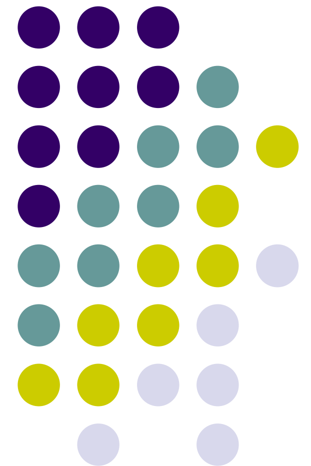
Clock Analysis of X10 Programs

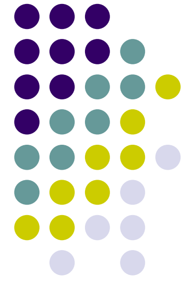
Nalini Vasudevan

Mentor: Olivier Tardieu

Manager: Vijay Saraswat



Background

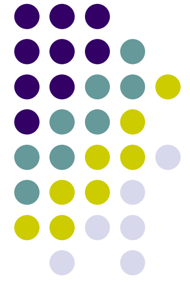


- X10 programming language
  - Parallel and Distributed
- Tasks/activities created using *async*

```
async {
/* body of async */
}
```

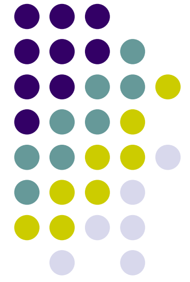
- Other constructs: *finish*, *atomic*
- **Focus:** Synchronization between activities through clocks

Clocks in X10



- Barriers
  - Declare clocks
  - Share clocks
  - *Synchronize*
    - *next()* function
      - All tasks clocked on c have to synchronize

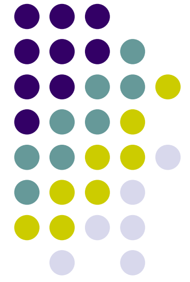
```
final clock c = clock.factory.clock();
async clocked (c) {
        ..
        c.next();
        ..
        c.next();
}
async clocked (c) {
        ..
        c.next();
        ..
        c.next();
}
```



More about Clocks

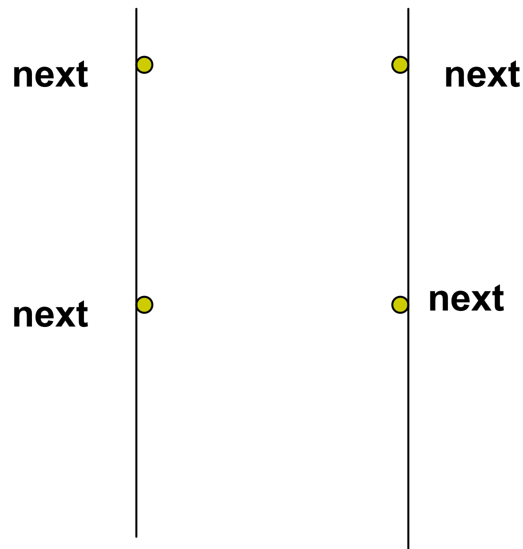
- *resume()*
  - Indicates the end of current phase

```
final clock c = clock.factory.clock();
async clocked (c) {
        ..
        c.next();
        ..
        c.next();
}
async clocked (c) {
        ..
        c.resume()
        ..
        c.next();
        ..
        c.next();
}
```



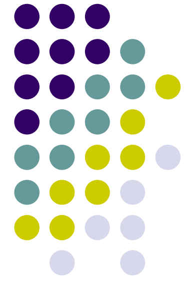
More about Clocks

- *resume()*
  - Indicates the end of current phase

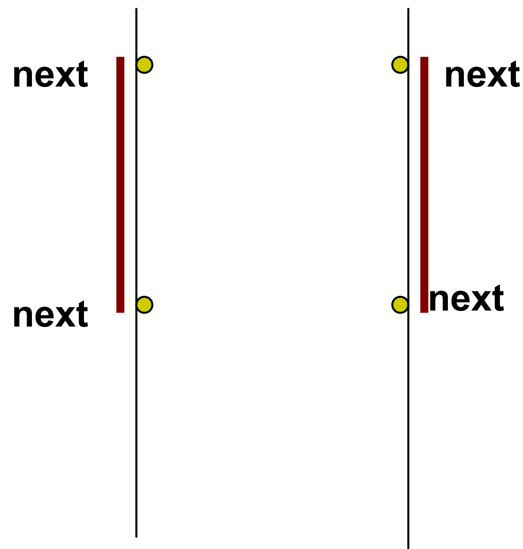



```
final clock c = clock.factory.clock();
async clocked (c) {
        ..
        c.next();
        ..
        c.next();
}
async clocked (c) {
        ..
        c.resume()
        ..
        c.next();
        ..
        c.next();
}
```

More about Clocks

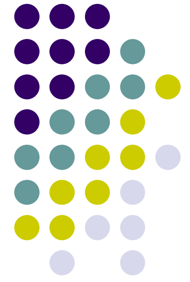


- *resume()*
  - Indicates the end of current phase

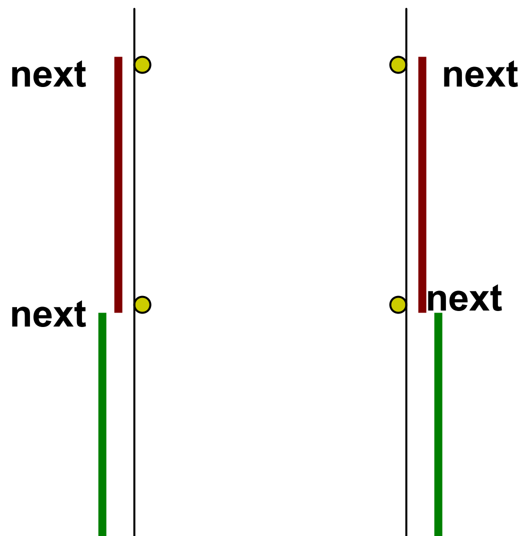



```
final clock c = clock.factory.clock();
async clocked (c) {
        ..
        c.next();
        ..
        c.next();
}
async clocked (c) {
        ..
        c.resume()
        ..
        c.next();
        ..
        c.next();
}
```

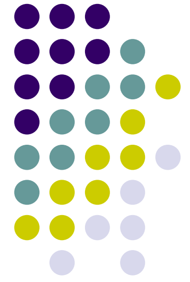
More about Clocks



- *resume()*
  - Indicates the end of current phase

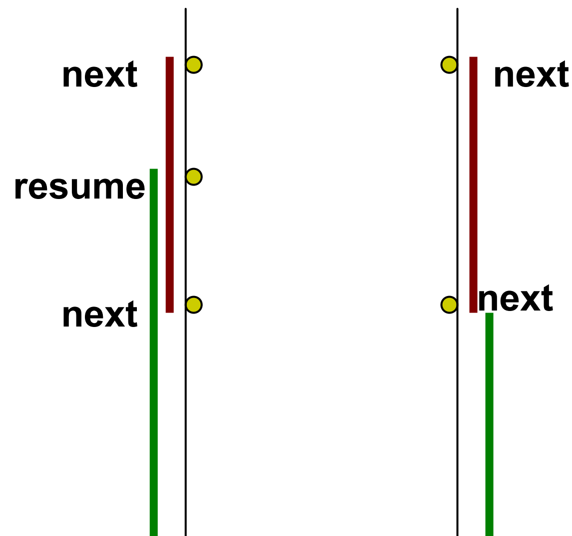



```
final clock c = clock.factory.clock();
async clocked (c) {
        ..
        c.next();
        ..
        c.next();
}
async clocked (c) {
        ..
        c.resume()
        ..
        c.next();
        ..
        c.next();
}
```



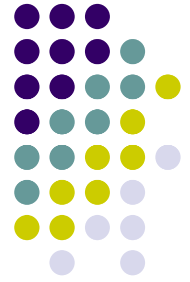
More about Clocks

- *resume()*
  - Indicates the end of current phase



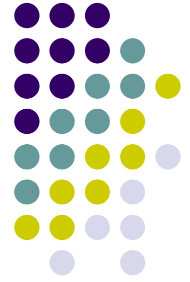

```
final clock c = clock.factory.clock();
async clocked (c) {
        ..
        c.next();
        ..
        c.next();
}
async clocked (c) {
        ..
        c.resume()
        ..
        c.next();
        ..
        c.next();
}
```

More about Clocks

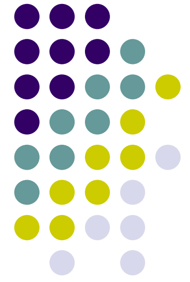


- *drop()*
  - Explicitly drop the clock.
  - Task does not have to synchronize anymore

The Protocol

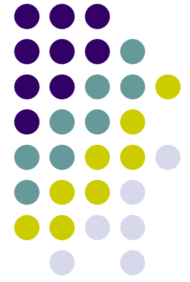


Motivation



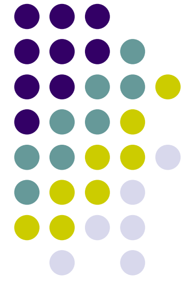
- Default implementation handles all cases
  - Handles protocol violation dynamically by throwing exceptions
    - *example: call next() after drop()*
- We can generate more efficient code if we know that
  - Activity does not violate the protocol?
  - Activity never calls *resume()* on clock c?
- We can also provide feedback to the user
  - Activity may violate the protocol…

Implementation Overview



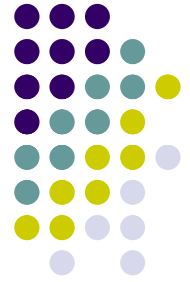
- Static Analysis
  - Use *wala* for intermediate representation and pointer analysis
  - Extract model from IR
    - One automaton per clock
  - Use NuSMV for model checking
- Code Specialization
  - X10 compiler plugin to choose clock implementation based on analysis result

Building the Automaton

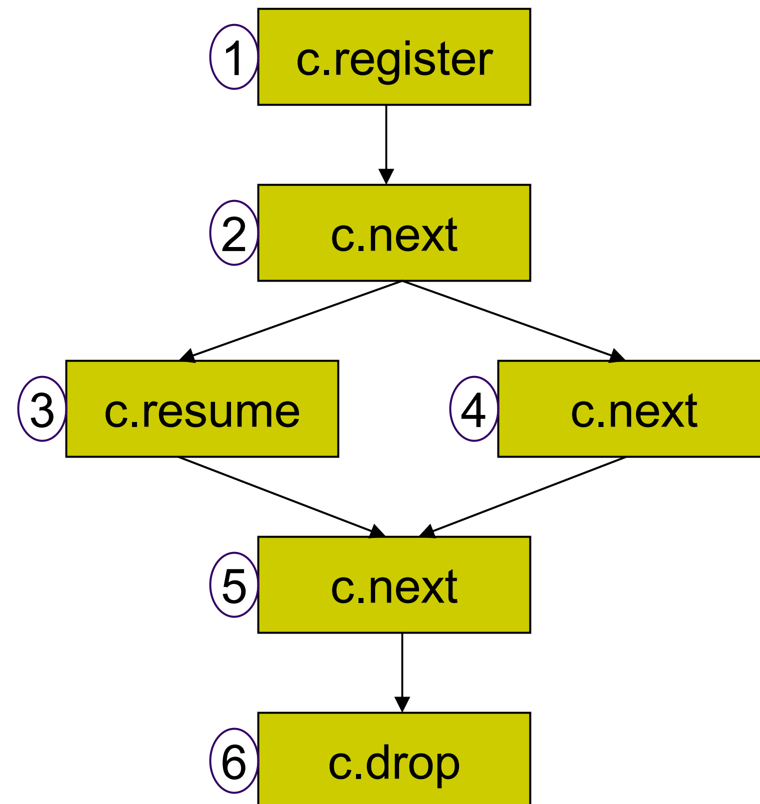


```
c = clock.factory.clock()
c.next()
 if (n > 1)
      c.resume()
else
      c. next();
 c. next();
 c.drop();
```

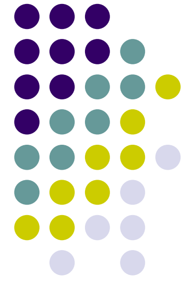
Building the Automaton

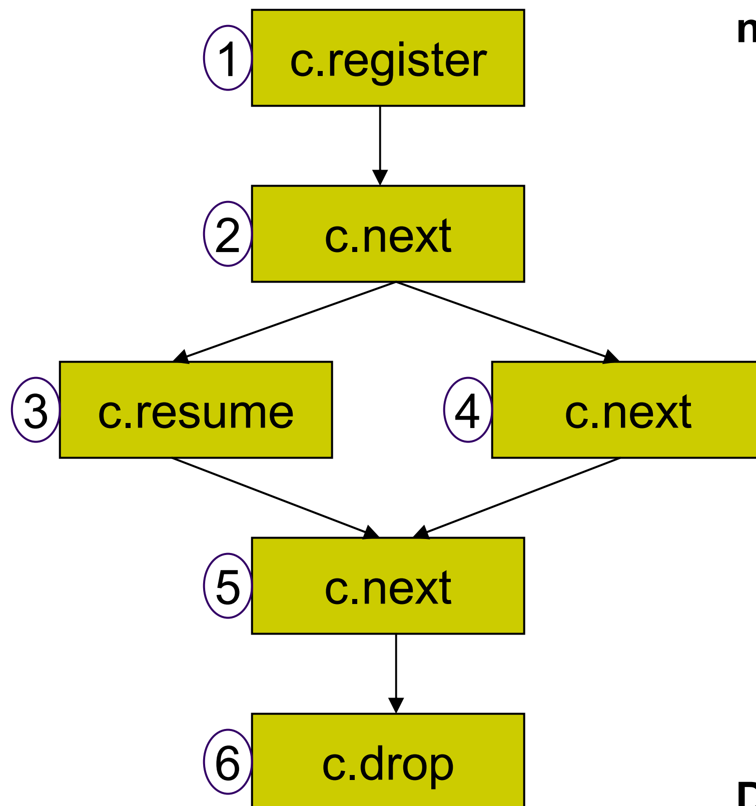


```
1     c = clock.factory.clock()
2     c.next()
       if (n > 1)
3          c.resume()
      else
4          c. next();
5      c. next();
6      c.drop();
```



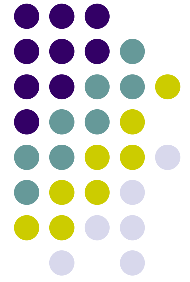
Building the Automaton





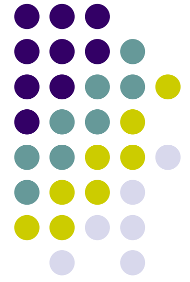

```
init (clock) = register;
next(clock) :=
    case
            (clock = register) : next_2;
            (clock = next_2) : {resume_3, next_4};
            (clock = resume_3) : next_5;
            (clock = next_4) : next_5;
            (clock = next_5) : drop_6;
            1: clock;
    esac;

DEFINE clock_next = clock in {next_2, next_4, next_5}
DEFINE clock_drop = clock in {drop_6}
```

State Automaton



```
init (status) = inactive;
next(status) :=
    case
        (status = inactive) & (clock_register) : active;
        (status = active) & (clock_drop) : inactive;
        (status = active) & (clock_resume): resumed;
        (status = resumed) & (clock_next): active;
        ..
 -- Exception cases
        (status = resumed) & (clock_resume):
resume_exception;
        (status = inactive) & (clock_next) : drop_exception;
        ..
```

Checking for properties



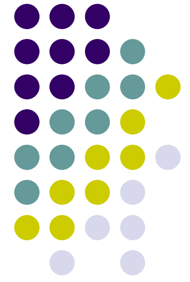
- Check if a clock is protocol violation free?

  DEFINE status_exception = status in {drop_exception, async_exception, …}

  SPEC AG(~(status_exception)

- Check if a clock never calls *resume()?*

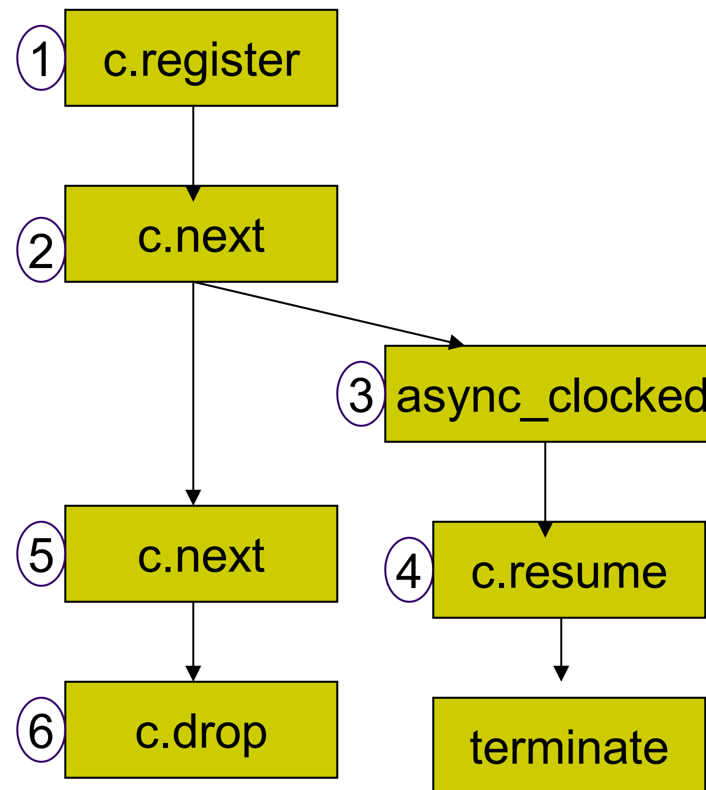
  SPEC AG(~(clock_resume))



Dealing with *asyncs*

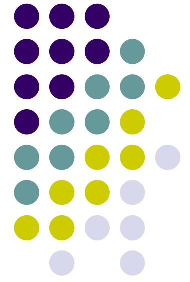
- Viewing *async* as a separate path

```
1    c = clock.factory.clock()
2    c.next();
3    async clocked (c) {
4         c.resume();
     }
5     c. next();
6     c.drop();
```



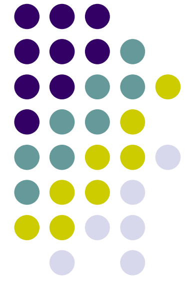

```
next(clock) :=
    case
        ..
        (clock = next_2): {async_clocked_3,
next_4};
```

Combining with Aliasing Analysis



```
clock c = clock.factory.clock ();

clock d = clock.factory.clock ();

clock x = (n>1)? c: d;

x.next();

x.resume();
```

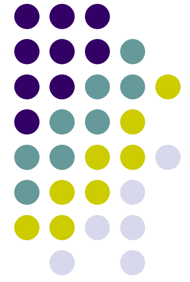
Combining with Aliasing Analysis



```
clock c = clock.factory.clock ();

clock d = clock.factory.clock ();

clock x = (n>1)? c: d;

x.next();

x.resume();
```

```
clock c = clock.factory.clock ();

clock d = clock.factory.clock ();

if(*) {
        c.next();

        c.resume();

}
else {
        d.next();

        d.resume();

}
```

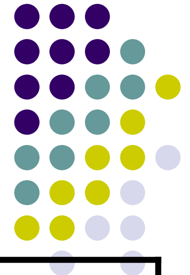
Combining with Aliasing Analysis



```
clock c = clock.factory.clock ();

clock d = clock.factory.clock ();

clock x = (n>1)? c: d;

x.next();

x.resume();
```

```
clock c = clock.factory.clock ();

clock d = clock.factory.clock ();

if(*)
                c.next();
 else
                d.next();

if(*)
                c.resume();
 else
                d.resume();
```

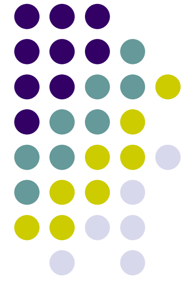
Results



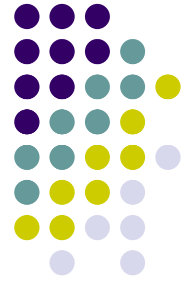
Example	Lines of Code	No. of Clocks	Analysis Time (sec)			Result	Speed Up
			IR and Automata construction	NuSMV	Total		
Kernel Algorithm	55	1	5.3	0.1	5.4	EF, NR	(38.4/40.4) -> 5.0%
All Reduction Barrier	65	1	21.7	0.1	21.8	EF, NR	(13.1/13.3) -> 1.5%
Sieve (Stream of Eratosthenes)	95	1	25.9	0.1	26.0	NR	(231.3/233.8) -> 1.1%
N-Queens	155	1	20.0	0.3	20.3	EF, NR, ON	(58.7/58.8) -> 0.2%
Sequence Alignment (Edmiston)	205	2	18.4	0.2	18.6	Clock 1: NR Clock 2: NR	(19.4/20.0) -> 3%
LU Decomposition	215	1	8.6	0.2	8.8	EF, NR	(25.4/26) -> 2.3%
Java Grande Forum Benchmark Suite	930	1	24.7	0.3	25.0	NR	(30.4/30.8) -> 1.3%

EF: Exception Free, **NR**: No Resume, **ON**: Only Clock Creator calls Next

Conclusion and Future Work



- Working tool!
- Sequential analysis for concurrency optimization
- Future Work
  - Inter-activity analysis
    - Deadlocks
  - Using clock information for refining aliasing analysis



Questions Guaranteed

Answers are not!