The FunLoft Language

Frédéric Boussinot
MIMOSA Project, Inria-Sophia
http://www.inria.fr/mimosa/rp

March 2007

Common work with Frédéric Dabrowski

ACI ALIDECS

Summary

1. FunLoft
2. Implementation
3. Multicore Programming
4. Future Work

FunLoft

- Inductive data types - First order functions
- References - Threads - Events
- Schedulers + `link`, `unlink`

$$
\begin{array}{lcll}
p & ::= & x \mid C(p,\ldots,p) & \qquad (\textit{patterns}) \\
e & ::= & x \mid C(e,\ldots,e) \mid f(e,\ldots,e) & \\
 & & \mid \texttt{match}\ x\ \texttt{with}\ p -> e \mid \ldots \mid p -> e & \\
 & & \mid \texttt{let}\ x = e\ \texttt{in}\ e \mid \texttt{ref}\ e \mid\ !e \mid e{:=}e & \\
 & & \mid \texttt{cooperate} \mid \texttt{thread}\ f(e,\ldots,e) \mid \texttt{join}\ e \mid \texttt{stop}\ e & \\
 & & \mid \texttt{unlink}\ e \mid \texttt{link}\ s\ \texttt{do}\ e & \\
 & & \mid \texttt{event} \mid \texttt{generate}\ e\ \texttt{with}\ e \mid \texttt{await}\ e & \\
 & & \mid \texttt{get\_all\_values}\ e\ \texttt{in}\ e & \\
 & & \mid \texttt{loop}\ e \mid \texttt{while}\ e\ \texttt{do}\ e & \qquad (\textit{expressions})
\end{array}
$$

Synchronous π-Calculus

- Purely functional (no references). Unique scheduler

$$
\begin{array}{rcl}
p & ::= & x \mid C(p, \ldots, p) \\
e & ::= & x \mid C(e, \ldots, e) \mid f(e, \ldots, e) \\
 & & \mid \texttt{match}\ x\ \texttt{with}\ p - {>}e \mid \ldots \mid p - {>}e \\
 & & \mid \texttt{let}\ x = e\ \texttt{in}\ e \mid e || e \\
 & & \mid \texttt{event} \mid \texttt{generate}\ e\ \texttt{with}\ e \mid \texttt{present}\ e\ \texttt{then}\ e\ \texttt{else}\ e \\
 & & \mid \texttt{pre}\ e
\end{array}
$$

- R. Amadio, *A synchronous π-calculus*, http://www.pps.jussieu.fr/~amadio
- Resources usage (memory & CPU) is polynomial in the size of the input provided some static checks (F. Dabrowski's thesis)

PACT

- FunLoft (without `join`)
- References can be separated (using a type and effect system):
  - threads linked to the same scheduler never interfere (cooperation!)
  - Schedulers own references only shared by threads linked to them
  - Threads own private references only accessible by them
- Consequence: absence of data-races (two threads accessing the same reference asynchronously)
- TV'06 paper was considering only a limited version (unique scheduler)
- F. Dabrowski's thesis

Implementation

- Type inference and type checking $->$ code production in Loft/C (pthreads + GC)
- Distinction function/module - no recursive module
- Non-termination detection of recursive functions with inductive type parameters
- Instantaneous loop detection
- Stratification of references and events
- Control of thread dynamic creation
- $\sim$ 8000 lines of code

Multicore Programming

- How can a single application benefit from a multicore architecture?
- Multithreaded applications. Weak/Strong synchronisation between threads
- Benchmarks:
  - Prey/predator system with one native thread for all preys and one native thread for all predators.
  - Several rooms for migrating preys/predators: one native thread by room
  - *Game Of Life (GOL)* divided in several synchronised areas: one native thread by area. Strong synchronisation. Global determinism.

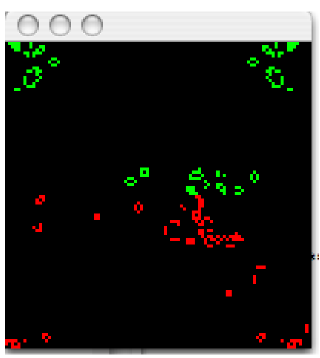
Synchronised Schedulers

- Asynchronous schedulers:
  - no sharing of memory (to avoid data races)
  - no event emitted from one scheduler to another scheduler (bounded size memory)
- Schedulers sharing same instants
  - no sharing of memory
  - shared events: events are common to synchronised schedulers
  - protocol for scheduler synchronisation (*distributed reactive machines* of SugarCubes/Junior)
- Syntax:

```
let s1 = scheduler
and s2 = scheduler
```

Multithreaded GOL

- Main differences with the one scheduler program:
  - Draw orders sent to the thread in charge of graphics
  - No global array of cells
  - Synchronised start of cells
- Difficult to get full benefit from multicore:
  - multi-threaded malloc
  - multi-threaded GC (Boehm's GC)
- Demo (10K cells, 500 instants, 1K cycles)



one scheduler	**two schedulers**
real 0m26.367s	real 0m20.944s
user 0m24.991s	user 0m26.548s
sys 0m0.381s	sys 0m0.626s

Conclusion & Future Work

- Resource control for S-π-calculus
- No data races in PACT
- Lack of formalisation: type inference, join primitive, synchronised schedulers
- Experimental implementation: Loft-C, pthreads, Boehm's GC
- Syntax for multithreaded applications running on multicore architectures
- Documentation + Available FunLoft v0.1
- Error messages!
- Specific automatic memory management?
- Language extension: exceptions? distribution (agents)?