The Practical Uses of TransLucid

John Plaice and Blanca Mancilla

School of Computer Science and Engineering
University of New South Wales, Australia

Toon Verwaest
Software Composition Group
University of Bern, Switzerland

What is TransLucid?

- Declarative programming language
- Expressions vary according to an arbitrarily-dimensional context

The Practical Uses of TransLucid

- Programming language in its own right
- Coordination language
- Target language for a variety of paradigms

  Potentially adding context-awareness

The Practical Uses of TransLucid

- Recursive functions
- Context-aware constants and operations
- Functional programs
- Context-aware programs

Recursive Functions

```
infixn   "=="    "operator==" 1;;
infixl   "+"     "operator+" 2;;
infixl   "-"     "operator-" 2;;
infixl   "*"     "operator*" 3;;

%%

fact =   if #0 _==_ 0 then 1
         else #0 * fact@[0:#0-1]
         fi;;

%%

fact@[0:4];;

→ 24;;
```

Recursive Functions

```
infixn   "=="    "operator==" 1;;
infixl   "+"     "operator+" 2;;
infixl   "-"     "operator-" 2;;
infixl   "*"     "operator*" 3;;

%%

fact =   if #0 _==_ 0 then 1
         else #0 * fact@[0:#0-1]
         fi;;

%%

fact@[0:4];;

→ 24;;
```

Key structure: **hyperdaton**

Context: dynamically generated lazy tuples (dimension, value)

Recursive Functions

```
infixn   "=="     "operator==" 1;;
infixl   "+"      "operator+" 2;;
infixl   "-"      "operator-" 2;;
infixl   "*"      "operator*" 3;;
%%

ack =    if #0 _==_ 0 then #1 + 1
         elseif #1 _==_ 0
         then ack@[0:#0-1, 1:1]
         else ack@[0:#0-1, 1:ack@[1:#1-1]]

%%
ack@[0:3, 1:4];;

→ 125;;
```

Context-dependent Constants

```
library “verb”;;
%%

%%
verb<envoyer> @
      [outmode: “indicative”,
       outtense: “future”,
       outperson: “1s”];;

verb<enverrai> @
      [inmode: “indicative”,
       intense: “future”,
       inperson: “1s”];;

→   verb<enverrai> ;;
    verb<envoyer> ;;
```

Functional Programming

```
add 3 4
where
   add x y = x+y;;
end where;;
```

Functional Programming

An n-argument function can be seen as an (at least) n-dimensional hyperdaton

A function call corresponds to looking up the value of that hyperdaton in the appropriate context

Functional Programming

- Single hyperdaton called F, which varies in the *fun* dimension
- Partially applied functions: args and count

```
add 3 4
where
      add x y = x+y;;
end where;;
```

[fun: “add”, args: 2, count: 0]
[fun: “add”, args: 2, count: 1, 1: 3]
[fun: “add”, args: 2, count: 2, 1: 3, 2: 4]

→ (#1 + #2) @ [1: 3, 2: 4]

Hyperdatons of Functions

```
dimension "d";;

%%

pow =  if #d _==_ 0 then \x : x
       else \x : x * (pow@[d:#d-1])(x) fi;

odd =  if #d _==_ 0 then 1
       else odd@[d:#d-1] + 2 fi;

%%

(pow@[d:4])(odd@[d:3]);;
```

→ 7^5 = 16807

Context-Aware Programming

```
%%

contextin = external;;
contextout = contextin @
    if #time _==_ 0 then []
    else [ x : (#x * 2) @ [ time : #time – 1 ] ] fi;;

%%

contextout;
```

TransLucid

- Programming language in its own right
- Coordination language
- Target language for a variety of paradigms
  Potentially adding context-awareness

- Recursive functions
- Context-aware constants and operations
- Functional programs
- Context-aware programs

Turing Completeness

- Clear since TransLucid can be used to write recursive programs.
- Declarative form of **unlimited register machines** (URM)

Unlimited Register Machines

- Infinite set of registers R1 to Rn
- Program is a finite sequence of instructions I1...Im

- Z(i): set register i to 0, goto next
- S(i): add 1 to register i, goto next
- T(i, j): copy register j into register i, goto next
- J(i, j, k): If registers i and j are equal, goto k, otherwise goto next

Unlimited Register Machines

Program:

%%

X @ [0: 1] = Translate(Instruction_1)
…
X @ [0: m] = Translate(Instruction_m)

X = #1

%%

X @ [0 : 1, 1: i_1, … , n: i_n]

Unlimited Register Machines

Translate:

- Z(i) → X @ [0: #0+1, i: 0]
- S(i) → X @ [0: #0+1, i: #i+1]
- T(i, j) → X @ [0: #0 + 1, i: #j]
- J(i, j, k) → if #i _==_ #j
  then X @ [0: k]
  else X @ [0: #0 + 1] fi

•Z(i): set register i to 0, goto next
•S(i): add 1 to register i, goto next
•T(i, j): copy register j into register i, goto next
•J(i, j, k): If registers i and j are equal, goto k, otherwise goto next