ESSP: An Embedded Software Synthesis and Prototyping Methodology

Trong-Yen Lee (李宗演)
National Taipei University of Technology, R.O.C.
Date: 12/16/2002

Outline

- Introduction
- Petri Net
- Scheduling Concept
- Embedded Software Synthesis and Prototyping Methodology
- Embedded System Example
- Conclusion
Outline

- Introduction
- Petri Net
- Scheduling Concept
- Embedded Software Synthesis and Prototyping Methodology
- Embedded System Example
- Conclusion

Introduction

An embedded system is one that is installed in a large system with a dedicated functionality.

In general, embedded systems have a microprocessor for executing software and some hardware in the form of ASICs, DSP, and I/O peripherals.
Software synthesis is a process in which a formally modeled system can be synthesized by a scheduling algorithm into a set of feasible schedules that satisfy all user-given constraints on system functions and memory space.

We propose and use a high-level variant of the model called *Complex-Choice Petri Nets* (CCPN).

---

Software verification formally analyzes the behavior of synthesized software to check if it satisfies all user-given constraints on function and memory space.

Finally, the generated embedded software is placed into an emulation platform for prototyping and debugging.
Introduction

The proposed ESSP (Embedded Software Synthesis and Prototyping) methodology will be illustrated using two examples: a Vehicle Parking Management System (VPMS) and a motor speed control system.

Outline

- Introduction
- Petri Net
- Scheduling Concept
- Embedded Software Synthesis and Prototyping Methodology
- Embedded System Example
- Conclusion
Petri Net

簡介:
- 理論起源Carl Adam Petri 於1962年提出
- 並將此觀念建構在Asynchronous和Concurrency的理論上
- 圖形與數學化的模組工具
- 結構特性(Structural properties): P, T
  行為特性(Behavioral properties): Token

定義與符號和運作規則

#Petri net 的組成元件:
定義：

\[ PN = (P, T, F, W, M_0) \]

- \( P = \{p_1, p_2, \ldots, p_n\} \) 為所有位置(Place)所構成的有限集合, 以 \(|P| = n\) 表示.
- \( T = \{t_1, t_2, \ldots, t_m\} \) 為所有轉移(Transition)所構成的有限集合, 以 \(|T| = m\) 表示, 且 \( P \cup T \neq 0 \) 和 \( P \cap T = 0 \).
- \( F = \{f_{input}, f_{output}\} \) 為所有有向箭頭(Arcs)的集合, 表示所有位置及轉移動作之間的關係, 分為輸入箭頭與輸出箭頭.

\[ f_{input} \in (P \times T) \]
\[ f_{output} \in (T \times P) \]

- \( W \) 是一個加權(weight)函數 \((W:F \to N)\).
  - \( W(p, t) \) 表示由 Place 到 Transition 的 Arc 的權重值.
  - \( W(t, p) \) 表示由 Transition 到 Place 的 Arc 的權重值.
  - 所有的權重值至少為“1”.
- \( M_0:P \to \{1, 2, 3, \ldots\} \) 為系統的初始狀態(Initial Marking), \( M_k = (m_1, m_2, \ldots, m_n) \), 被稱為系統的狀態.
  - 狀態值(Marking) \( m_i \) 即為位置 \( p_i \) 於狀態 \( k \) 時所含的 Token 個數.
#運作規則：

Enabled : $t_3, t_4$
Not Enable : $t_2, t_4$

If $t_j$ fire

$M_0 = (\emptyset, 0, 0, 2, 1)$

1. If $p_i$ Then $p_j$

2. If $p_{i1}$ and $p_{i2}$... and $p_{in}$ Then $p_j$

3. If $p_i$ Then $p_{j1}$ and $p_{j2}$... and $p_{jn}$

#類型與實例

#類型：

1. If $p_i$ Then $p_j$

2. If $p_{i1}$ and $p_{i2}$... and $p_{in}$ Then $p_j$

3. If $p_i$ Then $p_{j1}$ and $p_{j2}$... and $p_{jn}$
4. If \( p_{i1} \) or \( p_{i2} \)… or \( p_{in} \) Then \( p_j \)

5. If \( p_i \) Then \( p_{j1}\) or \( p_{j2} \)… or \( p_{jn} \)

#實例:

**Program**

```plaintext
begin
input(y1);
input(y2);
y3:=1;
while y1 > 0 do begin
  if odd(y1) then begin
    y3:=y3*y2;
y1:=y1-1;
  end;
y2:=y2*y2;
y1:=y1-2;
end;
output(y3);
end;
```

**State**

<table>
<thead>
<tr>
<th>Action</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>Input(y1);Input(y2); y3:=1;</td>
</tr>
<tr>
<td>b</td>
<td>y3&gt;0 ?</td>
</tr>
<tr>
<td>c</td>
<td>odd(y1)</td>
</tr>
<tr>
<td>d</td>
<td>y3:=y3*y2; y1:=y1-1;</td>
</tr>
<tr>
<td>e</td>
<td>y2:=y2*y2; y1:=y1-2;</td>
</tr>
<tr>
<td>f</td>
<td>output(y3)</td>
</tr>
</tbody>
</table>
Flowchart

Translating

Computation

Decision

Translation
begin
    input(y_1);
    input(y_2);
    y_3:=1;
    while y_1>0 do begin
        if odd(y_1) then begin
            y_3:=y_3*y_2;
            y_1:=y_1-1;
        end;
        y_2:=y_2*y_2;
        y_1:=y_1-2;
    end;
    output (y_3);
end;
分析系統行為的方法

以下幾種常見的屬性可以透過Petri Net進行分析:

1. 限制性(boundedness): 每個Place中的標記數量是有限制的。
2. 安全性(Safeness): 每個Place中的權杖會維持在一定的數量下，而不能無限的擴增。
3. 鎖死與活性(Deadlock and Liveness): 鎖死現象是指系統到一狀態後無法到達其他狀態，反之，一系統無此現象就稱為活性的。

方法一: 可到達樹(Reachability Tree)

可到達樹的建構是從一個初始狀態開始擴展。演算法為初始狀態開始，進入一個重複迴圈的程序以進行狀態擴展。其步驟為:

1. 產生下一狀態(next state)
2. 確認下一狀態是否為新的狀態(new state)。
   若為舊狀態(Old state)則將其捨棄，如此可避免重複的狀態產生。
3. 若是新的狀態才將此新的狀態插入可到達樹(reachability tree)中。
可到達樹的實例:

方法二: 系統矩陣
首先定義出一個系統矩陣，並利用此系統矩陣來表示位置與轉換間的流向關係，進而根據Embl的數學公式推算經過某個轉換之後的狀態。

\[ \mu' = \mu + \chi D \]

- \( \mu' \) : 代表下一個狀態
- \( \mu \) : 代表現在的狀態
- \( \chi \) : 代表兩狀態中間觸發轉換
系統矩陣的實例

\[ D^+ \] 表示系統輸出矩陣

\[ D^+ = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \]

\[ D^- \] 表示系統輸入矩陣

\[ D^- = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 1 & 1 & 0 & 0 \end{bmatrix} \]

系統矩陣 \( D \)

\[ D = D^+ - D^- = \begin{bmatrix} -1 & 0 & 1 & 1 & 0 & 0 \\ 0 & -2 & -1 & -1 & 1 & 1 \end{bmatrix} \]

\[ \mu' = \mu + \chi D \]

\[ = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} -1 & 0 & 1 & 1 & 0 & 0 \\ 0 & -2 & -1 & -1 & 1 & 1 \end{bmatrix} \]

\[ = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix} \]

\[ p_3 \quad p_4 \]
Outline

- Introduction
- Petri Net
- **Scheduling Concept**
- Embedded Software Synthesis and Prototyping Methodology
- Embedded System Example
- Conclusion

Scheduling Concept

- Embedded system specification
  - Control structures
  - Data computations
    - Data dependent controls:
      - If-then-else
      - While do loop
    - Real time controls:
      - Preemption
      - Suspension
      - Trigger action
- **Static scheduling (Static scheduling):**
  - Only contains data computation.

- **Quasi static scheduling (Quasi static scheduling):**
  - Data dependent control issues.
  - Bounded Memory.

- **Dynamic scheduling (Dynamic scheduling):**
  - Real time issues.
  - Preemption
  - Suspension
  - Trigger action

---

- **Free-Choice Petri Net:**
  - To resolve Bounded Memory and Data dependent control issues, for example, if-then-else or while-do-loop conflicts. The system waits for a value from a certain state to decide the next action.

- **Conflict Free Net:** A PN such that each place has at most one output transition.

- **Free Choice Net:** A PN such that every arc from a place has a unique outgoing arc or a unique incoming arc to a transition.

- **Schedulable:** Decompose the net into Conflict(CF) component and every CF component is static schedulable.
Cycle schedules:

\[ f(\sigma) = (4, 2, 1)^T \]

\[ (0, 0)^{\sigma=t_1t_3t_4t_5}\rightarrow(0, 0)^{\sigma=t_1t_3t_4t_5}\rightarrow(0, 0)\cdots \]

T-invariant:

\[ f(\sigma)^T \cdot D = 0 \]

Example:

Free-Choice Net

Not Free-Choice Net

Schedulable

Not Schedulable
# How to find a valid schedule

FCPN $\xrightarrow{\text{NET decomposition}}$ Conflict-Free Component

Deadlock
Free

Finite
Complete cycle

Quasi-static data scheduled

CF-Components

---

**Time Free-Choice Petri Net:**

In transition's expression, add time restriction. Added the immediate restriction, this restriction must be completed within 1-4 units of time.
Outline

- Introduction
- Petri Net
- Scheduling Concept
- **Embedded Software Synthesis and Prototyping Methodology**
- Embedded System Example
- Conclusion

---

**Embedded Software Synthesis and Prototyping Methodology**

- Embedded Software Specification
- Embedded Software Analysis
  - Hardware Specification
  - Graphic Model
  - Compiler and Simulation
  - Functional Verification
    - Yes
    - No
  - Hardware Code Loading
- Software Specification
  - System Model
  - Scheduling
    - Schedulable?
      - Yes
      - No
  - Software Code Generation
  - Embedded Software Emulation Platform

---
Software Synthesis and Code Generation

- Software synthesis is a scheduling process
- We proposed an Extended Quasi-Static Scheduling (EQSS) method for the synthesis of embedded software.
- EQSS takes a set of CCPN as input along with timing and memory constraints such as periods, deadlines, and an upper bound on system memory space.

Complex-Choice Petri Nets (CCPN)

- Definition 1. Complex-Choice Petri Nets (CCPN), A Complex-Choice Petri Net is a 4-tuple \((P, T, F, M_0)\)
Extended Quasi-Static Scheduling

Definition 2. Complex Choice Set (CCS):
Given a CCPN \( A_i = (P_i, T_i, F_i, M_{i0}) \), a subset of transitions \( C \subseteq T_i \) is called a complex choice set if they satisfy the following conditions.

- There exists a sequence of the transitions in \( C \) such that any two adjacent transitions are always conflicting transitions from the same choice place.
- There is no other transition in \( T_i \setminus C \) that conflicts with any transition in \( C \), which means \( C \) is maximal.

Extended Quasi-Static Scheduling

- Construct an Exclusion Table for each CCS
- Decompose each CCS into conflict-free subsets
- Decompose each CCPN into subnets according to the conflict-free subsets
- Schedule all Conflict-Free components
Code Generation with Multiple Threads

- Each source transition in a CCPN represents an input event.
- Corresponding to each source transition, a $P$-thread is generated.
- There are two sub-procedures in the code generator, namely Visit_Trans() and Visit_Place(), which call each other in a recursive manner, thus visiting all transitions and places and generating the corresponding code segments.

Embedded Software Verification

- “What to verify” ~ CCPN models are translated into timed automata models
- “When to verify” ~ We propose to verify software after scheduling (synthesis) and before code generation.
- “How to verify” ~ Using State Graph Manipulators (SGM) tool
Outline

- Introduction
- Petri Net
- Scheduling Concept
- Embedded Software Synthesis and Prototyping Methodology
- Embedded System Example
- Conclusion
Embedded System Examples

- We use two embedded system examples to illustrate our proposed embedded software synthesis and prototyping methodology.
- Display subsystem of Vehicle Parking Management System (VPMS)
- Another example is a motor speed control system
Display subsystem of Vehicle Parking Management System (VPMS)

Petri Net Model of Display System

<table>
<thead>
<tr>
<th>Place</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>Counter value updated</td>
</tr>
<tr>
<td>P2</td>
<td>Signal polling complete</td>
</tr>
<tr>
<td>P3</td>
<td>Digit selected</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Transition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>Initial counter</td>
</tr>
<tr>
<td>t2</td>
<td>Pull signal</td>
</tr>
<tr>
<td>t3</td>
<td>Select digit</td>
</tr>
<tr>
<td>t4</td>
<td>Decrement counter</td>
</tr>
<tr>
<td>t5</td>
<td>Increment counter</td>
</tr>
<tr>
<td>t6</td>
<td>Check count</td>
</tr>
<tr>
<td>t7</td>
<td>No operation</td>
</tr>
<tr>
<td>t8</td>
<td>Display digit</td>
</tr>
</tbody>
</table>

Software Code for VPMS Display System

Display C-code

```c
{(t1 t2 t4) (t1 t2 t5) (t1 t2 t6) (t1 t2 t7) (t1 t3)}
t1;
While (true) {
if (P1) {
t2;
   Switch (P2) {
      Case Car in: t4;
      Case Car out: t5;
      Case Time stamp button pushed: t6;
      Case Default: t7;
   }/* End of Switch */
   }/* End of If */
   Else {
      t3; t8;
   }/* End of While */

```
Platform Prototyping

Outline

- Introduction
- Petri Net
- Scheduling Concept
- Embedded Software Synthesis and Prototyping Methodology
- Embedded System Example
- Conclusion
Conclusion and Future Work

- A complete methodology called ESSP was proposed for emulating hardware and synthesizing and executing embedded software
- Extended quasi-static scheduling algorithm, a code generation procedure, and an emulation platform

Future Work

- User-friendly GUI is under development
- Real time extension to embedded software synthesis
- Interrupt scheduling in embedded software synthesis
- System mapping tools will be constructed
Thank you!!