Transforming Sources to Petri Nets: A Way to Analyse Execution of Parallel Programs

International Workshop on Petri Net Tools and APplications
March 3, 2008 - Marseille - France

- Université Pierre & Marie Curie, LIP6/MoVe, Paris, France

• Fabrice Kordon
• Jean-Baptiste Voron
Objectives...

- **Intrusion Detection System** Context
  - Guarantee the “correct” behavior of a program

- **Double check**
  - During the execution: **Online Analysis**
  - Before the execution: **Offline Analysis**

- **Without code instrumentation**

- **Automatic process**
Objective: **Producing correct software**

- **Behavioral analysis**
  - Dealing with very large state-spaces!
  - Formal specifications are expensive and difficult to handle for engineers

- **Our approach**
  - **Automatic transformation** from source code to Petri nets
  - Modular construction using only relevant information for analysis
  - Reuse of existing tools dedicated to Petri net analysis
Software Analysis for Concurrent Systems

Objective: **Producing correct software**

Behavioral analysis
- Dealing with **very large** state-spaces!
- Formal specifications are **expensive** and **difficult to handle** for engineers

Our approach
- **Automatic transformation** from source code to Petri nets
- Modular construction using only **relevant information** for analysis
- Reuse of **existing tools** dedicated to Petri net analysis
Model-Based Static Analysis

Several steps:

- **Extracting relevant information** from source code
  - Structural information
  - Other dedicated information
- **Building a complete Petri net model**
- **Reducing** and optimizing the produced net
- **Analysing** the final net

```c
inside(...) {
  if (...) {
    write;
  } else {
    close;
  }
  if (unlink) {
    ...;
  }
  else {
    C
  }
```
Model-Based Static Analysis

Several steps:

- **Extracting relevant information** from source code
  - Structural information
  - Other dedicated information

- **Building a complete Petri net model**

- **Reducing** and optimizing the produced net

- **Analysing** the final net

```
C
inside(...) {
  if (...) { write; } else { close; } if (unlink) { ...; }
  else
```

Parser
Model-Based Static Analysis

Several steps:

- **Extracting relevant information** from source code
  - Structural information
  - Other dedicated information
- **Building a complete Petri net model**
- **Reducing** and optimizing the produced net
- **Analysing** the final net
Model-Based Static Analysis

Several steps:

- **Extracting relevant information** from source code
  - Structural information
  - Other dedicated information

- **Building a complete Petri net model**

- **Reducing** and optimizing the produced net

- **Analysing** the final net

![Diagram](image)
Model-Based Static Analysis

- **Extracting relevant information** from source code
  - Structural information
  - Other dedicated information
- **Building a complete Petri net model**
- **Reducing** and optimizing the produced net
- **Analysing** the final net
Model-Based Static Analysis

Several steps:

- **Extracting relevant information** from source code
  - Structural information
  - Other dedicated information
- **Building a complete Petri net model**
- **Reducing** and optimizing the produced net
- **Analysing** the final net
Model-Based Static Analysis

- **Several steps:**
  - **Extracting relevant information** from source code
    - Structural information
    - Other dedicated information
  - **Building a complete Petri net model**
  - **Reducing** and optimizing the produced net
  - **Analysing** the final net
Parser : Extracting Information from Source Code

- Use of **GCC** to extract information
  - Independence from the programming language (front-ends)
  - **Control Flow Graph (CFG)** extraction

**CFG** : First abstraction of the program
- All paths that might be traversed during execution
- Description in terms of **blocks**
- Each block contains detailed instructions
All extracted information is not relevant for analysis

- **Structural information** is systematically extracted

Introduce a flexible way to analyse source code: **Perspectives**

- Each extracted information is stored into its dedicated perspective
- Each perspective is defined using an **XML file** given to the parser
  - *Define keywords and special constructions*
- Builder module **merges** all desired perspectives to build a unique model
Parser: Dealing with Perspectives

- All extracted information is not relevant for analysis
  - **Structural information** is systematically extracted

- Introduce a flexible way to analyse source code: **Perspectives**
  - Each extracted information is stored into its dedicated perspective
  - Each perspective is defined using an **XML file** given to the parser
    - Define keywords and special constructions
  - Builder module **merges** all desired perspectives to build a unique model
All extracted information is not relevant for analysis

- **Structural information** is systematically extracted

Introduce a flexible way to analyse source code: **Perspectives**

- Each extracted information is stored into its dedicated perspective
  - Define keywords and special constructions
- Builder module merges all desired perspectives to build a unique model

**Diagram:**
- Source Code → GCC → CFG Parser
- Perspectives: St, Sy, Sc

**Perspective's descriptions**
All extracted information is not relevant for analysis

- **Structural information** is systematically extracted

Introduce a flexible way to analyse source code: **Perspectives**

- Each extracted information is stored into its dedicated perspective
- Each perspective is defined using an **XML file** given to the parser
  - Define keywords and special constructions
- Builder module **merges** all desired perspectives to build a unique model
Parser: Dealing with Perspectives

- All extracted information is not relevant for analysis
  - **Structural information** is systematically extracted

- Introduce a flexible way to analyse source code: **Perspectives**
  - Each extracted information is stored into its dedicated perspective
  - Each perspective is defined using an **XML file** given to the parser
    - Define keywords and special constructions
  - Builder module **merges** all desired perspectives to build a unique model
Readers / Writers Example

 Classical problem

• \textbf{X} readers & \textbf{Y} writers (processes) for one critical resource (size = \textbf{N})

2 constraints to verify

• Exclusive Write

• Write First

```c
void \textbf{writer} (int fd, \textit{mutex}, \textit{rfree}, \textit{rfull}) {
  int cpt = 0;
  while (1) {
    cpt++;
    sem_wait(rfree); sem_wait(mutex);
    write(fd,&cpt,sizeof(int));
    sem_post(mutex); sem_post(rfull);
  }
}
```

```c
int \textbf{reader} (int fd, \textit{mutex}, \textit{rfree}, \textit{rfull}) {
  while (1) {
    sem_wait(mutex); sem_wait(rfull);
    read(fd,&a,sizeof(int));
    sem_post(rfree); sem_post(mutex);
  }
}```
Parser: Choosing information to extract

- **Struct Perspective**
  - Uses **CFG's embedded information**
  - Locate **function's calls**

- **Syscall & Sync Perspective**
  - Looks for **keywords**

```c
void writer (...) {
    int cpt = 0;
    while (1) {
        cpt++;
        sem_wait(rfree); sem_wait(mutex);
        write(fd,&cpt,sizeof(int));
        sem_post(mutex); sem_post(rfull);
    }
}
```
Parser : Choosing information to extract

- **Struct Perspective**
  - Uses **CFG’s embedded information**
  - Locate **function’s calls**

- **Syscall & Sync Perspective**
  - Looks for **keywords**

```c
void writer (...) {
    int cpt = 0;
    while (1) {
        cpt++;
        sem_wait(rfree); sem_wait(mutex);
        write(fd,&cpt,sizeof(int));
        sem_post(mutex); sem_post(rfull);
    }
}
```
Parser: Choosing information to extract

**Struct Perspective**
- Uses **CFG’s embedded information**
- Locate **function’s calls**

**Syscall & Sync Perspective**
- Looks for **keywords**

```c
void writer(...) {
    int cpt = 0;
    while (1) {
        cpt++;
        sem_wait(rfree); sem_wait(mutex);
        write(fd,&cpt,sizeof(int));
        sem_post(mutex); sem_post(rfull);
    }
}
```
Parser: Choosing information to extract

**Struct Perspective**
- Uses **CFG’s embedded information**
- Locate **function’s calls**

**Syscall & Sync Perspective**
- Looks for **keywords**

```c
void writer(...) {
    int cpt = 0;
    while (1) {
        cpt++;
        sem_wait(rfree); sem_wait(mutex);
        write(fd,&cpt,sizeof(int));
        sem_post(mutex); sem_post(rfull);
    }
}
```

```c
;; function writer
# BLOCK 2 // # PRED: ENTRY (fallthru)
    cpt = 0;
# SUCC: 3 (fallthru)

# BLOCK 3 // # PRED: 2 (fallthru) 3 (fallthru)
    D.3243 = cpt + 1;
    cpt = D.3243;
    sem_wait (rfree);
    sem_wait (mutex);
    write (fd,&cpt,4);
    sem_post (mutex);
    sem_post (rfull);
    goto <bb 3> (<L0>);
# SUCC: 3 (fallthru)
```
Parser: Choosing information to extract

- **Struct Perspective**
  - Uses **CFG’s embedded information**
  - Locate **function’s calls**

- **Syscall & Sync Perspective**
  - Looks for **keywords**

```c
void writer (...) {
  int cpt = 0;
  while (1) {
    cpt++;
    sem_wait(rfree); sem_wait(mutex);
    write(fd,&cpt,sizeof(int));
    sem_post(mutex); sem_post(rfull);
  }
}
```

```c
;; function writer
# BLOCK 2 // # PRED: ENTRY (fallthru)
  cpt = 0;
# SUCC: 3 (fallthru)

# BLOCK 3 // # PRED: 2 (fallthru) 3 (fallthru)
  D.3243 = cpt + 1;
  cpt = D.3243;
  sem_wait (rfree);
  sem_wait (mutex);
  write (fd,&cpt,4);
  sem_post (mutex);
  sem_post (rfull);
  goto <bb 3> (<L0>);
# SUCC: 3 (fallthru)
```
void writer (...) {
    int cpt = 0;
    while (1) {
        cpt++;
        sem_wait(rfree); sem_wait(mutex);
        write(fd,&cpt,sizeof(int));
        sem_post(mutex); sem_post(rfull);
    }
}
Builder: Establishing the Structural Model

- Building the structural model
  - Use as skeleton for other perspectives

---

<table>
<thead>
<tr>
<th>;; function writer</th>
</tr>
</thead>
<tbody>
<tr>
<td># BLOCK 2</td>
</tr>
<tr>
<td># PRED: ENTRY (fallthru)</td>
</tr>
<tr>
<td># SUCC: 3 (fallthru)</td>
</tr>
<tr>
<td># BLOCK 3</td>
</tr>
<tr>
<td># PRED: 2 (fallthru) 3 (fallthru)</td>
</tr>
<tr>
<td># SUCC: 3 (fallthru)</td>
</tr>
</tbody>
</table>

---

Set of 6 dedicated rules

- Function / Blocks transformation
- Links between blocks (x2)
- Function calls (x2)
Builder : Establishing the Structural Model

 Oculus

Building the structural model

• Use as **skeleton** for other perspectives

<table>
<thead>
<tr>
<th>Stru.dat</th>
</tr>
</thead>
<tbody>
<tr>
<td>;; function writer</td>
</tr>
<tr>
<td># BLOCK 2</td>
</tr>
<tr>
<td># PRED: ENTRY (fallthru)</td>
</tr>
<tr>
<td># SUCC: 3 (fallthru)</td>
</tr>
<tr>
<td># BLOCK 3</td>
</tr>
<tr>
<td># PRED: 2 (fallthru) 3 (fallthru)</td>
</tr>
<tr>
<td># SUCC: 3 (fallthru)</td>
</tr>
</tbody>
</table>

Set of **6 dedicated rules**

• Function / Blocks transformation
• Links between blocks (x2)
• Function calls (x2)
Builder: Establishing the Structural Model

- Building the structural model
  - Use as *skeleton* for other perspectives

```
;; function writer
# BLOCK 2
# PRED: ENTRY (fallthru)
# SUCC: 3 (fallthru)
# BLOCK 3
# PRED: 2 (fallthru) 3 (fallthru)
# SUCC: 3 (fallthru)
```

- Set of **6 dedicated rules**
  - Function / Blocks transformation
  - Links between blocks (x2)
  - Function calls (x2)
Builder : Establishing the Structural Model

Building the structural model

- Use as *skeleton* for other perspectives

;; function writer
# BLOCK 2
# PRED: ENTRY (fallthru)
# SUCC: 3 (fallthru)
# BLOCK 3
# PRED: 2 (fallthru) 3 (fallthru)
# SUCC: 3 (fallthru)

Set of 6 dedicated rules

- Function / Blocks transformation
- Links between blocks (x2)
- Function calls (x2)
Builder : Establishing the Structural Model

Building the structural model

- Use as **skeleton** for other perspectives

```
;; function writer
# BLOCK 2
# PRED: ENTRY (fallthru)
# SUCC: 3 (fallthru)
# BLOCK 3
# PRED: 2 (fallthru) 3 (fallthru)
# SUCC: 3 (fallthru)
```

Set of **6 dedicated rules**

- Function / Blocks transformation
- Links between blocks (x2)
- Function calls (x2)
Builder: Establishing the Structural Model

Building the structural model

- Use as **skeleton** for other perspectives

```
;; function writer
# BLOCK 2
# PRED: ENTRY (fallthru)
# SUCC: 3 (fallthru)
# BLOCK 3
# PRED: 2 (fallthru) 3 (fallthru)
# SUCC: 3 (fallthru)
```

- Set of **6 dedicated rules**
  - Function / Blocks transformation
  - Links between blocks (x2)
  - Function calls (x2)
Builder: Establishing the Structural Model

- Use as skeleton for other perspectives

---

Set of **6 dedicated rules**
- Function / Blocks transformation
- Links between blocks (x2)
- Function calls (x2)

---

```plaintext
;; function writer
# BLOCK 2
# PRED: ENTRY (fallthru)
# SUCC: 3 (fallthru)
# BLOCK 3
# PRED: 2 (fallthru) 3 (fallthru)
# SUCC: 3 (fallthru)
```

---

**Stru.dat**
Associate Petri subnets to reference places

- All subnets must have an Entry and Exit place
- "Keep an eye" on instruction's position inside block
Builder : Adding Information from Perspectives

- Associate Petri subnets to reference places
  - All subnets must have an Entry and Exit place
  - “Keep an eye” on instruction’s position inside block

```sys.dat
# BLOCK 3
4. write(fd,&cpt,4);
```

```
1_Entry
\[\rightarrow\]
\[\text{struct}\_1\_Entry\_2\]
\[\rightarrow\]
\[\text{1}\_2\]
\[\rightarrow\]
\[\text{struct}\_1\_2\_3\]
\[\rightarrow\]
\[\text{1}\_3\]
```
Builder: Adding Information from Perspectives

- Associate Petri subnets to reference places
  - All subnets must have an Entry and Exit place
  - “Keep an eye” on instruction’s position inside block

```c
# BLOCK 3
4.  write(fd,&cpt,4);
```

```
struct_1_Entry_2
1_2
struct_1_2_3
1_3
```

```
1_3_4_pre
sys_1_3_4_write
1_3_4_post
struct_1_3_3
```

```
sys_1_3_4_write
1_3_4_post
struct_1_3_3
```
Builder: Adding Information from Perspectives

- Associate Petri subnets to reference places
  - All subnets must have an Entry and Exit place
  - “Keep an eye” on instruction’s position inside block

**Sys.dat**

```plaintext
# BLOCK 3
4. write(fd,&cpt,4);
```

**Syn.dat**

```plaintext
# BLOCK 3
2. sem_wait (rfree);
3. sem_wait (mutex);
5. sem_post (mutex);
6. sem_post (rfull);
```
Builder: Adding Information from Perspectives

- Associate Petri subnets to reference places
  - All subnets must have an Entry and Exit place
  - "Keep an eye" on instruction's position inside block

```
# BLOCK 3
4.  write(fd,&cpt,4);
```

```
# BLOCK 3
2.  sem_wait (rfree);
3.  sem_wait (mutex);
5.  sem_post (mutex);
6.  sem_post (rfull);
```
Builder: Adding Information from Perspectives

- Associate Petri subnets to reference places
  - All subnets must have an Entry and Exit place
  - “Keep an eye” on instruction’s position inside block

```plaintext
# BLOCK 3
4. write(fd,&cpt,4);
```

```
2.  sem_wait (rfree);
3.  sem_wait (mutex);
5.  sem_post (mutex);
6.  sem_post (rfull);
```
Builder: Adding Information from Perspectives

- Associate Petri subnets to reference places
  - All subnets must have an Entry and Exit place
  - “Keep an eye” on instruction’s position inside block

Sys.dat

```c
# BLOCK 3
4.  write(fd,&cpt,4);
```

```
write(fd,&cpt,4);
```

**Example Diagram:**

1. Entry
2. struct_1_Entry_2
3. struct_1_2_3
4. 1_3_4_pre
5. 1_3_4_post
6. sys_1_3_4_write

---

```
2.  sem_wait (rfree);
3.  sem_wait (mutex);
5.  sem_post (mutex);
6.  sem_post (rfull);
```

**Example Diagram:**

- Sem_0 transition
- Sem_2 transition
- Sem_1 transition
- Instructions 2, 3, 5, 6
- Blocks 2, 3, 5, 6

---

```
4.  write(fd,&cpt,4);
```

**Example Diagram:**

- Sys.dat file
- Instructions 4, 2, 6, 5
- Blocks 4, 2, 6, 5
Builder: Merging selected perspectives

- **sem_0**: 1.3.2.pre
  - sys_1_3_2_semwait
  - 1.3.2.post

- **sem_1**: 1.3.6.pre
  - sys_1_3_6_sempost
  - 1.3.6.post

- **sem_2**: 1.3.5.pre
  - sys_1_3_5_sempost
  - 1.3.5.post

- **sys_1_3_2_sempost**: 1.3.6_pre

- **sys_1_3_3_sempost**: 1.3.3_pre

- **sys_1_3_4_write**: 1.3.4_post

- **sys_1_3_5_sempost**: 1.3.6_post

- **sys_1_3_6_sempost**: 1.3.3_post

- **sem_2**: 1.3.4_pre

- **sem_1**: 1.3.4_post
Builder: Merging selected perspectives

Diagram:

- **sem_0**: 1_3_2_pre
- **sys_1_3_2_semwait**: 1_3_2_post

- **sem_1**: 1_3_6_pre
- **sys_1_3_6_sempost**: 1_3_6_post

- **sem_2**: 1_3_5_pre
- **sys_1_3_5_sempost**: 1_3_5_post

- **sem_2**: 1_3_3_pre
- **sys_1_3_3_semwait**: 1_3_3_post

- **sem_2**: 1_3_4_pre
- **sys_1_3_4_write**: 1_3_4_post

- **sem_1**: 1_3_4_post

Diagram shows transitions between states with labeled pre and post conditions.
Builder: Merging selected perspectives

Diagram showing the flow of 1_3_2_pre, 1_3_2_post, 1_3_3_pre, 1_3_3_post, sys_1_3_2_semwait, sem_0, 1_3_4_pre, sys_1_3_4_write, 1_3_4_post, 1_3_5_pre, 1_3_5_post, sys_1_3_5_sem_post, sem_2, 1_3_6_pre, 1_3_6_post, sys_1_3_6_sem_post, sem_1, 1_3_7_pre, 1_3_7_post, sys_1_3_7_sem_post, sem_2, 1_3_8_pre, 1_3_8_post, sys_1_3_8_sem_post, sem_2.
Builder : Merging selected perspectives
Builder: Merging selected perspectives

Diagram:
- sem_0
  - 1_3_2_pre
    - sys_1_3_2_semwait
  - 1_3_3_pre
    - sys_1_3_3_semwait
    - 1_3_4_pre
      - sys_1_3_4_write
  - 1_3_5_pre
    - sys_1_3_5_semwait
    - 1_3_5_post

- sem_2
  - 1_3_6_pre
    - sys_1_3_6_semwait
  - 1_3_6_post

Graphical representation of the process flow.
Builder: Merging selected perspectives
Builder: Merging selected perspectives

Diagram: State transitions for selected perspectives.
Builder: Merging selected perspectives
Builder: Merging selected perspectives
Builder: Merging selected perspectives

Writer (1)
Builder : Merging selected perspectives
Optimization : Reducing Petri nets

- Objective: Do not alterate the modeled behavior
  - Only consider *structural places or transitions*

- Use of **Haddad’s reductions** (adapted to fit our strategy)
  - Pre-Agglomeration of transitions
  - Post-Agglomeration of transitions
  - *Diamond reductions*

- **New reductions** will be used as soon as necessary
  - New perspectives / New constructions

- On benchmarks: up to 65% gain
Optimization: Reducing Petri nets

Objective: Do not alterate the modeled behavior

- Only consider structural places or transitions

Use of Haddad’s reductions (adapted to fit our strategy)

- Pre-Agglomeration of transitions
- Post-Agglomeration of transitions
- Diamond reductions

New reductions will be used as soon as necessary

- New perspectives / New constructions

On benchmarks: up to 65% gain
Analysis: Structural Information

- **Particular structures**
  - Structural **infinite loops**
    - Cycle without exit condition
  - Structural **dead code**
    - Subnet not connected to the main Petri net (without initial marking)

- Other assertions coming from **programming good practices**
Analysis: Behavioral Information

❖ Check some specific configurations (eg. security concerns)

❖ Exclusive Write
  - Reachability: \( \text{card}(1_3_4_{\text{post}}) > 1 \) \( \lor \) \( (\text{card}(1_3_4_{\text{post}}) > 0) \land (\text{card}(0_3_4_{\text{post}}) > 0) \)

❖ Write First
  - CTL: \( \text{AG } (\text{card}(0_3_4_{\text{post}}) = 0) \lor (\text{card}(1_3_4_{\text{post}}) > 0) \)  
    - Failed!

❖ Existing deadlocks?
  - Yes. When a reader first accesses to the critical resource.

❖ Use of CPN-AMI to analyse the model
Analysis : Behavioral Information

๏ Check some specific configurations (e.g. security concerns)

๏ Exclusive Write

  • Reachability: \((\text{card}(1\_3\_4\_post) > 1) \lor (\text{card}(1\_3\_4\_post) > 0) \land \text{card}(0\_3\_4\_post) > 0)\)

๏ Write First

  • CTL: \(\text{AG} (\text{card}(0\_3\_4\_post) = 0) \lor \text{card}(1\_3\_4\_post) > 0)\)

  • Failed!

๏ Existing deadlocks?

  • Yes. When a reader first accesses to the critical resource.

๏ Use of CPN-AMI to analyse the model
Analysis: Behavioral Information

- Check some specific configurations (e.g., security concerns)

- Exclusive Write
  - Reachability: \((\text{card}(1\_3\_4\_post) > 1) \lor (\text{card}(1\_3\_4\_post) > 0) \land \text{card}(0\_3\_4\_post) > 0)\)

- Write First
  - CTL: \(\text{AG} \ (\text{card}(0\_3\_4\_post) = 0) \lor \text{card}(1\_3\_4\_post) > 0\)
  - Failed!

- Existing deadlocks?
  - Yes. When a reader first accesses to the critical resource.

- Use of CPN-AMI to analyse the model
Conclusion & Perspectives

Automatic way to build Petri nets from source code

- Use of GCC’s internal representations
- Focus on specific program’s aspects describes as perspectives

Our prototype scales up

<table>
<thead>
<tr>
<th></th>
<th>whois</th>
<th>ping</th>
<th>gzip</th>
</tr>
</thead>
<tbody>
<tr>
<td>program’s size (loc)</td>
<td>874</td>
<td>2454</td>
<td>7323</td>
</tr>
<tr>
<td>model size (nodes)</td>
<td>1499</td>
<td>2348</td>
<td>5692</td>
</tr>
<tr>
<td>optimized model size (nodes)</td>
<td>627</td>
<td>1037</td>
<td>3301</td>
</tr>
</tbody>
</table>

Future work

- Development of new perspectives (signals, array bounds...)
- Add data flow management to our parser
- Migrate information extraction to be independent from the input language
Thank you for your attention

Questions?