Languages for Programming: From Punched Cards to Wise Computing

David Harel
The Weizmann Institute
Languages for programming have to be endowed with formal syntax and semantics, which must unambiguously give rise to their intended functionality: full executability.
First, a very brief history of general programming methods
Once upon a time, we used punched tape and punched cards...
<table>
<thead>
<tr>
<th>Machine language (1945)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assembly language (1950)</td>
</tr>
</tbody>
</table>

```
$MOD8253
DSEG
Var1
STATE
OUTPUT
    ORG 20h
    DS 1
    BIT Var1.0
    BIT P1.0

CSEG
    ORG 0h
    AJMP START
    ORG 0Bh
    AJMP INTERRUPT

START
    MOV IE, #82h
    MOV TMOD, #01
    MOV TH0, #FEh
    MOV TL0, #0Ch
    SETB STATE
    SETB TR0

LOOP
    NOP
    SJMP LOOP

INTERRUPT
    CLR TR0
    MOV TH0, #FEh
    MOV TL0, #0Ch
    SETB TR0
    CPL STATE
    MOV C, STATE
    MOV OUTPUT, C
    RETI
    END
```
Machine language (1945)

Assembly language (1950)

High-level prog. langs. (1970)

PlayerControl ()
{
    euler = Vector3::GetZero();
    speed = 0.2;
    turnSpeed = 10.0;
    maxTurnLean = 50.0;
    maxTilt = 50.0;
    sensitivity = 10.0;
    forwardForce = 1.0;
}

virtual void Start () {
    // Get an access to another script attached to the same GameObject
    missileLauncher = GetComponent<MissileLauncher>();
}

virtual void Update () {
    for (int touchIndex = 0; touchIndex < Input::GetTouchCount(); touch
    {
        Touch touch = Input::GetTouch(touchIndex);
        if (touch.phase == TouchPhase::Moved)
        {
            speed = touch.position.y / Screen::height;
            guiSpeedElement.position = Vector3 (0, speed, 0);
        }
        if (touch.phase == TouchPhase::Ended)
        {
            missileLauncher->Fire();
        }
    }
}

virtual void FixedUpdate () {
    rigidbody.AddRelativeForce(0, 0, speed * forwardForce);
    Vector3 accelerator = Input::GetAcceleration();
Machine language (1945)
Assembly language (1950)
High-level prog. langs. (1970)
Modeling/graphical langs. (1985)
And what after that?

- **Machine language** (1945)
- **Assembly language** (1950)
- **High-level prog. langs.** (1970)
- **Modeling/graphical langs.** (1985)
Let’s concentrate on developing complex reactive systems

(term introduced with Pnueli 1985)

... which interact heavily with users or with other systems
Speedway to success
The actual development process
Specification Gridlock: A Closer Look
Specification Gridlock: The root of the problem

Specifiers
- interpret requirements
- create specification

Behavior!

Implementors
- interpret specification
- create hardware & software
Section 2.7.6: Security (~ page 10)
“If the system sends a signal hot then send a message to the operator.”

Section 9.3.4: Temperatures (~ page 150)
“If the system sends a signal hot and T>60°, then send a message to the operator.”

Summary of critical aspects (~ page 650)
“When the temperature is maximum, the system should display a message on the screen unless no operator is on the site except when T<60°.”
Statecharts (1984) were invented, at least in part, to help alleviate this problem.
Actually, we “program” all the time, though not necessarily computers…

And we use scenarios, examples, implicit instructions, analogies, constraints, etc.
The recent **scenario-based** approach (1999 and on) brings programming a lot closer to the way humans prescribe and describe behavior.

**Multi-modal**: includes mandatory, possible and forbidden behavior.
A live sequence chart (LSC)
Have several non-graphical versions of this (e.g., Java, C++)

Approach called more generally Scenario-Based (or Behavioral) Programming
How to most naturally construct LSCs?

I. Construct chart directly

II. “Play in” behavior from realistic graphical interface
III. Use Natural Language

Can start from scratch and go all the way to a full executable
IV. Use "Show & Tell"

Combine NL with play-in
Commercial break

New EdX online course
Liberating Programming:
System Development for Everyone
But,..... wouldn’t it be really nice if the process of programming a computer could be **two-way**, and the programming environment would be endowed with powerful **human-like wisdom**?

It would then become almost an **equal partner**, helpful and concerned, like human members of the system development team.
Indeed, humans can do a lot more…

(health care robot; credit: A. Marron)

- **Notice irregularities, unexpected properties:**
  
  “The arm movement is not smooth!”
  “Hear that strange noise when it turns”

- **Detect missing requirements, assumptions:**
  
  “Will it understand the voice of a hoarse patient?”
  “Can it process voice commands with the TV on?”

- **Ask (& answer) hard “what if” and “why” questions:**
  
  “Will a loud command from the TV confuse it?”
  “Why is it just walking around? Is it looking for something?”

- **Use broad knowledge and free association:**
  
  “Recently a pacemaker was remotely hacked. Can this happen here?”

- **Exhibit creativity, unusual thinking (outside the box):**
  
  “Let’s see what happens if I ask it to fetch something that’s glued to the table…”
We call such a futuristic approach to programming “Wise Computing”

It entails all that, and lots more...

arXiv, Jan 2015; and IEEE Computer, Feb 2018
From a tool to a proactive partner

A.
Current Practices

Stakeholders  Tools  Systems

B.
Wise Computing

Stakeholders  Systems
Main Research Directions:

- Formalization
- Analysis
- Interaction
- **Common Formalism:** Statecharts and LSCs at its heart, but with much more, intended to capture all relevant knowledge.

- **Analysis Engine:** proactive, uses heavy-duty learning, verification, SMT solving, etc., mimics human skills.

- **Interaction Language & Engine:** two way, multiple abstraction levels, natural language, captures all level of communication with human team.
Two demos of proof-of-concept wise development suite (mainly proactive analysis)

Concept and simple example: 12 min.

Cash coherence protocol: 18 min.
Main acks:
Amir Pnueli, Werner Damm, Rami Marelly,
Shahar Maoz, Assaf Marron, Smadar Szekely,
Gera Weiss, Michal Gordon, Guy Katz

Thank you for listening
Thank you for listening

😊