Delimited Continuations in Soar

(How to think about several things at once without getting confused)

Clarification: Concurrency vs. Parallelism

Parallelism:

- like having two or more brains
- does the same thing faster
- the opportunity of distributing computation

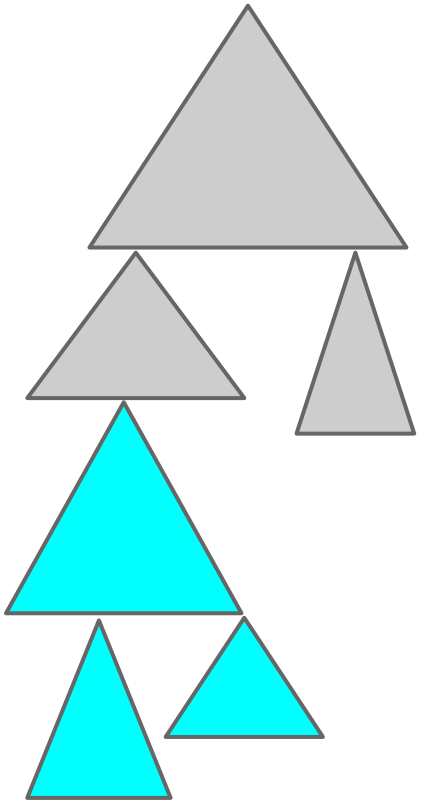
Concurrency:

- like having two or more hands
- usually has context switching costs
- controlling attention to interleave tasks
- the problem of dealing with distributed things

Benefits of easy-to-use concurrency

- Juggle tasks
- Interleave thinking about
  "what I'm doing over here"
  with thinking about
  "what they're doing over there"
- Take the perspective of everything you meet
- Explicitly balance deliberating and executing

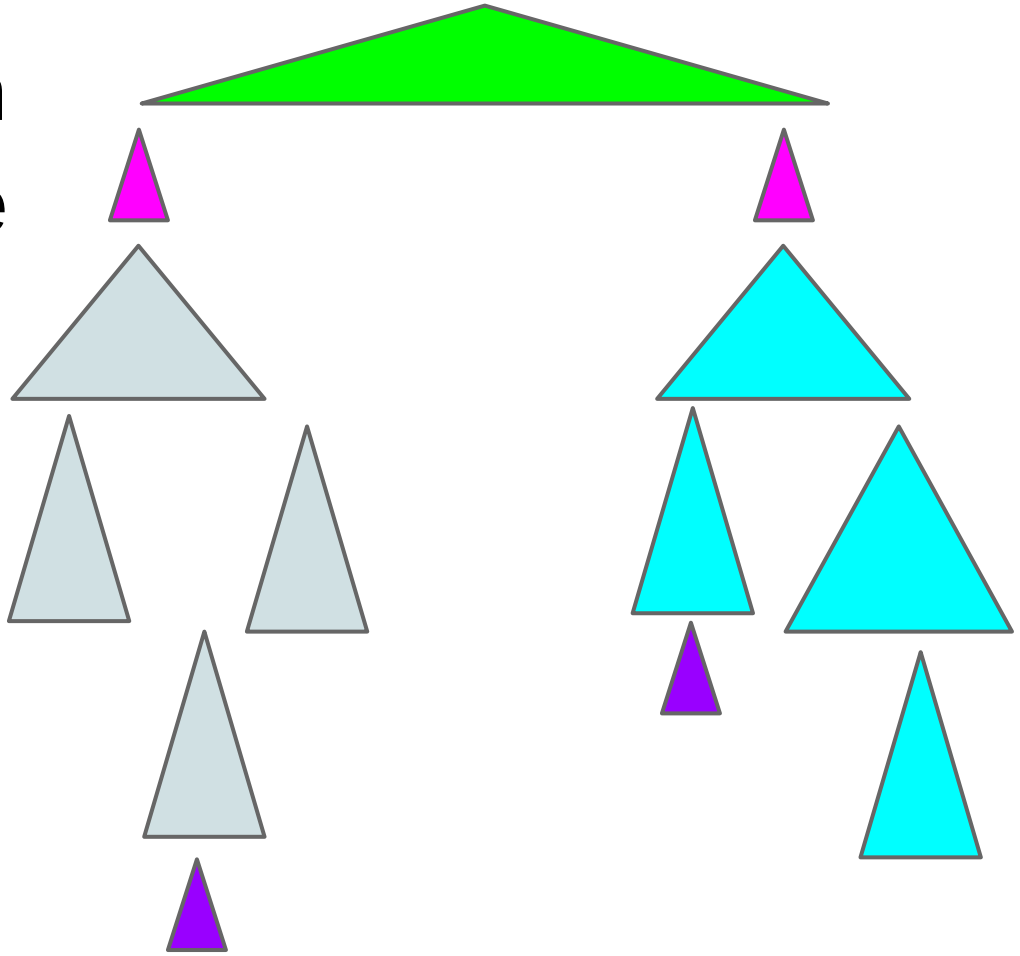
Concurrency Problems



- Real-time performance
  The supertask needs to run the subtask "often enough"
- Where to put the state
  The subtask needs to choose a stable place to store state

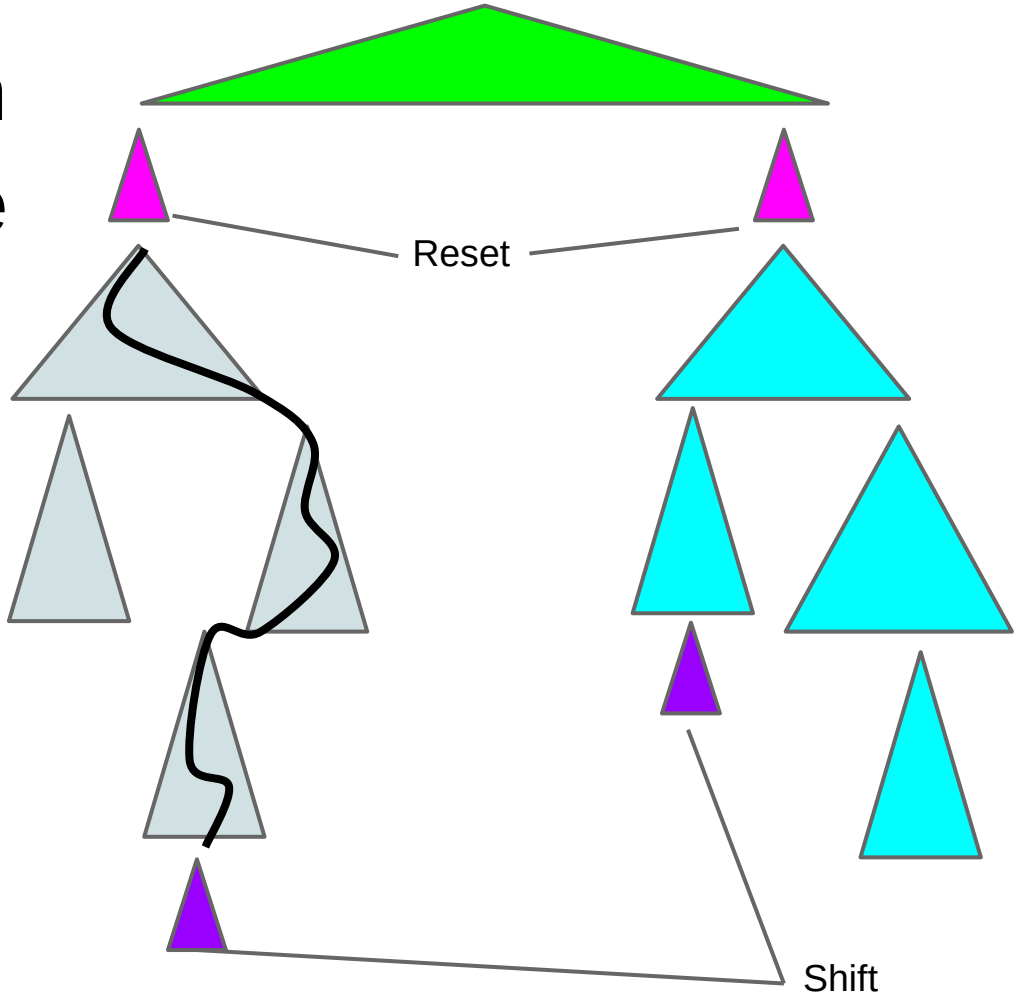
What delimited cont's are like

1. operating system
2. kernel/userspace boundary
3. userspace apps
4. system calls



What delimited cont's are like

1. operating system
2. kernel/userspace boundary
3. userspace apps
4. system calls
5. userspace stack (delimited continuation)



Lambda calculus with shift and reset

- A simple (15 rules) first order operational semantics.
- Easy to translate into Soar.

Based on "An Operational Foundation for Delimited Continuations"

by Biernacka, Biernacki and Danvy

```
run[t_] := eval[t, empty, hole, hole]
eval[int[n_], e_, C1_, C2_] := cont1[C1, int[n], C2]
eval[var[x_], e_, C1_, C2_] := cont1[C1, lookup[e, x], C2]
eval[lam[x_, t_], e_, C1_, C2_] := cont1[C1, clo[x, t, e], C2]
eval[app[t0_, t1_], e_, C1_, C2_] := eval[t0, e, appk1[t1, e, C1], C2]
eval[succ[t_], e_, C1_, C2_] := eval[t, e, succk[C1], C2]
eval[shift[x_, t_], e_, C1_, C2_] :=
eval[t, bind[x, cap[C1], e], hole, C2]
eval[reset[t_], e_, C1_, C2_] := eval[t, e, hole, compose[C1, C2]]
cont1[hole, v_, C2_] := cont2[C2, v]
cont1[appk1[t1_, e_, C1_], v_, C2_] := eval[t1, e, appk2[v, C1], C2]
cont1[appk2[clo[x_, t_, e_], C1_], v_, C2_] :=
eval[t, bind[x, v, e], C1, C2]
cont1[appk2[cap[C1p_], C1_], v_, C2_] := cont1[C1p, v, compose[C1, C2]]
cont1[succk[C1_], int[v_], C2_] := cont1[C1, int[v + 1], C2]
cont2[compose[C1_, C2_], v_] := cont1[C1, v, C2]
cont2[hole, v_] := v
```

Concurrency Example

```
/* Decompression code */
    while (1) {
        c = getchar();
        if (c == EOF)
            break;
        if (c == 123) {
            len = getchar();
            c = getchar();
            while (len--)
                emit(c);
        } else
            emit(c);
    }
    emit(EOF)
```

```
/* Parser code */
    while (1) {
        c = getchar();
        if (c == EOF)
            break;
        if (isalpha(c)) {
            do {
                add_to_token(c);
                c = getchar();
            } while (isalpha(c));
            got_token(WORD);
        }
        add_to_token(c);
        got_token(PUNCT);
    }
```

Demo

Gold and Coal

Gold:

- Towards a general-purpose reusable library for concurrency in Soar
- Implementing operational semantics of little languages in Soar is easy and fun
- Can chunk over (cooperative) task time slices

Coal

- Example isn't particularly agentish
- Currently fragile, brittle and arcane

Future work

- Build backtracking search via delimited cont's
- More sophisticated schedulers (e.g. Kiselyov and Shan's ZipperFS)
- Delimited cont's to implement a Discrete Event Simulations
- Delimited continuations might offer a new mechanism for modularity in agents
- Delimited cont's to build new operators

Thank you very much
The code is on github

johnicholas@johnicholas.com
http://github.com/johnicholas/learn_to_soar

References

"An Operational Foundation for Delimited Continuations in the CPS Hierarchy". Małgorzata Biernacka, Dariusz Biernacki, Olivier Danvy. BRICS Report Series RS-05-24

"Coroutines in C", Simon Tatham. http://www.chiark.greenend.org.uk/~sgtatham/coroutines.html

"From Interpreter to Logic Engine by. Defunctionalization". Biernacki Dariusz. Danvy Olivier. BRICS Report Series. RS-04-5

"Delimited continuations in operating systems" Oleg Kiselyov and Chung-chih Shan Proc. CONTEXT2007: 6th International and Interdisciplinary Conference on Modeling and Using Context. Roskilde, Denmark, August 20-24, 2007. Lecture Notes in Artificial Intelligence 4635, pp. 291-302.

Lambda calculus

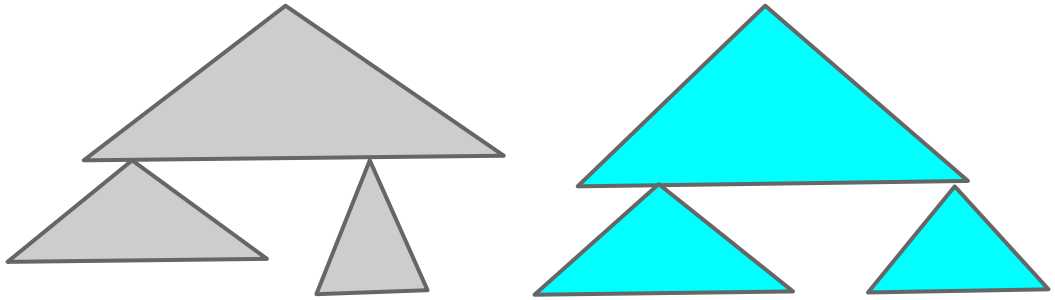
Based on BRICS-RS-05-24

"An Operational Foundation for Delimited Continuations in the CPS Hierarchy"

by Biernacka, Biernacki and Danvy

```
eval[Literal[m_], env_, k_] := go[k, m]
eval[Variable[x_], env_, k_] := go[k, Lookup[env, x]]
eval[Lam[x_, exp_], env_, k_] := go[k, Closure[x, exp, env]]
runfun[Closure[x_, exp_, env_], v_, k_] := eval[exp, Bind[x, v, env], k]
eval[App[exp0_, exp1_], env_, k_] :=
eval[exp0, env, Appk1[exp1, env, k]]
go[Appk1[exp1_, env_, k_], v_] := eval[exp1, env, Appk2[v, k]]
go[Appk2[v1_, k_], v2_] := runfun[v1, v2, k]
eval[Succ[exp_], env_, k_] := eval[exp, env, Succk[k]]
go[Succk[k_], v_] := go[k, v + 1]
go[Halt, v_] := v
```

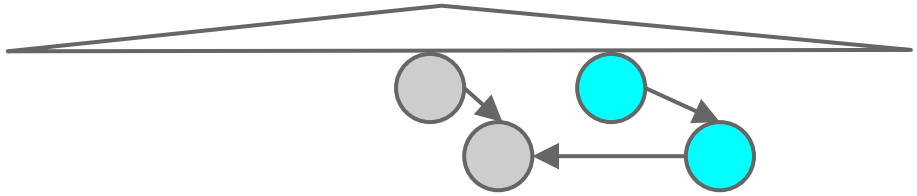
Strategies for concurrency:
Michigan Style



"Blow away" substates to context switch

- Symmetrical, elegant, automatic
- Programmers need to relax about their control of control flow
- Subtasks still need to tuck state away somewhere ad-hoc to achieve continuity
- Interruptions may prevent chunking

Strategies for concurrency:
NGS



Maintain long-term goals as an explicit data structure rather than on the architectural stack

- Goals can be non-hierarchical
- Can switch tasks faster than Michigan (?)
- Models expert performance; no learning story
- Programmer needs to learn NGS library API