DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF COPENHAGEN
D I K U

Program Analysis and Transformation (PAT)

Robert Glück
University of Copenhagen

Block 4, 2020

Motivation & This Course

- Despite tremendous progress in hardware, the production of software is
  - manual
  - error prone
  - costly
- Exploding demand for software leds to low SW quality (and lack of skilled CS graduates ☺
- Methods for more automatized SW construction needed.
- This course: Foundations and methods that have the potential to raise program development to a more automated process.

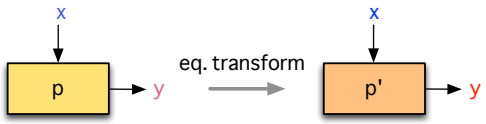
Approach: Programs as Data Objects

Build programs that treat programs as data objects:
- Analyze, transform, generate programs
- Manipulate programs by means of programs

Three basic operations on programs [Glück,Klimov'94]
1. Specialize: e.g. generating extensions, staging
2. Invert: e.g. reversible computing
3. Compose: e.g. deforestation, program slicing

▲ Programs are semantically the most complex data objects in the computer.

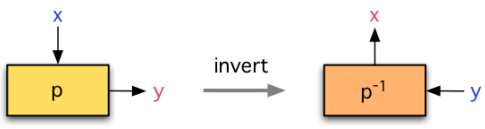
Familiar: Equivalence Transformation



- optimize in time, space...
- high abstraction levels
- close algorithmic gap
- preserve original semantics

- Example: compilers

Inversion of Programs



- choose which direction to write
- two programs from one program
- transform by inverse semantics

- Example: inverse interpreter [AbramovGlück'00], LR-program inverter [GlückKawabe'03]

Inversion of Programs

- choose which direction to write
- two programs from one program
- transform by inverse semantics

- Example: inverse interpreter [AbramovGlück'00], LR-program inverter [GlückKawabe'03]

Forward & Backward Computation

```
procedure main_fwd
  n += 4
  call fib
procedure main_bwd
  x1 += 5
  x2 += 8
  uncall fib
```

procedure forward

procedure backward

```
procedure fib
```

one procedure for both directions
⇒ reuse, code sharing

uncall: invokes inverse semantics

[ReillyFederighi65,LutzDerby82]

Specialization of Programs

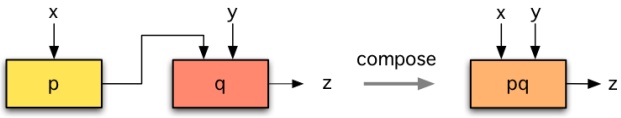
- **Partial evaluation**: technique to specialize programs by dividing a computation into two stages.

- Partial evaluators were designed & implemented. Scheme, ML, C, ...
- Literature: standard book [JonesGomardSestoft'93].
- Most intense research phase from mid 80ies to end 90ies.

Examples of Staged Computations

Program ***1-stage** computation*	*Generating Extension* ***2-stage** computation*
[interpreter] (p, data)	= [[translator] p] data
[parser] (grm, text)	= [[parser-gen] grm] text

Composition



- remove interface operations
- remove redundant operations
- reduce memory consumption

- Example: deforestation [Wadler'90]

"Generation of Algorithms by Algorithms"

Such investigations would appear to lead into the regions of meta-mathematics, where the problems deal with the generation of systems rather than the systems themselves.

Future work may depend very strongly on an understanding of just these problems arising in the "generation" of algorithms by other algorithms.

[Brown,Carr'54]

Metacomputation = Programs as Data + Metasystem Transition

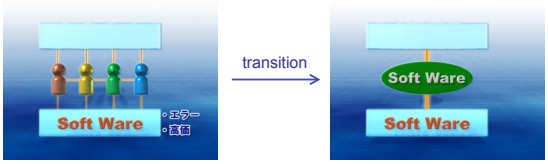
Programs-as-Data Operations [Glück,Klimov'94]

1. Specialize: e.g. staging computation
2. Invert: e.g. reversible computing
3. Compose: e.g. deforestation, unstaging

Metalevels of Programs on Programs

1. Programs as data objects (Jones)
2. Self-application as method (Futamura)
3. Metasystem transition (Turchin)

“Programs produce Programs”



Build programs that treat programs as data objects

- Automatically manipulate programs by programs
- Analyze & transform programs by programs
- Metacomputation – a long-term goal in CS

▲Programs are semantically the **most complex** form of **data objects** in the computer.