Parallel Automata and their implementation

(workshop “semantics/verif. of hard/soft systems”, may 20-22, 2003, Tel-Aviv)

H.G. Mendelbaum\(^1,3\), R.B. Yehezkael\(^1\) (formerly Haskell), T. Hirst\(^1\), A. Teitelbaum\(^1\), S. Bloch\(^2\)

\(^1\)Jerusalem College of Technology – Jerusalem – Israel
\(^2\)Univ. Reims, RESYCOM – Reims - France
\(^3\)Univ. Paris V- René Descartes, IUT – Paris - France

Emails: \{mendel, rafi\} @ jct.ac.il
simon.bloch @ univ-reims.fr
Example of DesCARTES design (washing Machine) (with Teitelbaum - JCT)

const MinWaterLevelForHeating = 1.5
const DrumCapacity = 5.0

Drum
Thermometer

Reactive Kernel

WaterQuant
WaterTemp

TempRequired
Program

Behavior

Filling: Regulator
Thermostat: Regulator
Pump

TapOn
HeaterOn
PumpOn
Process

Reactive Kernel

Tap
Heater
Pump

TapLED
HeaterLED
PumpLED

Internal LEDs

MotorOn
Direction
Speed

Motor
TCOM : Temporal Components
(with Vidal-Naquet, SupElec)

- Var, signals, flags, clocks
- Operations: I/O, functions
- Behavior (Manager): APA ∨ PTL

- Goals: PTL
- Assertion Guards: APA ∨ PTL
TCOM : PTL engineering dialect for validation
(with Vidal-Naquet, SupElec)

GOALS/GUARDS in TCOM dialect for the Fill-regulator component

when (Prog4 and Start) then

(Eventually (Full and tap_closed)
  imply Fill_Goal ));

Translation in PTL

(Prog4 \and Start \implies <> (Full \and closed \Rightarrow Fill_Goal ))
Validation by Prover and Description of Behavior (with Vidal-Naquet, SupElec)

Goal Validation (e.g. ‘STEP’ Prover)

( Fill_Goal ∧ Thermo_Goal ∧ Pump_Goal ∧ Motor_goal ∧ Main_Behavior)

==> Main_Goal
How to execute the behavior element

Direct PTL execution (as Fisher-UK)

Or

Execution through a bridge-model between “Extended FSM” and PTL
Parallelism, explosion of states and Mealy model

• For the representation of R-T control, we need:
  – Parallel tests on variables, timers or Parallel receiving of events or inputs
  – Run Parallel actions and outputs
• We need to differentiate the same elements in various situations,
• this leads to an explosion of number of states
Main trends in Extended FSM for component behavior execution

- Adding Conditions to the global state
  \[ E_m, s_k, c_n, !c_p \ldots \rightarrow a_i, s_r, !c_q, c_t \]

- Adding variables to the global state
  \[ E_m, s_k, d_n \ldots \rightarrow a_i, s_r, d_t \ldots \]

- Adding clocks to the global state
  \[ E_q, s_p, \text{cond}_n(c_a) \rightarrow a_k, s_j, \text{reset}(c_{a_i}) \]
Extended FSM : (2)
for component behavior execution

• Automata with multiple states
  \[ E_i, \{n_2, n_5 \ldots\} \rightarrow \{n_4, n_6 \ldots\} \]

• Automata with several events and multiple actions (I/O automata) and global state
  \[ E_r, E_h, st_j \rightarrow act_i, out_k, st_k \]
Seeking a general bridging-model "TCOM-PTL" – "Extended-FSM"

- **In PTL**: no specific "states"
- **In EFSM**: various extensions to states: conditions on var, time, ...
  with specific semantic meaning
- **formally**: states, events, inputs, flags, timers, actions, outputs, ...
  can be variables to test or to assign
  \[ \text{Cond}_n (V) \Rightarrow \text{Assign}_i (V, N) \]
Representation of time control (discrete or continuous time?)

- The controller needs only to know dates and interval of times.
- It needs only to read (input) or reset time variables (timers).
- These time variables are updated by an external mechanism which does not interest the controller, so the notion of time can be viewed as an abstract notion external to control.
APA: Abstract Parallel Automata
for behavior execution

• Abstract forms

conjunction form

\[ \pi_j \mid \text{cond}_j \mid \rightarrow \pi_k \mid \text{assign}_k \]

General form

compound cond \rightarrow \pi_k \mid \text{assign}_k \\

\textbf{cond}_j \textbf{ are boolean relation tests : }
- event, input signal or flag arrived
- state or variable or time condition

\textbf{assign}_k \textbf{ are setting of values to variables }: 
- setting a state, a variable or a clock
- execution of actions or functions
- output of flags, Sending of Events
More formal definition of APA:
Abstract Parallel Automata
(with Yehezkael, T. Hirst – JCT)

APA = (I, V, O, K, R)

I = \{x_1, \ldots, x_r\} \quad \text{a finite set of input variables.}

V = \{y_1, \ldots, y_s\} \quad \text{a finite set of internal variables.}

O = \{z_1, \ldots, z_t\} \quad \text{a finite set of output variables.}

K = \text{the finite range of values of each variable}

R = \text{a finite number of rules:}

\begin{align*}
\text{test the values of variables} & \quad \rightarrow \\
\text{make assignments to variables}
\end{align*}
The execution of automata rules takes place in a succession of cycles:

- In each cycle, each automata rule is scanned once only.
- During each cycle, variables are tested and assigned.
- The new value of a variable
  \[ I \cup V \cup O \rightarrow \{0,1,2,\ldots,K\} \]
  becomes available in the following cycle.
Example of Behavior (in APA): double parallelism: horizontal / vertical (with Bloch – Reims)

[1] /prog4=1/ /start=1/ /full=0/ → /fill:=1/ /LED:=2/

   /motor:=1//timer0:=0/


[4] /motor=1//timer0=30/ → /thermo:=0//pump:=1/
   /motor:=0/ /LED :=4/


Architecture of a APA operating system (with Teitelbaum - JCT)

- **Clock**: \( \Delta t_c \)
- **Condition functions**
- **Events and condition flags**
- **Automata-table**
  - **Entries**
    - APA 1
    - APA 2
    - APA 3
  - **Reactions**
- **Scan engine**: \( \Delta t_s \) < \( \Delta t_c \)
- **Inputs**
- **Functions**
- **Actions**
- **Outputs**
Problems of WW conflicts in APA

- (i) A **strong** conflict occurs when two or more assignments of **different** values are made pseudo-simultaneously to the same location.
- (ii) A **weak** conflict occurs when two or more assignments of the **same** value are made pseudo-simultaneously to the same location.

These conflicts can be **ACTUAL** or **POTENTIAL**
A-priori detection of (actual or potential) conflicts
(with Yehezkael, T. Hirst – JCT)

- A-priori Actual conflicts detection for general parallel automata is P-space complete
- And Potential conflicts detection is NP complete
- but for simple conjunctions, Potential conflict detection is possible in polynomial size
Detecting potential conflicts is possible in polynomial time when all conditions are conjunctions (Yehezkael, T. Hirst, JCT)

```plaintext
potential_conflict:=false;
for every pair of rules
    loop
        if there exist values for making the left hand sides of the pair of rules true and the same variable is assigned (different values) on the right hand sides of the pair of rules
            then potential_conflict:=true; exit for loop;
        end if;
    end for;

• Checking pairs of conditions contributes a quadratic term to the complexity, and then we need to determine whether the conjunction of a pair of conditions is satisfiable.
```
Satisfiability (Yehezkael, T. Hirst, JCT)

- As all the conditions are conjunctions of primitive conditions, then so too is the conjunction of pairs of such conditions. Fortunately, determining the satisfiability of a conjunction of primitive conditions is easily done in polynomial time as follows.

```plaintext
for each variable in the condition
loop Form the intersection of the ranges of values this variable takes.
end for;
if all these intersections of are not empty
then the condition is satisfiable
else the condition is unsatisfiable
end if;
```
A-priori detection of conflicts
Other results (2)
(Yehezkael, T. Hirst, JCT )

• any parallel automaton can be converted in polynomial time into a nearly equivalent automaton which detects potential conflicts, with size proportional to the size of the original automaton.

• The new automaton is equivalent to the original automaton, when the original automaton is free of conflicts.
Conversion to a form which detects potential conflicts
(Yehezkael, T. Hirst, JCT)

(a) Ensure that there is only one assignment on the right hand side of a rule:

A rule such as "Condition" → /a:=1//b:=2//c:=3/
would be replaced by the following three equivalent rules.

"Condition" → /a:=1/
"Condition" → /b:=2/
"Condition" → /c:=3/
(b) **Group rules:** For each variable there is a group consisting of all the rules which assign to it. Between groups there is not even potential conflict. A conflict may only occur in a group.

(c) **Ensure no potential conflict in a group:**

Consider a typical group which assigns on the variable $x$.

"Cond. 1" $\rightarrow /x:=1/
"Cond. 2" \rightarrow /x:=2/
"Cond. 3" \rightarrow /x:=2/

This would be replaced by the detecting rule:

"Cond. 1" and ("Cond. 2" or "Cond. 3") $\rightarrow /!conflict:_x:=1/

**NOTE:** Weak conflict not treated as an error in the above.
conflicts A-priori detection of
Other results (3)
(Yehezkael, T. Hirst, JCT )

• conversion of a general parallel automaton into a form where all conditions are conjunctions of primitive conditions is possible in polynomial size, providing that computation rules are given priority over I/O rules.
A-posteriori on-line detection of conflicts (with Teitelbaum-JCT)

- Instead of changing the automaton, the execution mechanism can test for conflicts \textit{a-posteriori at run time}, and not perform the assignments involved in conflicts.
- The \textit{a-priori} converted automaton has no conflicts because its rules test for assignments causing conflicts at run time, and so would require more memory for storing the modified automaton rules.
- The \textit{a-posteriori} approach requires that the execution mechanism tests for conflicts at run time, which would make the execution mechanism more complex.