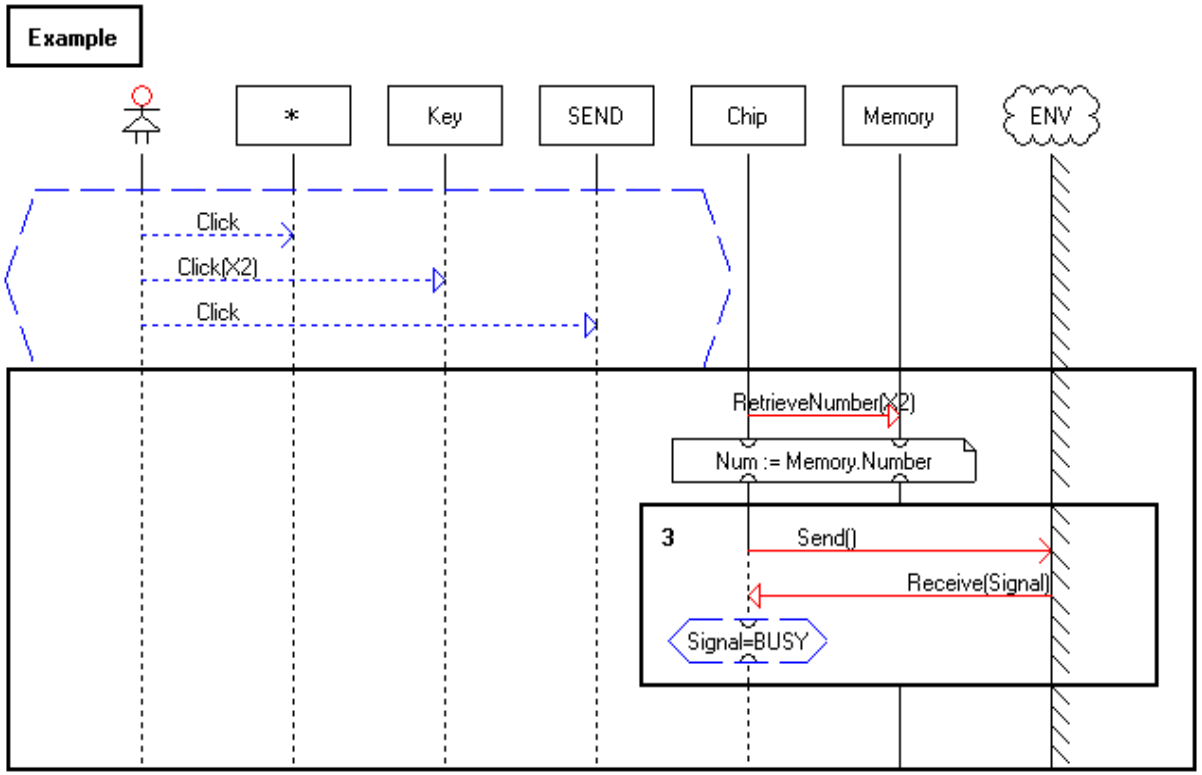




1. One chart activates the other
2. The two proceed simultaneously

ND IN LSC – SOME EXAMPLES

- Must vs. May modality



Research setting

- 45 hours semestrial course on LSC
- 12th grade high-school students majoring in computer science
- Course structure: Theory + lab + final project
- Assessment: Exams + final projects

Assessing students' understanding

- A combined Bloom/SOLO taxonomy*:

	Unistructural	**Multistructural**	**Relational**
Applying	Quantitative + Qualitative	Quantitative + Qualitative	Qualitative
Creating	Quantitative + Qualitative	Qualitative	Qualitative

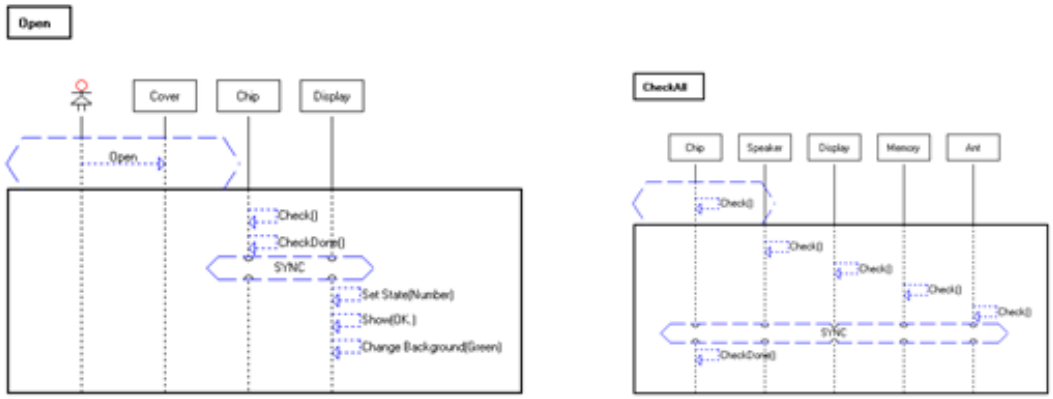
* Based on: Meerbaum-Salant et al. (2010). "Learning computer science concepts with scratch".

OPERATIONALIZATION

	Unistructural	Multistructural	Relational
Applying	Quantitative + Qualitative	Quantitative + Qualitative	Qualitative
Creating	Quantitative + Qualitative	Qualitative	Qualitative

- The category of Applying-Multistructural:
  - Applying: The ability to mentally simulate pieces of code that contain a non-deterministic element.
  - Multistructural: a perspective that incorporate multiple LSC charts.
  - Example of a question that falls into this category:

"1. Write a possible execution order for the following charts.
2. Is it the only possible order? If not, write another possible order."



Findings (example of)

- Quantitative:

	Unistructural	**Multistructural**	**Relational**
Applying	83%, N=26	76%, N = 18	
Creating	100%, N=10		

- Qualitative (not shown here)

- Summary of findings:
  - Comprehend systems that contain ND
  - Create systems that contain ND

Conclusions

- High-school students can reach a significant understanding of *operative* ND, when the concept is introduced in:
  - The context of a hands-on programming course
  - Using a nondeterministic language like LSC
- Implementation:
  - Can be done by integrating a section on nondeterministic programming into an advanced high-school course
  - Using LSC achieves additional educational goals, such as introducing a new programming paradigm and developing abstract thinking
- Open issues:
  - The effect of learning *operative* ND on the learning of the kind of ND that appears in automata theory