

# Modeling Systems in CLP

Joxan Jaffar, Andrew E. Santosa and Răzvan Voicu

School of Computing, National University of Singapore

{joxan, andrews, razvan}@comp.nus.edu.sg

# Outline of Presentation

A talk on general CLP approach for program verification

- Modeling
  - Representing transition relations in CLP
  - A non-standard timed automata
  - Microarchitecture
- Proof method based on coinductive tabling
- Ongoing works: liveness, abstraction

## 2-Process Bakery Algorithm

```
while (true) do  
⟨0⟩   t1 := t2 + 1  
⟨1⟩   await (t1 < t2 ∨ t2 = 0)  
⟨2⟩   t1 := 0  
end
```

```
while (true) do  
⟨0⟩   t2 := t1 + 1  
⟨1⟩   await (t2 < t1 ∨ t1 = 0)  
⟨2⟩   t2 := 0  
end
```

### CLP Representation (TOP-DOWN)

$\text{state}([1, P2], T1', T2) :- T1' = T2 + 1, \text{state}([0, P2], T1, T2).$

GOAL:  $\text{state}([1, 2], T1', T2'), \underline{T1' < T2'}$

REDUCT:  $\text{state}([0, 2], T1, T2), T1' < T2', T1' = T2 + 1, T2' = T2$

PROJECT:  $\text{state}([0, 2], T1, T2), \underline{T2 + 1 < T2}$

*A weakest precondition predicate transformation.*

## 2-Process Bakery Algorithm

```
while (true) do  
⟨0⟩   t1 := t2 + 1  
⟨1⟩   await (t1 < t2 ∨ t2 = 0)  
⟨2⟩   t1 := 0  
end
```

```
while (true) do  
⟨0⟩   t2 := t1 + 1  
⟨1⟩   await (t2 < t1 ∨ t1 = 0)  
⟨2⟩   t2 := 0  
end
```

### CLP Representation (TOP-DOWN)

state([0,0], 0, 0).

state([1,P2], T1', T2) :- T1' = T2 + 1, state([0,P2], T1, T2).  
state([2,P2], T1, T2) :- (T1 < T2 ∨ T2 = 0), state([1,P2], T1, T2).  
state([0,P2], T1', T2) :- T1' = 0, state([2,P2], T1, T2).

state([P1,1], T1, T2') :- T2' = T1 + 1, state([P1,0], T1, T2).  
state([P1,2], T1, T2) :- (T2 < T1 ∨ T1 = 0), state([P1,1], T1, T2).  
state([P1,0], T1, T2') :- T2' = 0, state([P1,2], T1, T2).

## 2-Process Bakery Algorithm

```
while (true) do  
⟨0⟩   t1 := t2 + 1  
⟨1⟩   await (t1 < t2 ∨ t2 = 0)  
⟨2⟩   t1 := 0  
end
```

```
while (true) do  
⟨0⟩   t2 := t1 + 1  
⟨1⟩   await (t2 < t1 ∨ t1 = 0)  
⟨2⟩   t2 := 0  
end
```

### CLP Representation (TOP-DOWN)

$\text{state}([1, P2], T1', T2) :- T1' = T2 + 1, \text{state}([0, P2], T1, T2).$

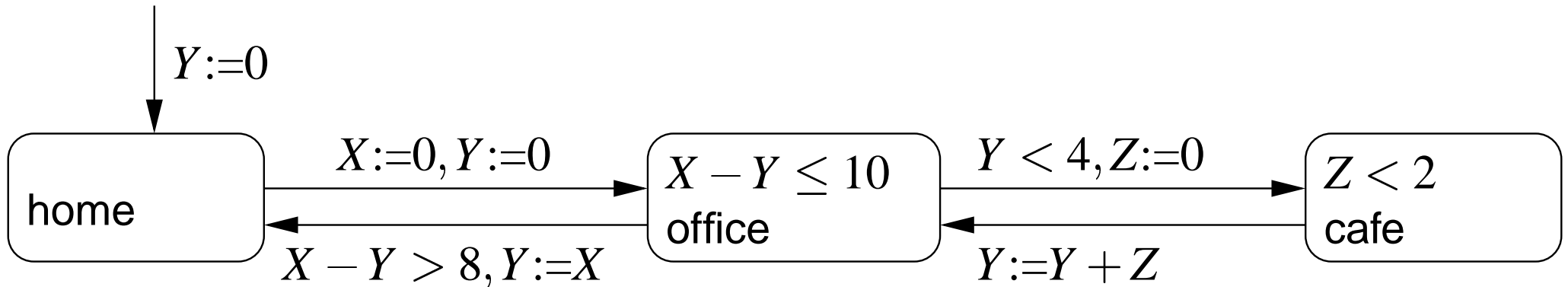
*A weakest precondition transformer.*

### CLP Representation (BOTTOM-UP)

$\text{state}([0, P2], T1, T2) :- T1' = T2 + 1, \text{state}([1, P2], T1', T2).$

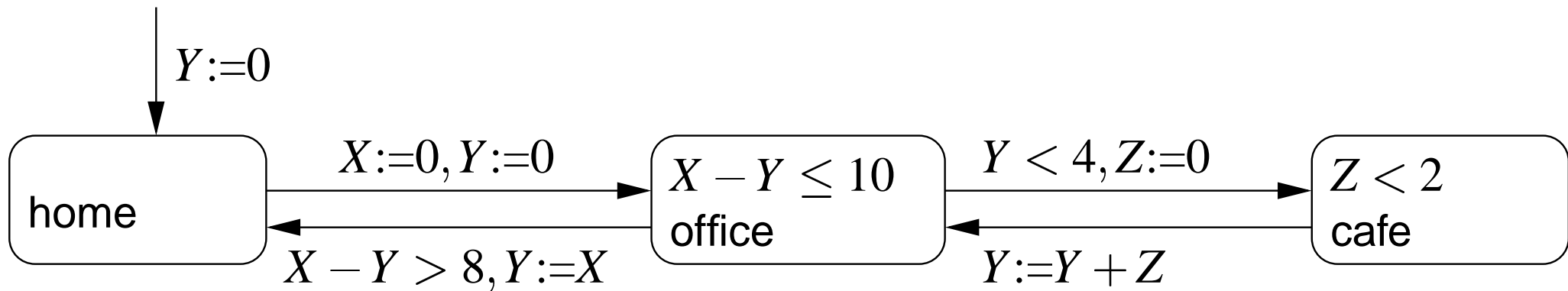
*A strongest postcondition transformer.*

# Non-Standard Timed Automata



- Clocks and normal variables are mixed in expressions
- Constraints with more than 2 clocks are possible

## Non-Standard Timed Automata



`state(home, X, Y, Z) :- E ≥ 0, X=E, Y=0, Z=E.`

`state(office, X1, Y1, Z1) :- E ≥ 0, X1=E, Y1=0, Z1=Z+E, X1-Y ≤ 10,`  
`state(home, _, Y, Z).`

`state(cafe, X1, Y1, Z1) :- E ≥ 0, X1=X+E, Y1=Y, Z1=E, Y < 4, X-Y ≤ 10, Z1 < 2,`  
`state(office, X, Y, _).`

`state(office, X1, Y1, Z1) :- E ≥ 0, X1=X+E, Y1=Y+Z, Z1=Z+E, Z < 2, X1-Y1 ≤ 10,`  
`state(cafe, X, Y, Z).`

`state(home, X1, Y1, Z1) :- E ≥ 0, X1=X+E, Y1=X, Z1=Z+E, 8 < X-Y, X-Y ≤ 10,`  
`state(office, X, Y, Z).`

# Microarchitecture

```
⟨0⟩ j := 1
⟨1⟩ while (j < 3) do
⟨2⟩   if (a[j] > a[j+1]) then ⟨3⟩ swap (a[j], a[j+1])
⟨4⟩   j := j + 1
  end ⟨5⟩
```

- Fixed assignment of cache line to instructions (direct-mapped cache).
- 2 cache lines, each contains at most 2 instructions
- Instructions ⟨0⟩, ⟨2⟩, ⟨4⟩ mapped to line 0  
Instructions ⟨1⟩, ⟨3⟩, to line 1
- Cache hit costs 1 t.u., miss 5 t.u.

# Microarchitecture

```
state(0,A,K,J,T,Tt) :- K = [[],[ ]], update(0,K,K1,E), state(1,A,K1,1,T+E,Tt).
state(1,A,K,J,T,Tt) :- J<3, update(1,K,K1,E), state(2,A,K1,J,T+E,Tt).
state(1,A,K,J,T,Tt) :- J≥3, update(1,K,K1,E), state(5,A,K1,J,T+E,Tt).
state(2,A,K,J,T,Tt) :- A[J]>A[J+1], update(2,K,K1,E), state(3,A1,K1,J,T+E,Tt).
state(2,A,K,J,T,Tt) :- A[J]≤A[J+1], update(2,K,K1,E), state(4,A,K1,J,T+E,Tt).
state(3,A,K,J,T,Tt) :- swap(A,J,J+1,A1), update(3,K,K1,E), state(4,A,K1,J,T+E,Tt).
state(4,A,K,J,T,Tt) :- update(4,K,K1,E), state(1,A,K1,J+1,T+E,Tt).
state(5,A,K,J,T,T).
```

```
update(Instr,[CL0,CL1],[CL0,CL1],1) :- in(Instr,CL0),!.
update(Instr,[CL0,CL1],[CL0,CL1],1) :- in(Instr,CL1),!.
update(Instr,[CL0,CL1],[CL01,CL1],5) :-
    cl_assgn(Instr,0), update_line(CL0,Instr,CL01).
update(Instr,[CL0,CL1],[CL0,CL11],5) :-
    cl_assgn(Instr,1), update_line(CL1,Instr,CL11).
```

```
update_line([],Instr,[Instr]). % cache empty
update_line([H1],Instr,[H1,Instr]). % partial
update_line([_,H2],Instr,[H2,Instr]). % cache full
```

# Coinductive Tabling

- Based on coinduction
- Used for storing proof obligations
- Not dealing with redundant calls

# Safety Assertions

$G \models \Psi$ , where  $G$  is a goal and  $\Psi$  is a constraint

## Examples:

- Bakery mutual exclusion:

$$\text{state}([P1, P2], T1, T2) \models P1 \neq 2, P2 \neq 2$$

- Timed automata:  $\text{state}(\text{home}, X, Y, Z) \models Y \leq 20$

- Microarchitecture: WCET

$$\text{state}(5, A, K, J, T, Tt) \models Tt \leq 30$$



# Ongoing Work

- **Liveness:** A proof method based on well-founded induction for verifying liveness of infinite-state systems
- **Abstraction:** An abstraction-based proof framework that is **intermittent** and **compositional**  
Intermittence: abstract only at certain points  
Requires CLP projection mechanism

## Some Related Work

J. Jaffar, A. E. Santosa, R. Voicu, *A CLP Proof Method for Timed Automata*, RTSS '04

G. Delzanno and A. Podelski, *Model Checking in CLP*, TACAS '99

Y. S. Ramakrishna *et al.*, *Efficient Model Checking Using Tabled Resolution*, CAV '97

*And many others ...*