Static Slicing of Reactive Programs

S. Ramesh
Department of Computer Science and Engineering,
Indian Institute of Technology Bombay,

Jointly with A. R. Kulkarni, Veritas India

with support from Centre for Formal Design and Verification of Software, IIT Bombay

Program Slicing

- Functional Decomposition technique
- Ease of debugging, testing and understanding
- Formal Verification - recent interest
- Sequential Program Slicing (Weiser '84)
- Notion of slicing criterion: $< pc, Var >$
- Definition: $slice(P)$ w.r.t. $< pc, x >$ is P' where,
  - P' is obtained from P by removing some statements
  - If P reaches pc then P' also reaches pc and
  - x has the same value in both P, P' at pc.
- Our focus is on Syntactic Static Slicing
- Other approahes: Dynamic, Semantic, Amorphous slicing

Example:

Slicing Criterion: $< write(sum), sum >$

```
Entry
read(n)
i:=1
sum:=0
pro:=1
while    i<=n
        sum:=sum+i
        pro:=pro*i
        i:=i+1
end while
write(sum)
write(pro)
Exit
```

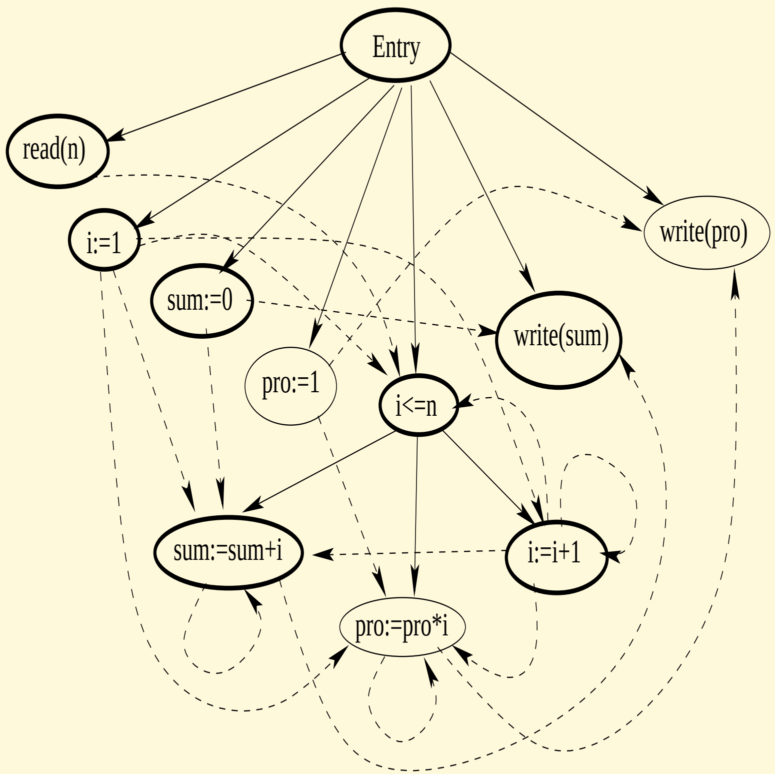
```
Entry
read(n)
i:=1
sum:=0

while    i<=n
        sum:=sum+i

        i:=i+1
end while
write(sum)

Exit
```

Issues

- Correctness: Slice includes **all** necessary statements
- Precision: **only** reqd. statements
- Only approximate slices possible (termination problem)
- P itself a slice

Computation of Slices

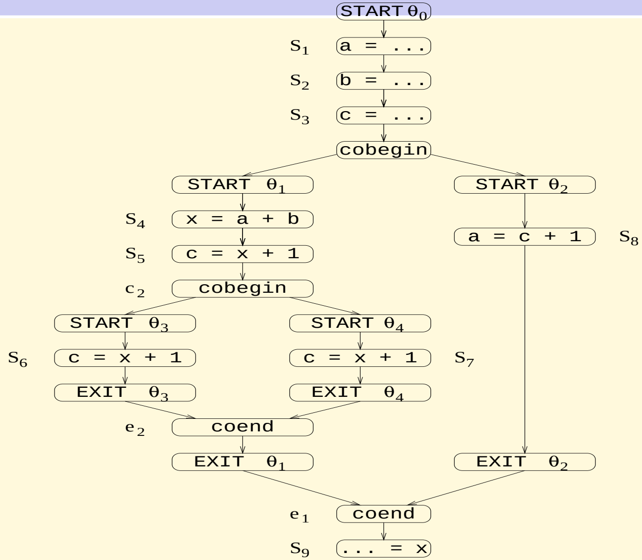
- Program Dependence Graph:
  - Control Dependence edges
  - Data Dependence edges
  - Example:



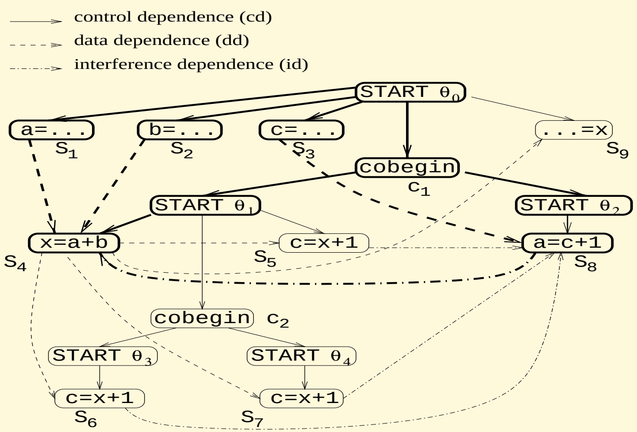
- Reachability Analysis

Concurrent Program Slicing

- Cheng ’97, Krinke ’98, Gowri and Ramesh ’00
- Slicing Criteria same
- More elaborate definition:
- Computation of Slices
  - Extended PDG - Threaded Program Dependence Graph (TPDG)
  - Example:



(a) A threaded control flow graph (TCFG)



(b) The corresponding threaded program dependence graph (TPDG)

Figure 3.1: Representation of threaded programs

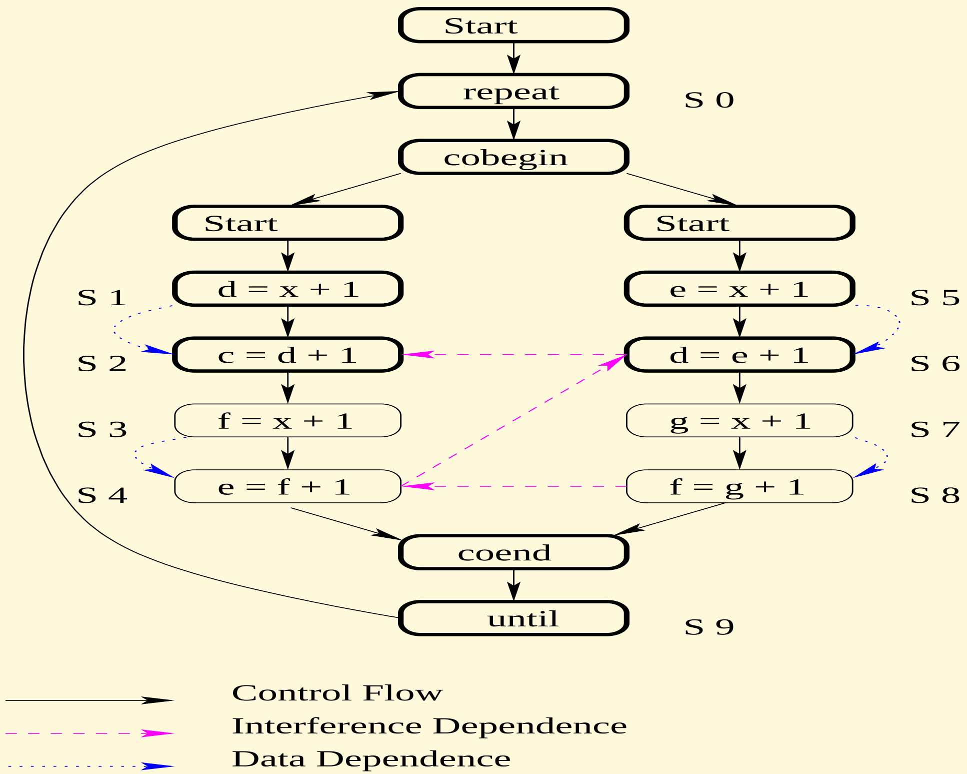
Computation of Slices

- Naive reachability analysis is imprecise
- S4 is interference dependent upon S8 and S8 upon S5
- S4 not dependent upon S5
- Interference dependency is not transitive
- Refined reachability analysis
- Notion of Trace Witness
  $(n_1, ..., n_k)$ in the threaded PDG is a trace witness provided it forms a subsequence of some execution trace in the program.
- it is part of a valid execution trace
- Definition of Slice:

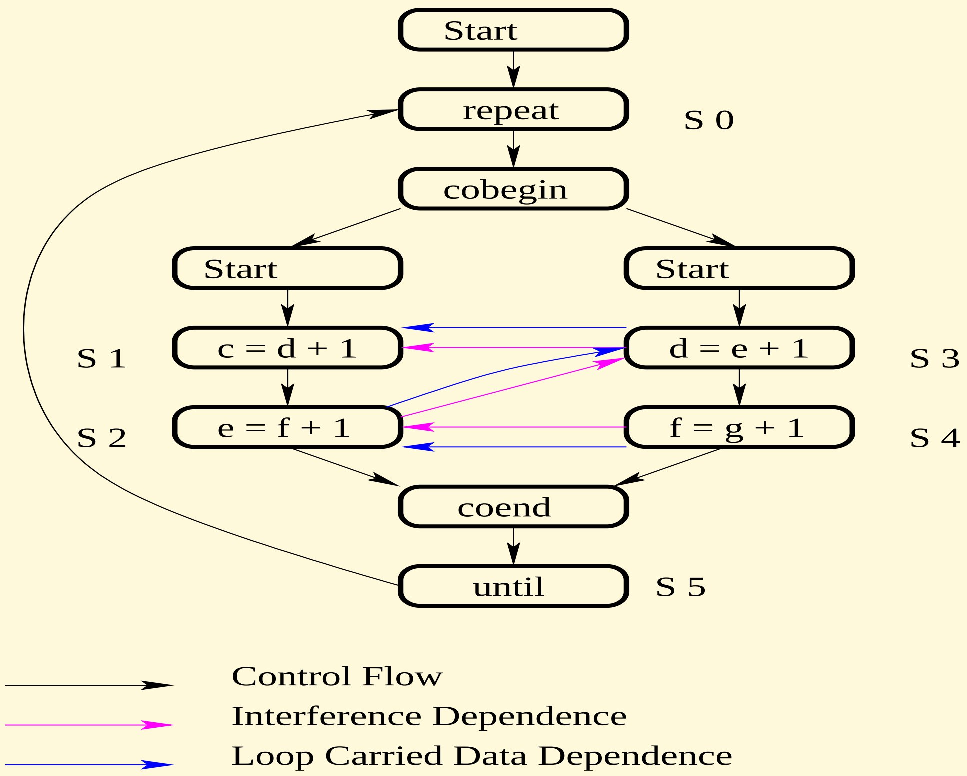
$$\text{Slice(p)} = \{q | q = n_1 \xrightarrow{d_1} n_2 \xrightarrow{d_2} \cdots \xrightarrow{d_{k-1}} n_k = p,$$
$$d_i \in \{cd, dd, id\}, (n_1, \cdots, n_k) \text{ is a trace witness}\}$$

Computation of Slices

- Sophisticated traversal algorithm
- Traversal carries information about nodes already visited in each thread
- Original algorithm due to Krinke.
- Does not work when threads are inside the loop
- inaccurate slices
- Data dependence to be classified as direct and loop-carried dependence



Start
repeat
S 0
cobegin
Start
Start
d = x + 1
S 1
e = x + 1
S 5
c = d + 1
S 2
d = e + 1
S 6
f = x + 1
S 3
g = x + 1
S 7
e = f + 1
S 4
f = g + 1
S 8
coend
until
S 9
Control Flow
Interference Dependence
Data Dependence


Start
repeat
S 0
cobegin
Start
Start
c = d + 1
S 1
d = e + 1
S 3
e = f + 1
S 2
f = g + 1
S 4
coend
until
S 5
Control Flow
Interference Dependence
Loop Carried Data Dependence

Slicing Algorithm

- state of execution recorded in tuple $[t_0, t_1, \ldots, t_n]$
- performs a backward traversal of TPDG
- for nodes y reached via edge $y \longrightarrow x$:
  - data or control dependence: add y to slice. update tuple.
  - interference dependence:
    - t = last node visited in y's thread
    - if trace witness exists for $\langle y, t\rangle$, add y. update tuple.
  - loop-carried data dependence: add y to slice. update tuple.

Complexity

- Exponential on the number of threads (in theory)
- Many optimisations possible (in practice)
- Inter-procedural slicing
- Nontrivial extension
- Implemented Java Slicer (Gowri's thesis)

Other works:

- slicing Promela (Millett, Teitelbaum '98)
- VHDL slicing (Clarke et al. '99)
- concurrency issues not addressed properly

Slicing Synchronous Reactive Programs

- Reactive programs are ones that maintain continuous interaction with the environment
- Contrast with transformational programs
- Termination is a bad behaviour
- Examples of reactive systems:
  - Operating systems functions
  - Hardware
  - Embedded Controllers

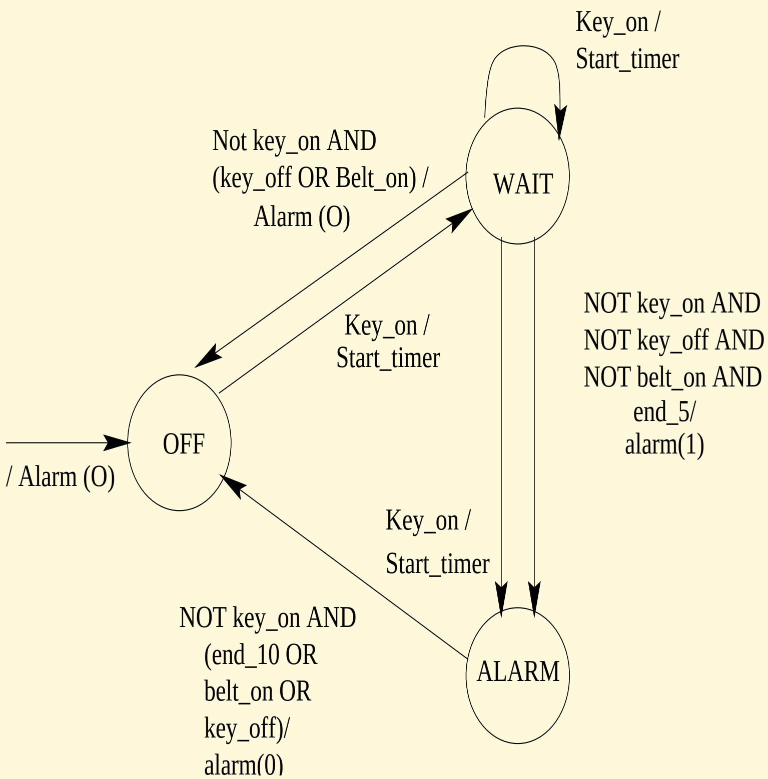
An Example

- ” Five seconds after the key is turned on, if the belt has not been fastened, an alarm will beep for five seconds or until the key is turned off ”

An Esterel Solution

```
module belt_control:
input reset, key_on, key_off, belt_on,
      end_5, end_10;
output alarm(boolean), start_timer;
loop
  abort
    emit alarm(false);
    every key_on do
      abort
        emit start_timer;
        await end_5;
        emit alarm(true);
        await end_10;
      when [key_off or belt_on];
      emit alarm(false);
    end
  when reset
end.
```

Behavior of this program:



Reactive Programs

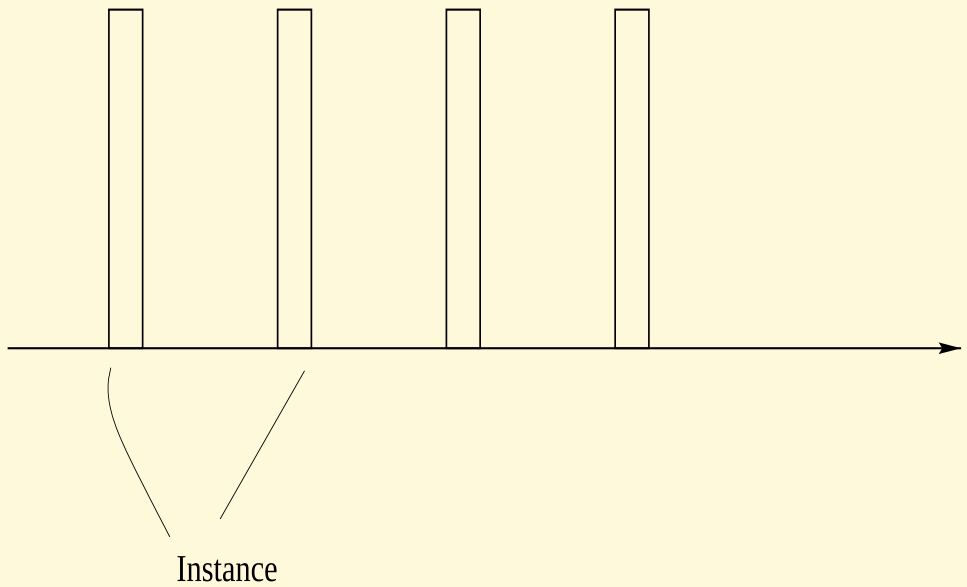
- reactive programs are event-oriented
- time or event ordering need to be preserved
- Events are more fundamental than program control points

Esterel

- a well-known language for programming embedded control programs
- used in avionics (Aerospatiale), DSP chips (TI France)
- Control flow quite different -back and forth from the enivironment and program

Esterel Execution Model

- execution is a series of **reactions**.
- invoked from an external 'main' program repeatedly at discrete points of time
- one reaction per invocation
- control returns after each reaction



Esterel Constructs

- many novel features
  - imperative paradigm, synchronous concurrency.
  - delay statements
  - instantaneous execution
  - Signal handling statements
  - Preemption and Exception handling statements
  - rich in control constructs:
    * preemption: `abort p when S`
    * suspension: `suspend p when S`
    * exception handling: `trap and exit`
  - Concurrent statements (synchronous concurrency)
  - Communication via broadcast signals
- Challenge for standard program analysis techniques

Synchronous Parallelism

$$\texttt{[stat1} \parallel \texttt{stat2} \parallel \texttt{stat3]}$$

- simultaneous (not concurrent) execution of all the statements
- signals are used for communication
- signal emitted by one thread is broadcast to all other threads
- terminates when every `stati` terminates
- no sharing of variables
- compare with asynchronous parallelism

Example:

```
[ emit S

|| present S then emit O1 else emit O2

|| present S then emit O3 else emit O4

]
```

`S,O1` and `O3` are simultaneously executed

Preemption Statements

strong abort

```
abort stat when S
```

- watchdog primitive
- The body `stat` is executed only when S is not present
- Presence of S instantaneously ‘kills’ the body
- No statement in `stat` is executed when S is present
- terminates either when either `stat` terminates or when S is present

Example:

```
abort
  pause;
  emit S1;
  pause;
  emit S2
when S
```

- emits S1 in the second instant and S2 in third instant if S is not present during these instants.
- if S is present in second instant then nothing happens; the whole statement exits.
- if S is not present in second instant and present in third instant then S1 is emitted in the second instant, terminates in the third instant; no S2 is emitted in the third instant
- S in the first instant is ignored `S` in the first instant is not ignored if you write `abort stat when immediate S`

Traps and exits

```
trap T in
  stat1
handle T do
  stat2
end trap
```

- Weak preemption primitive
- The body `stat1` may contain exit statement `exit T`
- execution starts with execution of `stat1`
- when `exit T` is encountered the control jumps to the `handle` statement
- `handle` statement is optional - control then returns to the statementfollowing the `trap` statement
- if `stat1` is terminated then the whole trap statement is exited - `stat2` is not executed

Slicing Reactive Programs

- Traditional slicing criterion not very natural
- Proposal for a new criterion
- Slicing Criterion: b, an output signal
- Slice of P w.r.t to b has the same ongoing behaviour as P as far as signal b is concerned
- That is, b is present in a computation of P iff it is present in a computation of $Slice(P)$
- $Slice(P)$ obtained from P, by removing statements
- More generally, $< S, b >$, S, a state(ment) and b a signal
- Slice of P w.r.t $< S, b >$ preserves behaviour w.r.t. b in all computations that reach state(ment) S.

Formal Definition:

- M, reactive program
- M_s is the slice w.r.t $< S, b >$ iff
  - M_s is obtained by removing zero or more states of M and
  - $\forall\sigma$, sequence of input signals
    - $(M[\sigma]/b) = (M_s[\sigma]/b))$
  - $M[\sigma]$, output sequence produced by M on input σ
  - $M[\sigma]/b$, sequence restricted to only b.
- Useful for formal verification

Home Page

Title Page

Page 26 of 38

Go Back

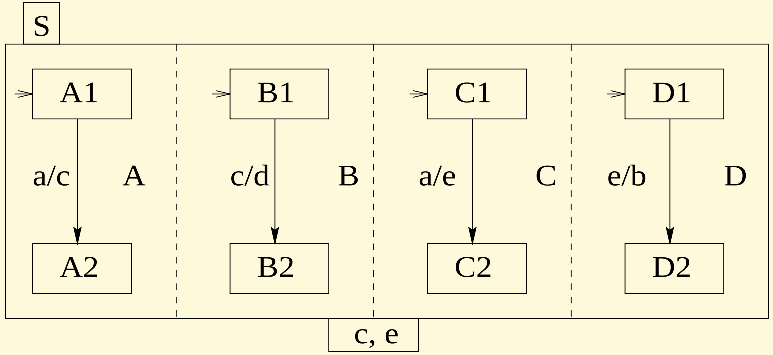
Full Screen

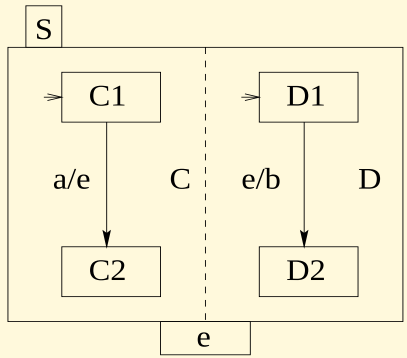
Close

Quit

Example:

An Argos Example





- Slice w.r.t b:
- Slice has the same behaviour as the original program as far as b

Computing Slices

- Inadequacy of classical dependency
- Very many new dependencies in Esterel
- interference control dependency (arises due to trap statements)
- time dependency (due to pause statement)
- dependency graph is generalised
- Synchronous Threaded Program Dependency Graph (STPDG)

Home Page

Title Page

◀◀ ▶▶

◀ ▶

Go Back

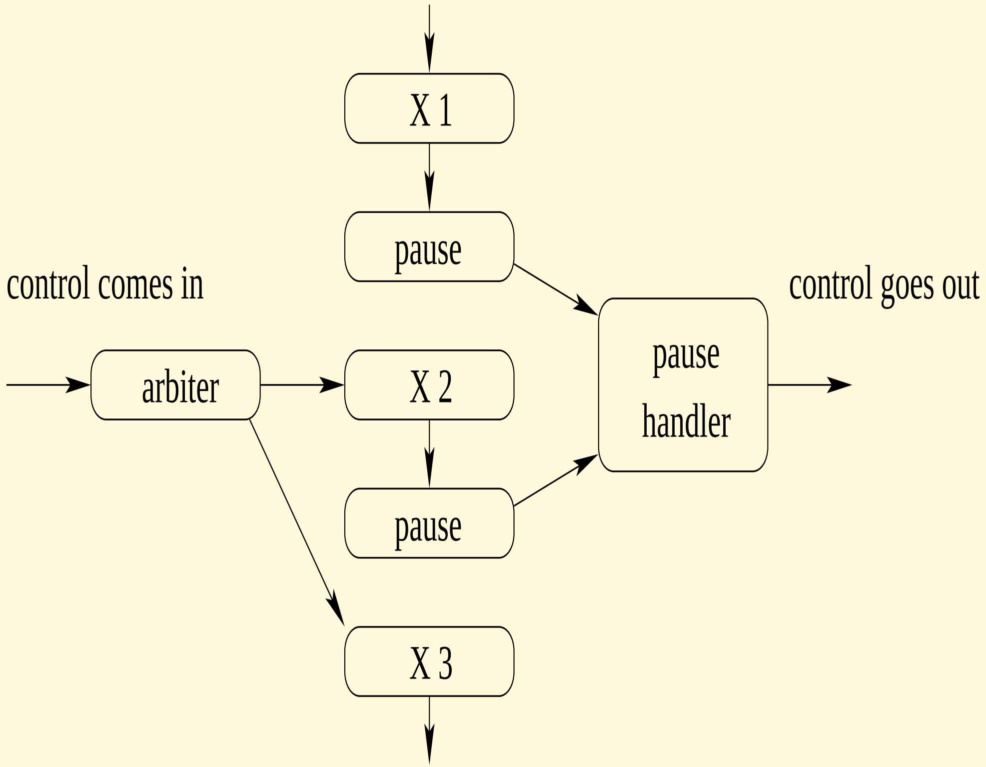
Full Screen

Close

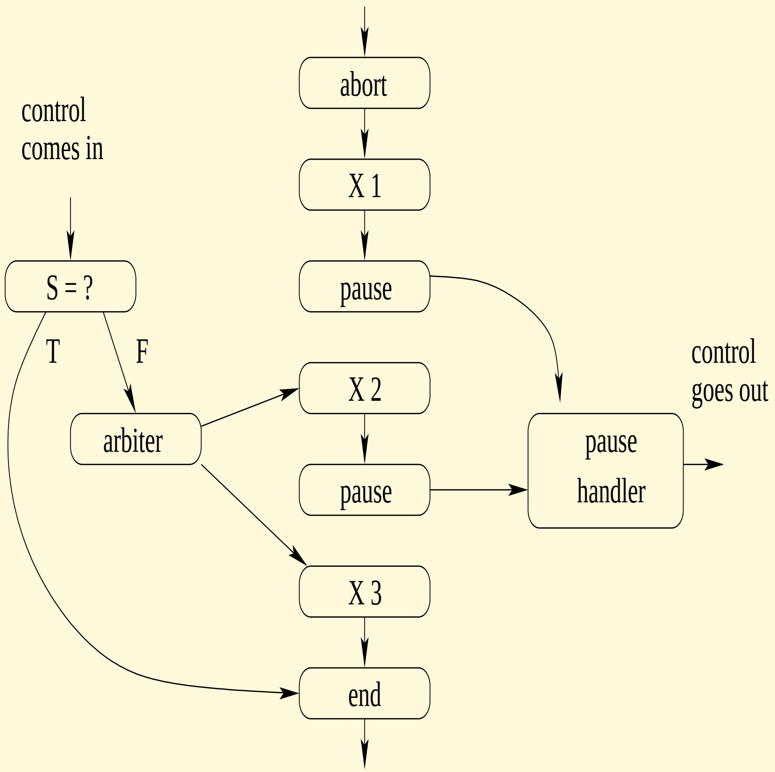
Quit

Pause

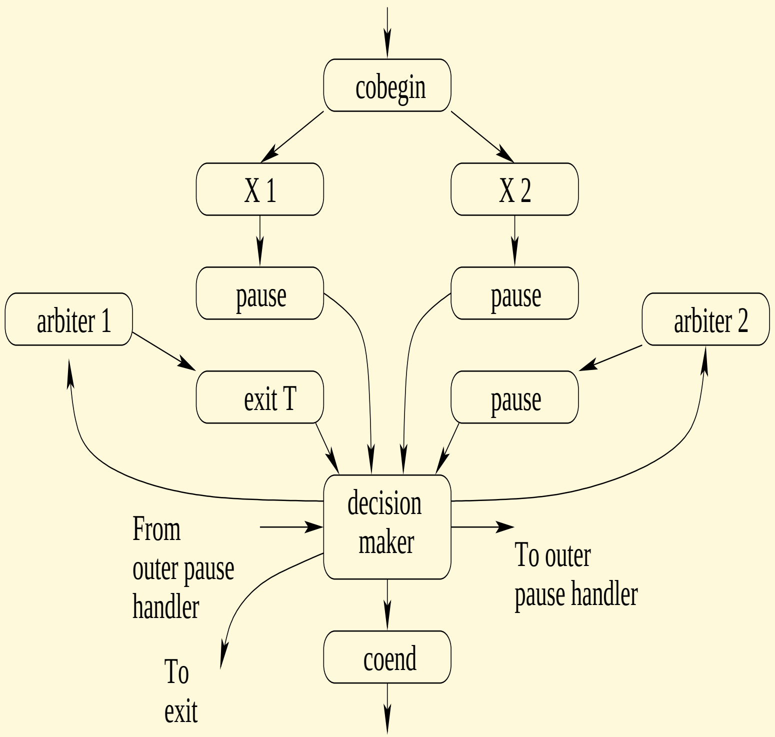
X 1 ;
pause ;
X 2 ;
pause ;
X 3 ;



Abort

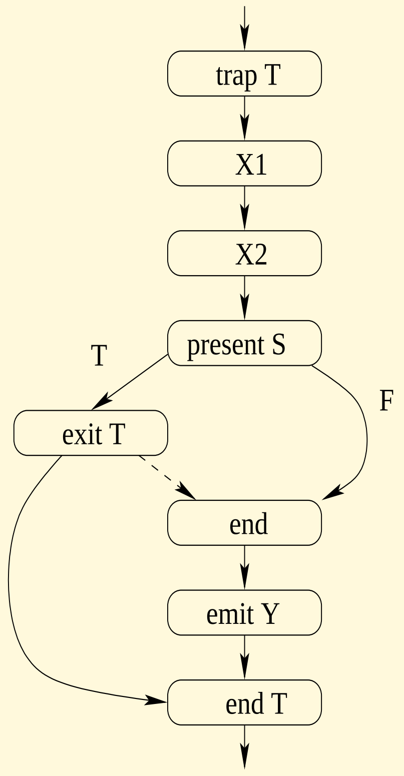

abort
X 1
pause
control
comes in
S = ?
T
F
arbiter
X 2
pause
control
goes out
pause
handler
X 3
end

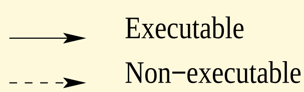
Parallel


cobegin
X 1
X 2
pause
pause
arbiter 1
arbiter 2
exit T
pause
decision maker
From outer pause handler
To outer pause handler
To exit point
coend

Exit

```
trap T
    X1 ;
    X2 ;
    present S then
        exit  T
    end ;
    emit Y ;
end
```

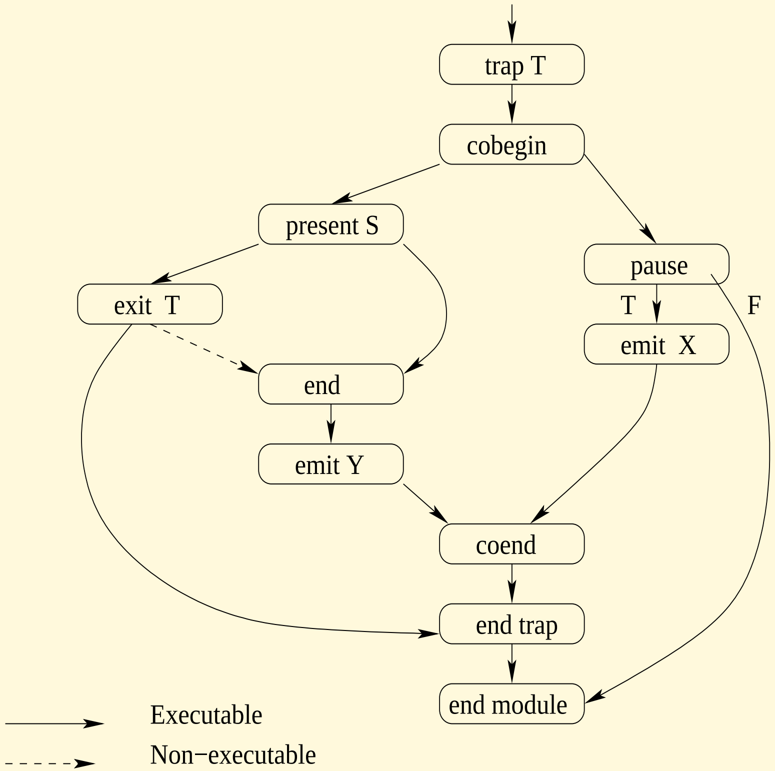




Interference Control Dependence

control dependence defn: j control dependent on i iff:

- j does not post dominate i
- $\forall\ k$ along path i to j, j post dominates k.



Dependencies in Esterel

- data dependencies: cannot exist across threads in Esterel.
- signal dependencies – three types:
  - simple: exist in non-concurrent threads
  - loop-carried: actually loop-carried, and cross thread boundaries
  - interference: in concurrent threads
- control dependencies – two types:
  - induced in non-concurrent threads
  - induced in concurrent threads, because of preemption
- time dependencies – captured as control dependencies.

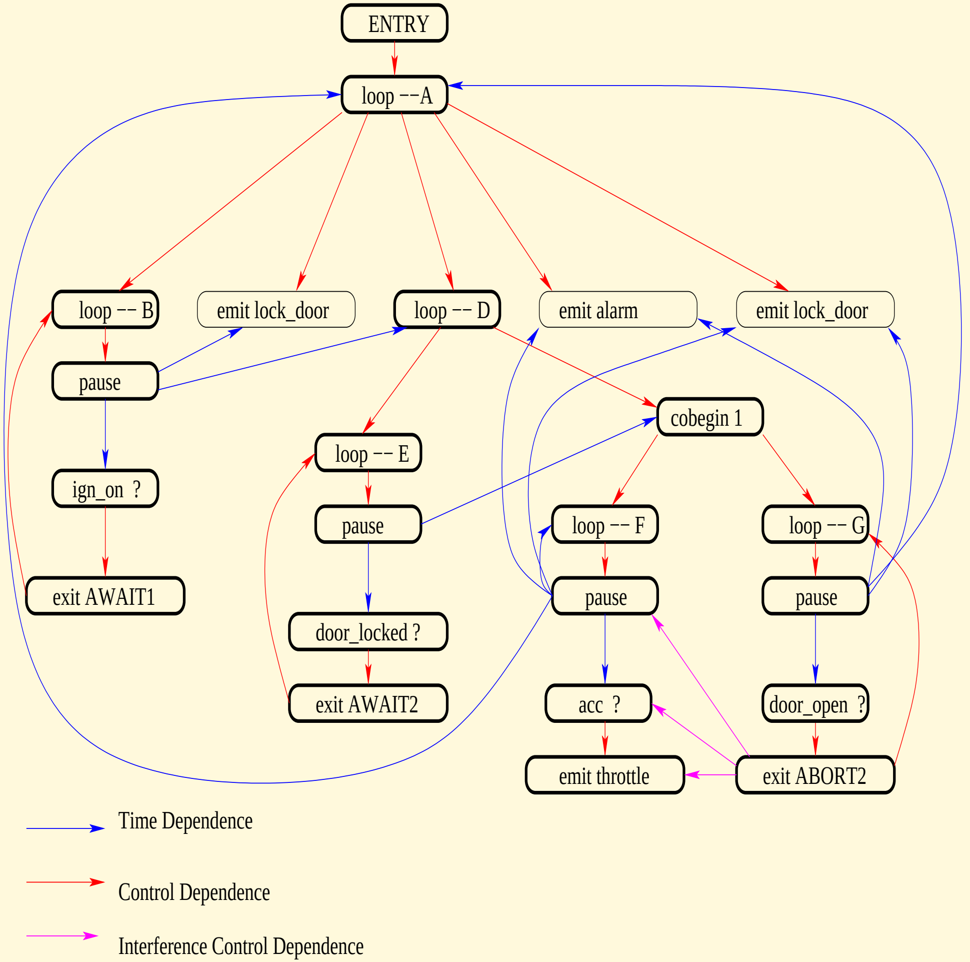
STPDG

- Synchronous Threaded Program Dependence Graphs
- Slicing involves traversal along this graph
- Notion of Synchronous Trace Witness
- A path in the graph that is a possible execution sequence in the program
- Slice definition:

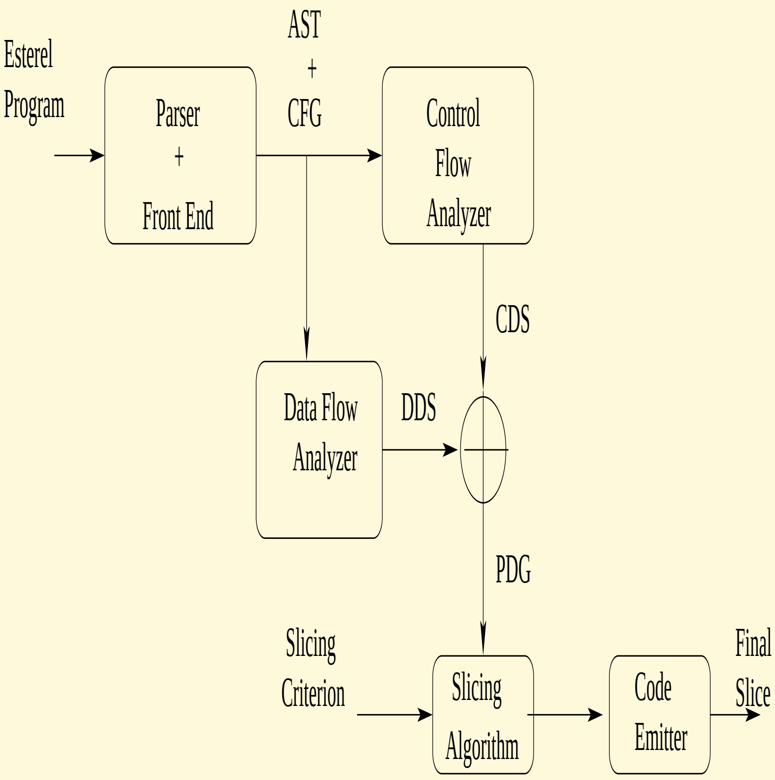
$$
\begin{aligned}
Slice(s) = \{\, q \mid \Gamma = \langle n_1, n_2, \cdots, n_k \rangle,\ & \\
& q = n_1 \longrightarrow^{d_1} \cdots \longrightarrow^{d_{k-1}} n_k = p, \\
& p \text{ is a `emit s' or `sustain s' node}, \\
& d_i \in E_{dd} \cup E_{cd} \cup E_{td} \cup E_{ssd} \cup \\
& \qquad E_{isd} \cup E_{icd}, 1 \leq i < k, \\
& \Gamma \text{ is a trace witness in } G \}
\end{aligned}
$$

Slicing Algorithm for Esterel

- slicing criterion = $\langle output\ signal\rangle$
- state of execution recorded in tuple $[t_0, t_1, \ldots, t_n]$
- performs a backward traversal of TPDG
- for nodes y reached via edge $y \longrightarrow x$:
  - data, control, simple signal dependence: add y to slice. update tuple.
  - interference signal or control dependence:
    - t = last node visited in y's thread
    - if trace witness exists for $\langle y, t\rangle$, add y. update tuple.
  - loop-carried signal dependence: add y to slice. update tuple.


ENTRY
loop --A
loop -- B
emit lock_door
loop -- D
emit alarm
emit lock_door
pause
cobegin 1
ign_on ?
loop -- E
pause
loop -- F
loop -- G
exit AWAIT1
pause
pause
door_locked ?
exit AWAIT2
acc ?
door_open ?
emit throttle
exit ABORT2
Time Dependence
Control Dependence
Interference Control Dependence

Implementation



Conclusions

- A New definition of slicing natural for reactive programs.
- Novel dependency graph representation
- A preliminary slicer for Esterel
- Same idea used for other reactive languages
- Slicers for Argos (Statecharts), VHDL