ESUIF: An Open Esterel Compiler

Stephen A. Edwards

Department of Computer Science

Columbia University, New York

www.cs.columbia.edu/~sedwards

Not Another One…

- My research agenda is to push Esterel compilation technology much farther
- We still don't have a technique that builds fast code for all large programs
- No decent Esterel compiler available in source form

Quick History of Esterel Compilers

- Automata-based
  - V1, V2, V3 (INRIA/CMA) [Berry, Gonthier 1992]
  - Excellent for small programs with few states
  - Don't scale well
- Netlist-based
  - V4, V5 (INRIA/CMA)
  - Scales very nicely
  - Produces slow code for sequential programs
- Executables for these available at www.esterel.org
- Not open-source

Quick History of Esterel Compilers

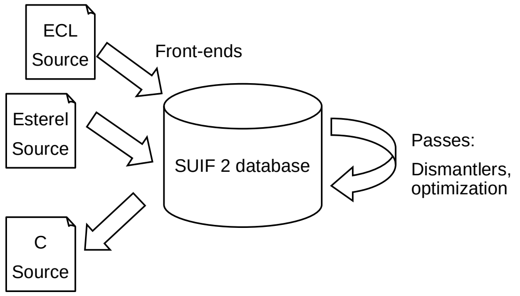
- Control-flow-graph based
  - EC [Edwards 1999, 2000, 2002]
  - Produces very efficient code for acyclic programs
- Discrete-event based
  - SAXO-RT [Weil et al. 2000]
  - Produces efficient code for acyclic programs
- Both proprietary & unlikely to ever be released
- Neither has V5's ability to analyze static cycles
  - Many valid programs are rejected

ESUIF

- New, open-source compiler being developed at Columbia University
- Based on SUIF 2 infrastructure (Stanford University)
- Divided into many little passes
- Common database represents program throughout

Open, Flexible Architecture

- Common database used throughout

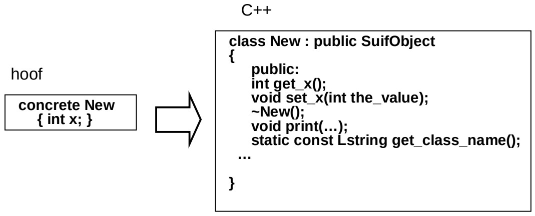


SUIF 2 Database

- Main component of the SUIF 2 system:
- User-customizable persistent, object-oriented database
- Written in C++
- Not the most efficient, but very flexible

SUIF 2 Database

- Database schema written in their own “hoof” language
- Automatically translated into C++



```
concrete New
    { int x; }
```

```
class New : public SuifObject
{
    public:
    int get_x();
    void set_x(int the_value);
    ~New();
    void print(…);
    static const Lstring get_class_name();
  …

}
```

Three Intermediate Representations

- Front end generates AST-like database
  - One-to-one mapping between classes and Esterel statements
- Dismantled into concurrent IC-like statements
  - Described next
- Scheduling produces C code
  - SUIF 2 has complete schema for C

Intermediate Representation

- Goal: simpler semantics than IC [Gonthier 1988]
- Slightly lower-level
- More symmetry between strong and weak abort
  - IC uses awkward implicit exceptions for weak abort
- More division between concurrency and exception handling

IR Primitives

- ***var* := *expr***
- **if (*expr*) { *stmts* } else { *stmts* }**
- ***Label*:**
- **goto *Label***

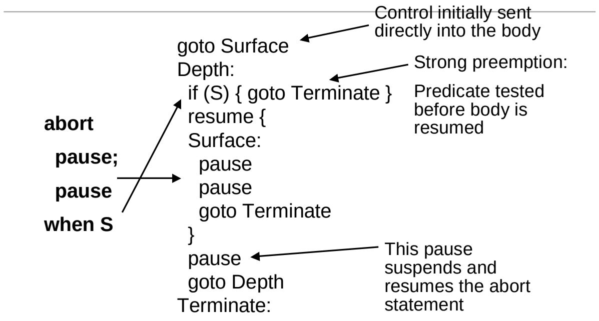
- **resume (*state-var*) { *stmts* }**
- **pause**

- **trapScope (*Handler-Label*) *T1*,…,*Tn* { *stmts* }**
- **fork *L1*, …, *Ln***
- **join**
- **thread (*exit-var*, *Join-Label*) { *stmts* }**
- **exitAt *n***

Pause and Resume

- Idea: single pair of primitives that implement ability to suspend and resume sequences of instructions
- Semantics:
  - pause sends control just past its enclosing resume
  - resume sends control to just after the last-executed pause
- Trivial translation into a C switch statement
- Simple enumeration of states (just pause statements)
- Strong and weak abort just tests before and after

Translating Strong Abort



```
abort
  pause;
  pause
when S
```

```
goto Surface
Depth:
  if (S) { goto Terminate }
  resume {
  Surface:
    pause
    pause
    goto Terminate
  }
  pause
  goto Depth
Terminate:
```

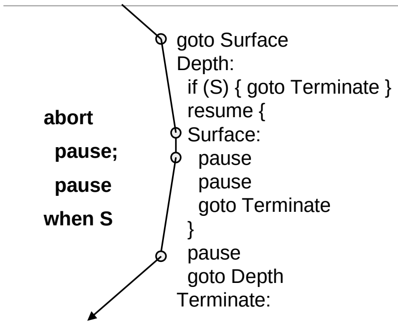
Control initially sent directly into the body

Strong preemption:

Predicate tested before body is resumed

This pause suspends and resumes the abort statement

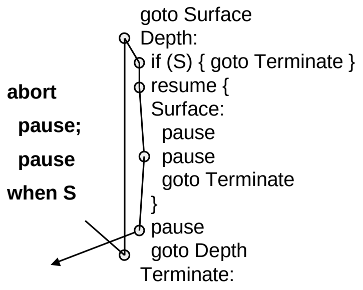
First Reaction



```
abort
  pause;
  pause
when S
```

```
goto Surface
Depth:
  if (S) { goto Terminate }
  resume {
  Surface:
    pause
    pause
    goto Terminate
  }
  pause
  goto Depth
Terminate:
```

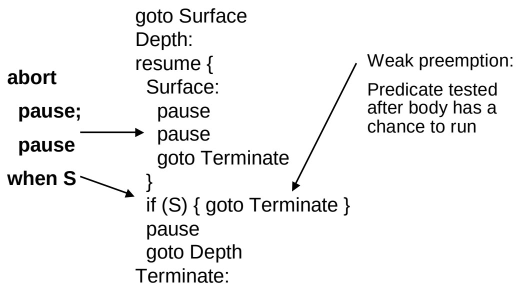
Second Reaction



```
abort
  pause;
  pause
when S
```

```
goto Surface
Depth:
  if (S) { goto Terminate }
  resume {
  Surface:
    pause
    pause
    goto Terminate
  }
  pause
  goto Depth
Terminate:
```

Translating Weak Abort



```
abort
  pause;
  pause
when S
```

```
goto Surface
Depth:
resume {
  Surface:
    pause
    pause
    goto Terminate
  }
  if (S) { goto Terminate }
  pause
  goto Depth
Terminate:
```

Weak preemption:

Predicate tested after body has a chance to run

Dismantling

- **Multiple passes dismantle AST-like Esterel into the IR**
- **Each dismantles a single Esterel statement**
- **Most are trivial**

Parallel, Trap, and Exit

- **Translation of exit differs depending on parallel behavior**

trap T in	**Does not terminate siblings**
exit T	**No prioritization of exits**
end	
trap T in	**Terminates siblings**
stmts || exit T	**Must worry about trap priorities**
end	

Parallel, Trap, and Exit

- Translation is tedious, but not difficult
- Uses Berry and Gonthier's encoding of exit levels:

0 = terminate

1 = pause

2 = exit innermost trap

3 = exit next innermost trap

4 = etc.

Ideas for Code Generation

- ESUIF does not currently have a back-end
- I am considering a few possibilities

Static Unrolling

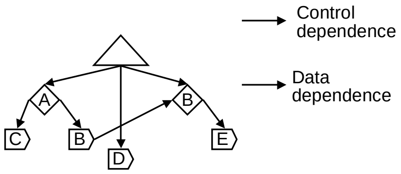
- Cyclic programs can always be evaluated by unrolling: $\mathrm{lfp}(F) = F(\bot)^n$
- Three-valued evaluation costly, not clear with control-flow
- Theorem (suggested to me by Berry)

  If a program is causal, then two- and three-valued evaluation will produce the same result
- Proof: F is monotonic, lfp does not contain ⊥

Program Dependence Graph

- Program Dependence Graph [Ferrante et al., TOPLAS 1987] is concurrent
  - Represents only control and data dependencies
  - Natural for Esterel because it represents concurrency

```
present A then
  emit B
else
  emit C
end;
emit D;
present B then
  emit E
end
```



Program Dependence Graph

- Idea: Represent Esterel program as a program dependence graph
  - Unroll to resolve cycles (duplicate code)
- Generate code that conforms to the program dependence graph
- Some PDGs do not require additional predicates when sequentialized [Ferrante et al., Steensgaard]
- Heuristics will have to be used to insert a minimum number of predicates in most cases

Discrete-Event Approaches

- Weil et al. [CASES 2000] have taken this approach
- Successful, but scheduler could be better
- Does not handle statically cyclic programs
- Techniques such as French et al. [DAC 1995] schedule as much as possible beforehand, but allow some dynamic behavior
- Idea: Generate an unrolled schedule and invoke unduplicated basic blocks more than once per reaction (solves causality and schizophrenia)

Conclusions

- ESUIF compiler under development at Columbia
  - Front-end completed
  - Most dismantlers written
  - Work beginning on back-end
- New intermediate representation
  - pause and resume primitives
- Some new ideas for code generation
  - Static unrolling with two-valued evaluation
  - Program Dependence Graph
  - Event-driven Approaches

For More Information

- Visit my website

http://www.cs.columbia.edu/~sedwards