A Tool Chain for the Automatic Generation of Circus Specifications from Control Law Diagrams

Chris Marriott
Ana Cavalcanti  Frank Zeyda
2nd November 2010
Outline

• Introduction
  – Motivation
  – Objectives

• Tool Chain
  – Existing Tools
  – Modifications
  – New Tools & Techniques

• Conclusions & Future Work
Motivation

• Control systems are used in safety-critical applications
• MATLAB’s Simulink is the *de facto* standard in Avionics and Automotive industries
• *Circus* is capable of expressing highly concurrent systems using Z and CSP
• Refinement strategy for *Circus*
• Verification of Ada programs
• Existing tool support requires manual intervention
Objectives

• Provide a seamless translation from Simulink to *Circus*
  – Simulink
  – ProofPower
  – CZT
  – Circus Tools

• Extend translation capabilities
  – Extract type information
  – Enabled & Action Sub-systems

• Produce a valid specification
Designed Tool Chain

- ClawZ Library
  - Z Producer
    - Type Extractor
      - Simulink Diagram (mdl file)
    - Z Database Definitions
  - Simulink Type Information
    - Merge
      - Generic Converter
        - CZT Converter
          - ClawCircus
            - Complete Circus Model
            - LaTeX
              - Circus Document
                - Necessary LaTeX Styles For Circus
            - Circus Parser & Type Checker
Designed Tool Chain

ClawZ Library

Z Producer

Z Database Definitions

Type Extractor

Simulink Type Information

Generic Converter

CZT Converter

Merge

ClawCircus

Simulink Diagram (mdl file)

Necessary LaTeX Styles For Circus

LaTeX

Complete Circus Model

Circus Document

Circus Parser & Type Checker
Designed Tool Chain

- ClawZ Library
- Derivative
  - Differentiator
  - Kd
  - Sd
- Proportion
  - Kp
  - Sp
- Integral
  - Ki
  - Si
  - Int
- Sum
- Circus Document
- Circus Parser & Type Checker
Designed Tool Chain

Process for pid/Diff/Sum

channelset pid__Diff__Sum_InpCSet == {}
    pid__Diff__In1,
    pid__Diff__UnitDelay_out

process pid__Diff__Sum_Process ≜ begin

state pid__Diff__Sum_State == []

Init

pid__Diff__Sum_State’
true

Calculate_pid__Diff__Sum

Δpid__Diff__Sum_State
In1? : ℝ
In2? : ℝ
Out1! : ℝ

(∃ b : pid__Diff__Sum •
    In1? = b.In1? ∧
    In2? = b.In2? ∧
    Out1! = b.Out1!)

Calculate_pid__Diff__Sum_out ==
(Calculate_pid__Diff__Sum)
∧
∃ pid__Diff__Sum_State
Designed Tool Chain

- ClawZ Library
  - Z Producer
    - Type Extractor
      - Simulink Diagram (mdl file)
    - Z Database Definitions
      - Simulink Type Information
      - Merge
        - ClawCircus
        - Necessary LaTeX Styles For Circus
          - Complete Circus Model
          - LaTeX
            - Circus Document
            - Circus Parser & Type Checker
Designed Tool Chain

ClawZ Library

Z Producer

Type Extractor

Simulink Diagram (mdl file)

Z Database Definitions

Simulink Type Information

Generic Converter

CZT Converter

Merge

ClawCircus

Necessary LaTeX Styles For Circus

LaTeX

Complete Circus Model

Circus Parser & Type Checker

Circus Document
Designed Tool Chain

1. Simulink Diagram (mdl file)
2. Z Producer
3. Z Database Definitions
4. Generic Converter
5. CZT Converter
6. Merge
7. ClawCircus
8. LaTeX
9. Complete Circus Model
10. Circus Parser & Type Checker
11. Circus Document
12. Necessary LaTeX Styles For Circus
13. ClawZ Library
14. Type Extractor
15. Simulink Type Information
Integrating Data Types

• Data types in Simulink vs. *Circus*
  – double -> Real, int8 -> Integer, uint8 -> Natural, ...

• Data dimensions in Simulink vs. *Circus*
  – scalar, vector (1D), vector (2D), matrix

• Additional type definitions

• Data-dependent proofs feasible
Parsing and Type Checking

• CZT markup
  – Syntactic errors

• Incomplete specifications
  – Translation errors

• Specific errors
  – Special blocks and signals
  – Parallel processes
  – Generic definitions
Translation Configuration File

- ClawCircus GUI removed
- Translation parameters required
- Hierarchical structure

\[
\begin{align*}
\text{<PID>} & \quad \text{Name of diagram} \\
\text{<unsimplified>} & \quad \text{(Un)simplified spec.} \\
\text{"PID / +"} & \quad \text{Block / Subsystem to translate} \\
\text{"Diff / +"} & \quad \text{Hierarchical structure} \\
\text{"Int / -"} & \quad \text{...}
\end{align*}
\]
Generic Definitions Converter

- ClawZ definitions use the Universe type
- Data types are inferred automatically by ProofPower

\[
\begin{align*}
\text{Test} & \quad \text{In1?} : \mathbb{U}; \\
& \quad \text{Out1!} : \mathbb{U} \\
& \quad \text{Out1!} = \text{In1?}
\end{align*}
\]

- Generic definitions in ProofPower are not standard Z
Generic Definitions Converter

• CZT requires standard Z
  – Replace all instances of the Universe type, or
  – Convert generic definitions into standard Z

Test
  In1? : \text{\texttt{U}};
  Out1! : \text{\texttt{U}}

Out1! = In1?

Test\_Generic[X]
  In1? : X;
  Out1! : X

Out1! = In1?
Tool Chain Automation

- Tool developed
- Certain tools not automated – Z Producer (ClawZ)
- No user input required
Industrial Example

- Successfully applied translation to the NDI controller (Nonlinear Dynamic Inversion controller)

- Non-trivial example
  - Nested sub-systems
  - Generic definitions
  - Range of data types

- Approximately 38,000 lines of Circus
Enabled & Action Sub-Systems

Diagram:
- **In1**
- **u1** (if \( u1 > 4 \) then **else**)
- **Out1**
- **Subsystems**
  - **Constant** with value 1
  - **Unit Delay**
  - **Action Port**
  - **if {}**
  - **else {}**

Connections:
- From **In1** to **u1**
- From **u1** to **if** and **else**
- From **if** and **else** to **Out1**
- From **Out1** to **2**
- From **Out1** to **if Action Subsystem1**
- From **if Action Subsystem1** to **2**
- From **Out1** to **if Action Subsystem**
- From **if Action Subsystem** to **2**
Enabled & Action Sub-Systems

• Behaviour based on a condition

• Currently not translatable

• Additional Schema definitions required
  – Behaviour on enabling
  – Behaviour when enabled
  – Behaviour when disabled
Enabled & Action Sub-Systems

• Sub-Systems with an Action/Enabled block
  – **Held**: Blocks hold their state when the sub-system is enabled
  – **Reset**: Block states are reset when the sub-system is enabled

• Blocks within Action/Enabled Sub-Systems
  – **Held**: Blocks hold their output value when the sub-system is disabled
  – **Reset**: Block values return to their default when the sub-system is disabled
### Enabled & Action Sub-Systems

<table>
<thead>
<tr>
<th>Internal Blocks</th>
<th>Sub-System</th>
<th>Held</th>
<th>Reset</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Reset</td>
<td>• Output = initial value (disabled)</td>
<td>• Output = initial value (disabled)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Block states reset (on enable)</td>
</tr>
<tr>
<td></td>
<td>Held</td>
<td>• Output = previous state (disabled)</td>
<td>• Output = previous state (disabled)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Block states reset (on enable)</td>
</tr>
</tbody>
</table>
Conclusions & Future Work

• Conclusions
  – Significantly reduced user input
  – More comprehensive specification produced
  – Correct specification according to parser and type checker
  – Applicable to large-scale industrial examples

• Future work
  – Integrated error reporting
  – Enabled & Action subsystem translation
  – Refinement