Industrialization of Software Production?

Jakob Rehof
Technical University of Dortmund
Fraunhofer-ISST Dortmund

In honor of Neil D. Jones, Copenhagen June 16, 2014
A Jones‘ian Topic

• Within a formulated vision of SWP-industrialization:
  – Pioneer in programs as data objects
  – Pioneer in program analysis and algorithms and complexity
  – Pioneer in meta-computation and partial evaluation
If you Google „car production process“ ...
If you Google „software production process“ ...
If you Google „software productivity“ ...
The Paradox

• Software technology is generally recognized as a major contributor to production automation

• ... only not when it comes to the production of software!
Intelligence?

**Code monkey** may refer to:
- Code monkey, term for a computer programmer who can perform trivial or repetitive programming tasks or a reference to a job that treats even experienced programmers in a way that trivializes their problem-solving abilities.
- A derogatory term to express "even a monkey could write this code", which is often resented by computer programmers as it trivializes the complexity of programming. This term also draws an analogy from the infinite monkey theorem involving monkeys typing ad infinitum to produce *Hamlet*.
- Code Monkeys, an animated television series
- "Code Monkey" (song), by Jonathan Coulton

**Monkey** sometimes used with adjectives other than **Code**:
- "adjective" monkey, a non-derogatory term for a skilled programmer who is tasked with performing a more trivial or repetitive computer programming task in addition to projects more consistent with their skill set.
The Problem is Intrinsically Hard

- In principle
  - Computability theory
  - Complexity theory

- In practice
  - Separating the „monkey work“ from the creative parts
  - Software is still an enigmatic object
Significant Scientific Progress

• Program analysis
  – Model checking, abstract interpretation, SAT-solving, SMT-solving

• Computational logic
  – Automated methods in deduction, proof checking

• Program synthesis
  – Modern renaissance

• ...

• EXPTIME is the new PTIME (?)
DARPA is going to launch a program called MUSE (Mining and Understanding SoUware Enclaves).

The first step is for MUSE to suck up all of the world’s open-source software—hundreds of billions of lines of code—and organize it in a giant database. The thinking is that of the 20 billion lines of code written each year, most of it repeats something that lots of programmers all over the globe have already done. MUSE will assemble a massive collection of chunks of code that can perform almost any task anybody could ever think of and tag all the code so it can automatically be found and assembled.

One day computers may have that kind of dexterity and intuition; the DARPA program is a good first step in that direction. But the path to the automated, thinking computer will also require a shift in research priorities, from the currently popular focus on the question “What you can do with Big Data?” back to A.I.’s original, driving one: “How do you build machines that are broadly intelligent?” It’s certainly possible that machines may someday be able to program themselves, but in a generation in which even the nerdiest, most cloistered programmer in Silicon Valley continues to have a far better intuitive sense of the world than any computer does, that day still feels a long way away.

Gary Marcus, a professor of psychology at New York University, is a co-editor of the forthcoming book “The Future of the Brain.” Ernest Davis is a professor of computer science at New York University.
Current Personal Inspiration

- Synthesis from collections of components
- Research programme on **Combinatory Logic Synthesis (CLS)**
  - Inhabitation (provability) in combinatory logic as foundation for component-based synthesis
  - Important ingredient: *meta-computation* and *staged computation* in a type-theoretic framework
A Path Beginning with NDJ

Jones (1980‘s) et.al.:  
• Partial Evaluation  
• Futamura projections  
• Self application  
• Binding time analysis

Staged computation  
Meta-computation  
Untyped setting

Davies & Pfenning (JACM 2001)  
• Modal Analysis of Staged Computation  
• Modal operator $\Box \tau$ as „code of type $\tau$“  
• Logical model of quote

Type-theoretic (logical)  
account of staged computation

BTW: Path from M. Felleisen to T. Griffin reminiscent of this?
Combinatory Logic Synthesis

- Finite Combinatory Logic with Intersection Types. With P. Urzyczyn. TLCA 2011.
Staged Composition Synthesis (ESOP'14)

Modal types $\sigma ::= \Box \tau \mid \sigma \rightarrow \sigma'$, where $\tau$ ranges over L1-types

Metalanguage L2 (adapted from $\lambda_e^{\Box,\rightarrow}$) over L1:

$$\frac{\Delta; (\Gamma, x : \sigma) \vdash_{L2} \lambda x : \sigma. M : \sigma \rightarrow \sigma'}{\Delta; \Gamma \vdash_{L2} \lambda x : \sigma. M : \sigma \rightarrow \sigma'} (\rightarrow I)$$

$$\frac{\Delta; \Gamma \vdash_{L2} M_1 : \sigma \rightarrow \sigma' \quad \Delta; \Gamma \vdash_{L2} M_2 : \sigma}{\Delta; \Gamma \vdash_{L2} (M_1 M_2) : \sigma'} (\rightarrow E)$$

$$\frac{\Delta; \Gamma \vdash_{L2} \Box M_1 : \tau}{\Delta; \Gamma \vdash_{L2} \Box M_1 : \tau} (\Box I)$$

$$\frac{\Delta; \Gamma \vdash_{L2} M_1 : \Box \tau \quad (\Delta, u : \tau); \Gamma \vdash_{L2} M_2 : \sigma}{\Delta; \Gamma \vdash_{L2} \text{letbox } u : \tau = M_1 \text{ in } M_2 : \sigma} (\Box E)$$

Reduction

$$\text{letbox } u = \text{box } T \text{ in } M \quad \rightarrow \quad M[u := T]$$

Semantic types \( s \quad ::= \quad b \mid s \to s' \mid s \cap s' \mid \Box t \)

Types \( \psi \quad ::= \quad \sigma \mid \psi \cap s \mid \Box \phi \mid \psi \to \psi' \mid \psi \cap \psi' \)

- Combinatory Logic C2 over C1:
  \[ \mathcal{C} = \{ \text{box } X_i : \Box \phi \} \cup \{ Y_j : \psi \} \]

- Leads to stratified modal logic S4

\[
@ \triangleq \lambda F : \Box (\alpha \to \beta). \lambda X : \Box \alpha. \\
\text{letbox } f : \alpha \to \beta = F \text{ in} \\
\text{letbox } x : \alpha = X \text{ in} \\
\text{box } f(x) \\
@ : \Box (\alpha \to \beta) \to \Box \alpha \to \Box \beta
\]

\[
\text{eval} \triangleq \lambda X : \Box \alpha. \text{letbox } x : \alpha = X \text{ in } x \\
\text{eval} : \Box \alpha \to \alpha
\]
\[ C = \{
\]
\[ \begin{align*}
\text{O} & \triangleq \text{box } 0 \quad : \quad \Box \text{TrObj} \\
\text{Tr} & \triangleq \text{box } \text{Tr} \quad : \quad \Box(\text{TrObj} \rightarrow D((R, R) \cap \text{Cart}, R \cap \text{Gpst}, R \cap \text{Cel})) \\
\text{tmp} & \triangleq \text{box } \text{tmp} \quad : \quad \Box(D((R, R), R, R \cap \alpha) \rightarrow R \cap \alpha \cap \text{ms})
\end{align*}
\]
\[ \}
\]
\[ \mathcal{D} = \{
\]
\[ \begin{align*}
\odot & \quad : \quad \Box(\alpha \rightarrow \beta) \rightarrow \Box \alpha \rightarrow \Box \beta \\
\bullet & \quad : \quad \Box(\beta \rightarrow \gamma) \rightarrow \Box(\alpha \rightarrow \beta) \rightarrow \Box(\alpha \rightarrow \gamma) \\
c12fh & \quad : \quad (\Box(R \cap \text{Cel}) \rightarrow \Box(R \cap \text{Fh})) \cap \text{Conv} \\
\lozenge & \quad : \quad \Box(\alpha \cap \text{ms}) \rightarrow (\Box \alpha \rightarrow \Box \beta) \cap \text{Conv} \rightarrow \Box(\beta \cap \text{ms})
\end{align*}
\]
\[ \}
\]
Combinator bindings for $\Diamond$:

$\bullet \quad \triangleq \quad \lambda G : \Box(\beta \to \gamma). \lambda F : \Box(\alpha \to \beta).
\text{letbox } f : \alpha \to \beta = F \text{ in }
\text{letbox } g : \beta \to \gamma = G \text{ in box} (\text{fn } y : \alpha \Rightarrow (g (f y)))$

$c12fh \quad \triangleq \quad \lambda z : \Box R.
\text{letbox } u : R = z \text{ in }
\text{box} (\text{let } x : R = u \text{ in } x \ast (9 \text{ div } 5) + 32)$

$\diamond \quad \triangleq \quad \lambda z : \Box \alpha. \lambda F : \Box \alpha \to \Box \beta. (F z)$
Query: \( C \cup D \vdash_{C_2} \square (R \cap F h \cap m s) \)

Solution: \((\text{tmp} \@ (\text{Tr} \@ O)) \diamond c12fh\)

L2-reduction:

\[
(\text{tmp} \@ (\text{Tr} \@ O)) \diamond c12fh \longrightarrow^* \text{box(let } x : R = \text{tmp} (\text{Tr} 0) \text{ in } x*(9 \text{ div } 5) + 32)\]
patternFollower($E_1, E_2, E_3, E_4, E_5, E_6, E_7, E_8, E_9, E_{10}, E_{11})$ with

$E_1 =$ box createSingleTask

$E_2 =$ box createSequence

$E_3 =$ box createEventPair

$E_4 =$ box GatewayExlusiveOr

$E_5 =$ box createDeprecatedGatewayWithCondition

$E_6 =$ initializationLight(box createSingleTask, box GatewayParallelAnd,
box createGateway, setTouchSensorTask, setLightSensorProc)

$E_7 =$ zigZagCondition(box createSingleTask, readLightSensorTask,
box GatewayExlusiveOr, box createGatewayWithCondition, $E_1^Z, E_2^Z$,
box createSequence, turnLight)

$E_8 =$ stopMotors(box createSubProcess, box createSingleTask, stopMotorTask3,
stopMotorTask2, box GatewayParallelAnd, box createGateway)

$E_9 =$ readTouchSensorTask

$E_{10} =$ stopTouch

$E_{11} =$ newSignal

$E_1^Z =$ makeLeftTurn(box createSingleTask, stopMotorTask3, startMotorTask2Car,
box GatewayParallelAnd, box createGateway)

$E_2^Z =$ makeRightTurn(box createSingleTask, stopMotorTask2, startMotorTask3Car,
box GatewayParallelAnd, box createGateway)
Thanks
&
Congratulations, Neil