

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

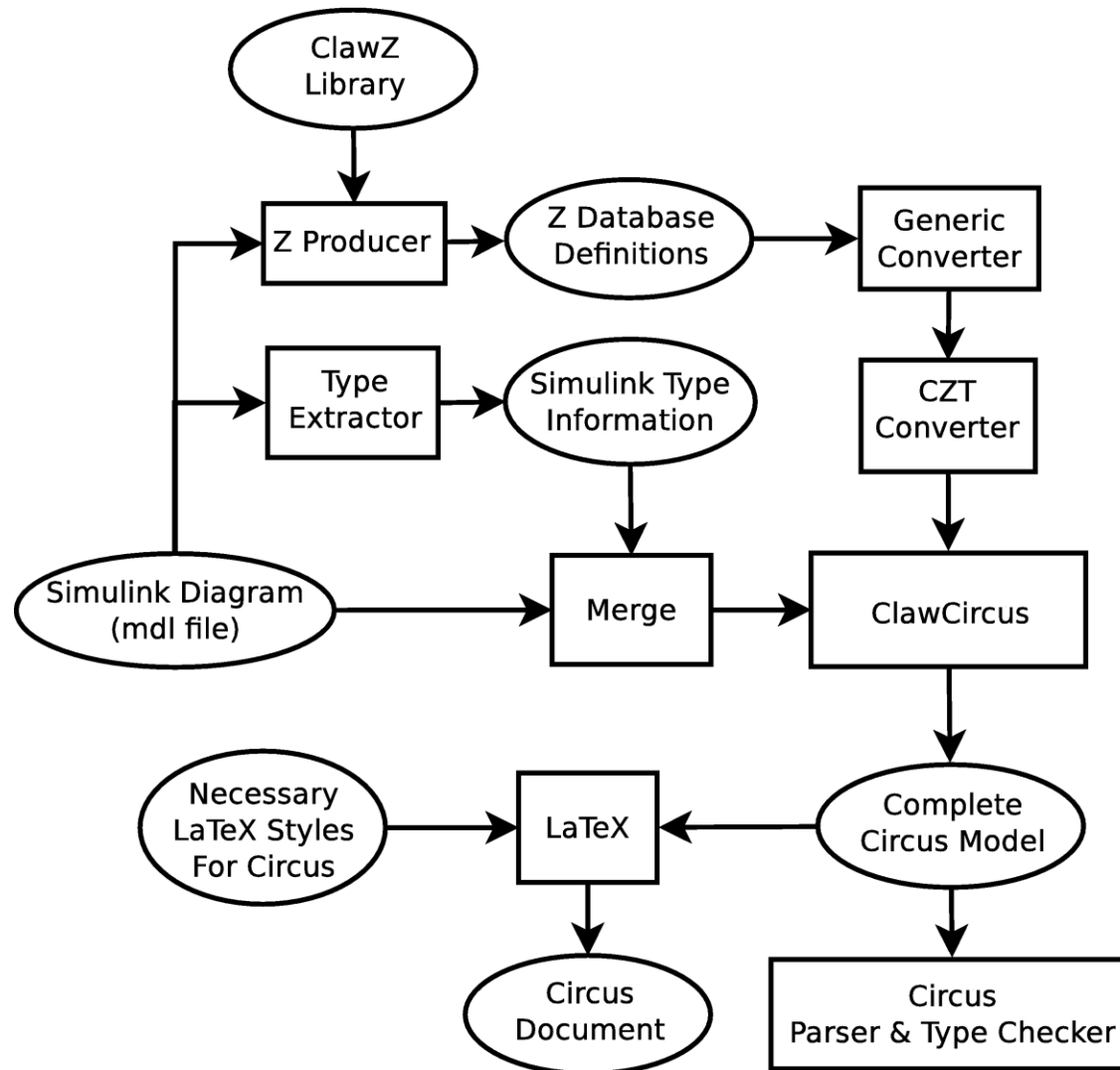
Motivation

- 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

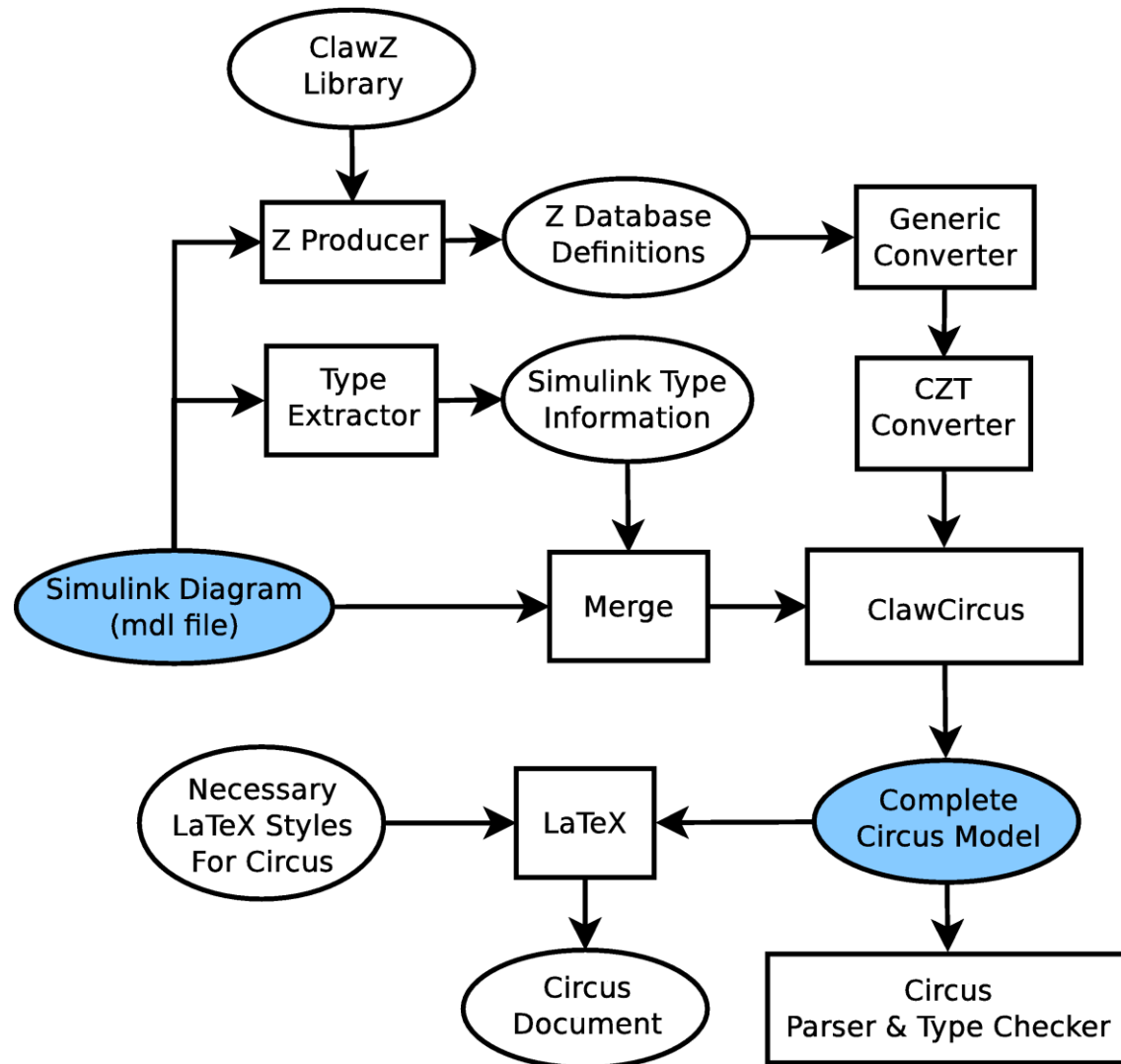
Objectives

- 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

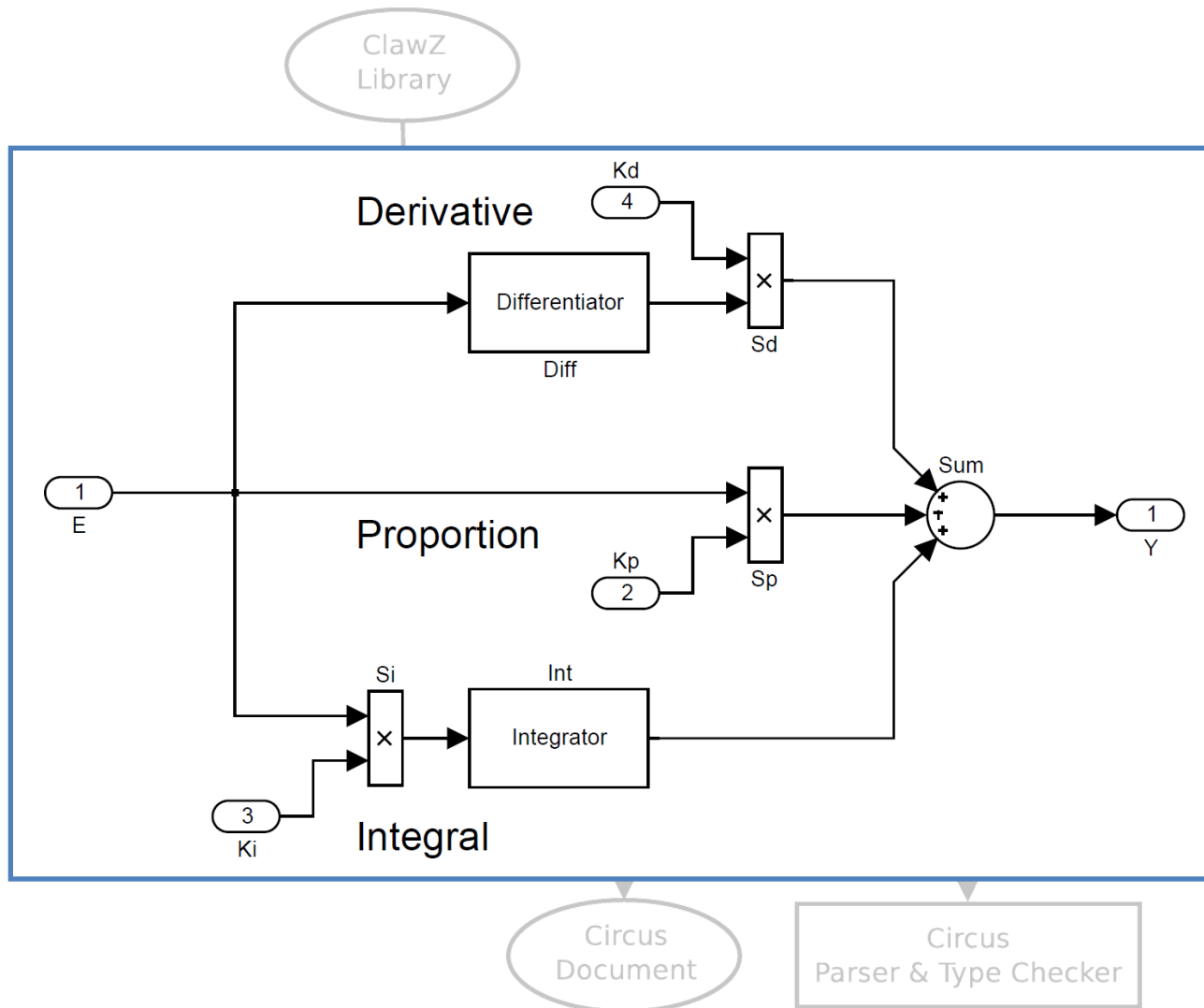
Designed Tool Chain



Designed Tool Chain



Designed Tool Chain



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  $\hat{=}$  begin
```

```
state pid__Diff__Sum__State == []
```

Init

```
pid__Diff__Sum__State'
```

```
true
```

Calculate_pid__Diff__Sum

```
 $\Delta$ pid__Diff__Sum__State
```

```
In1? :  $\mathbb{R}$ 
```

```
In2? :  $\mathbb{R}$ 
```

```
Out1! :  $\mathbb{R}$ 
```

```
( $\exists$  b : pid__Diff__Sum •
```

```
  In1? = b.In1?  $\wedge$ 
```

```
  In2? = b.In2?  $\wedge$ 
```

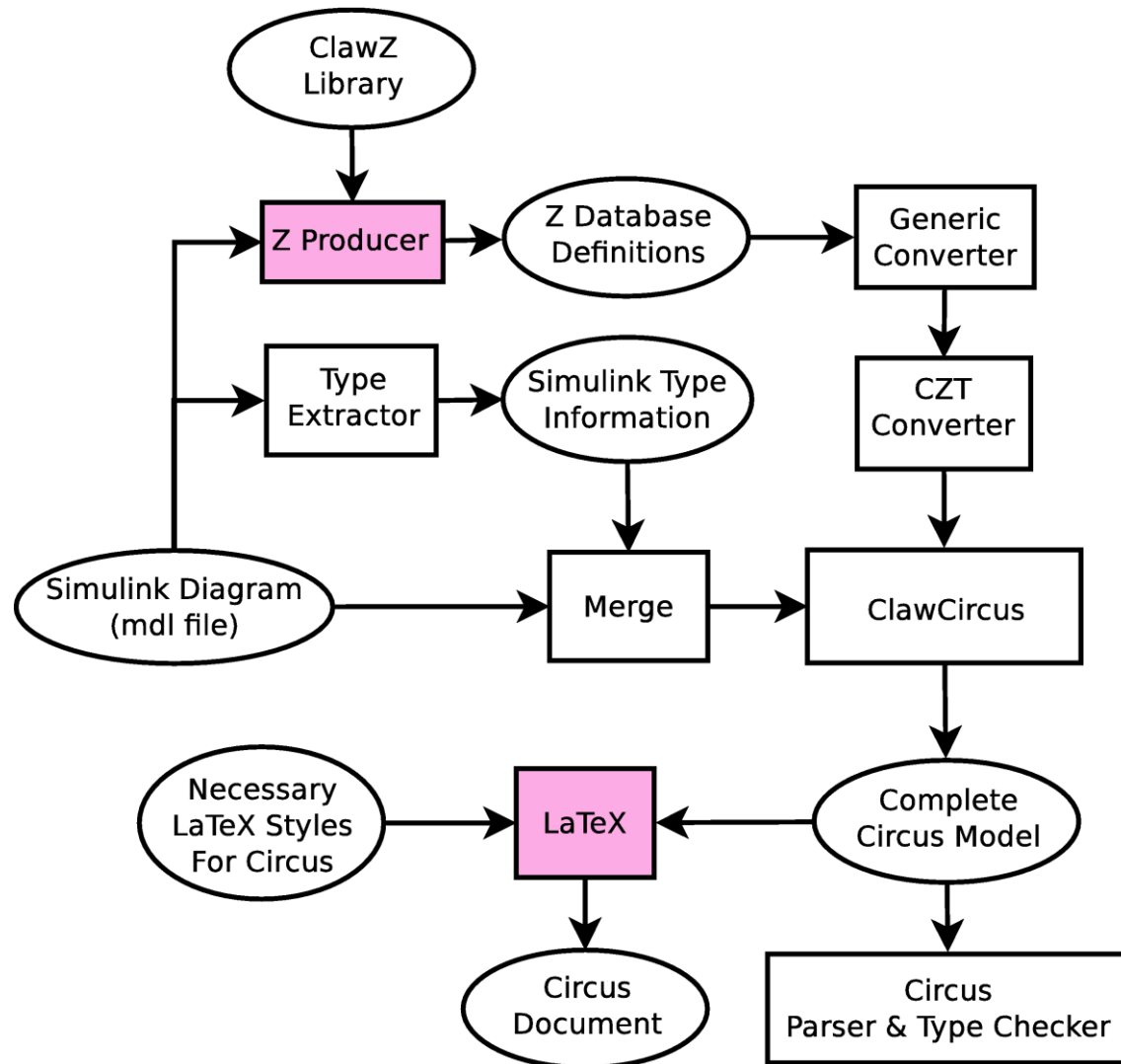
```
  Out1! = b.Out1!)
```

```
Calculate_pid__Diff__Sum__out ==  
(Calculate_pid__Diff__Sum)
```

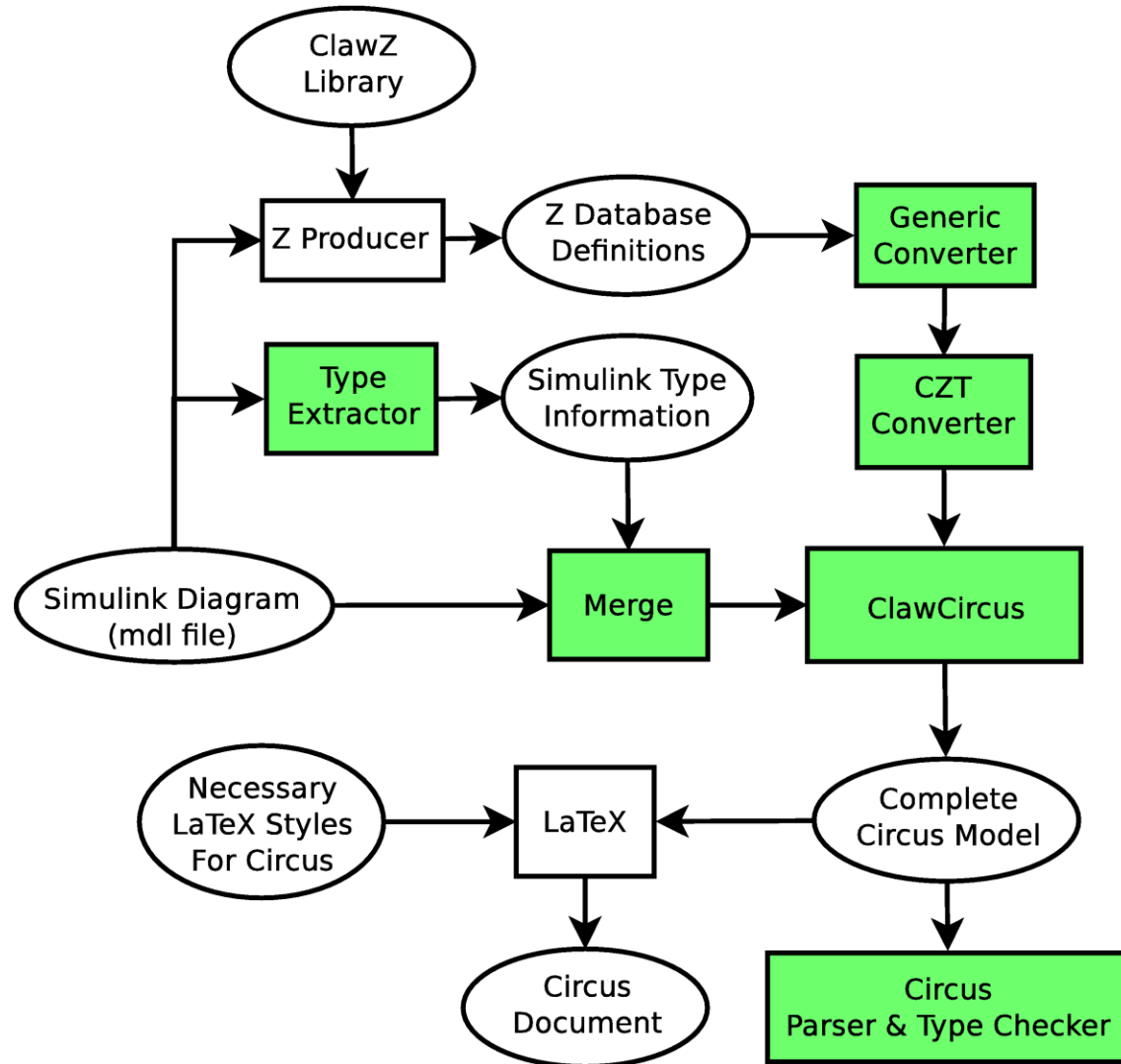
```
 $\wedge$ 
```

```
 $\exists$ pid__Diff__Sum__State
```

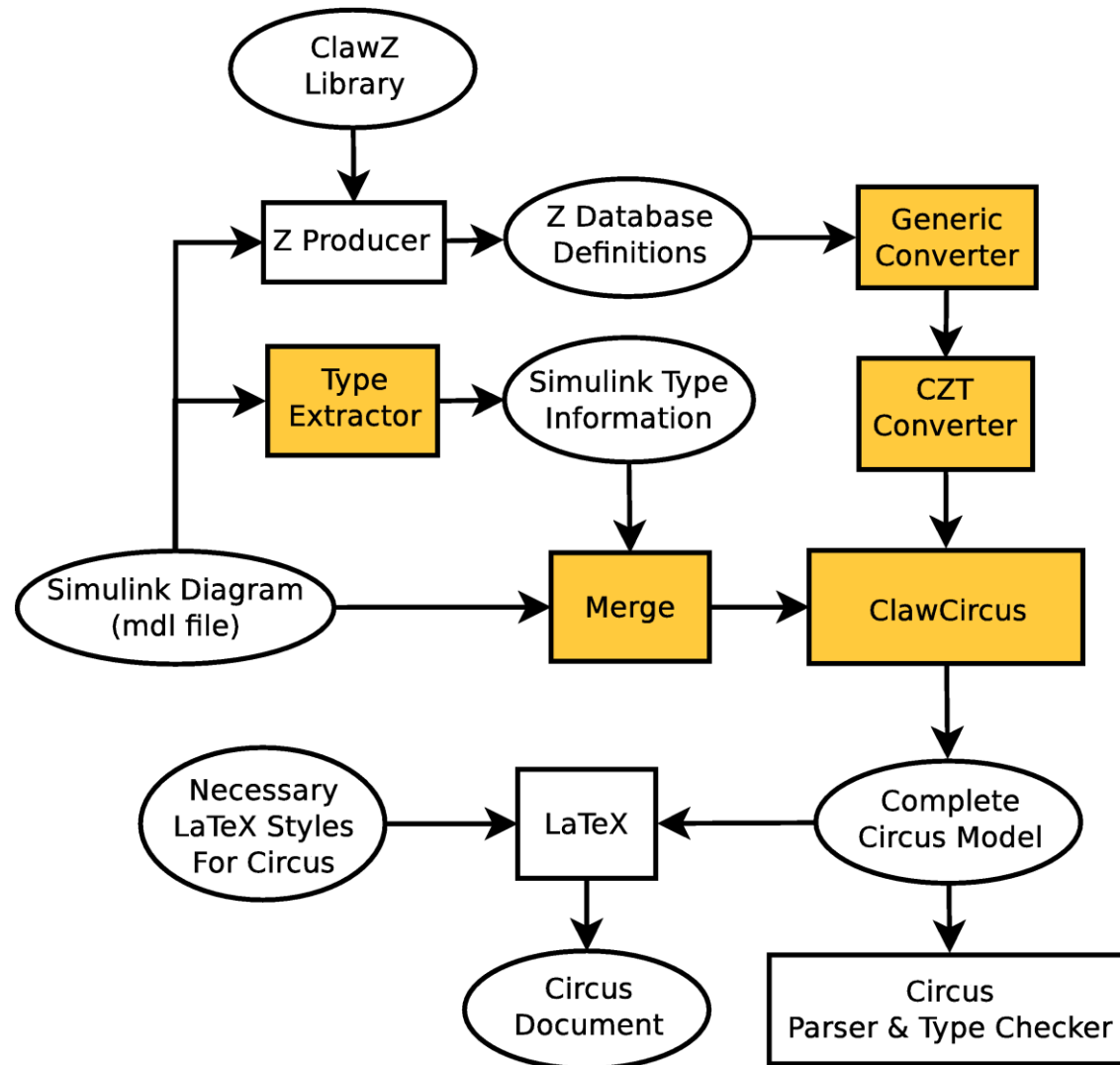
Designed Tool Chain



Designed Tool Chain



Designed Tool Chain



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

<code><PID></code>	Name of diagram
<code><unsimplified></code>	(Un)simplified spec.
<code>"PID / +"</code>	Block / Subsystem to translate
<code>"Diff / +"</code>	Hierarchical structure
<code>"Int / -"</code>	...

Generic Definitions Converter

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

```
Test _____  
In1? : U;  
Out1! : U  
-----  
Out1! = In1?
```

- Generic definitions in ProofPower are not standard Z

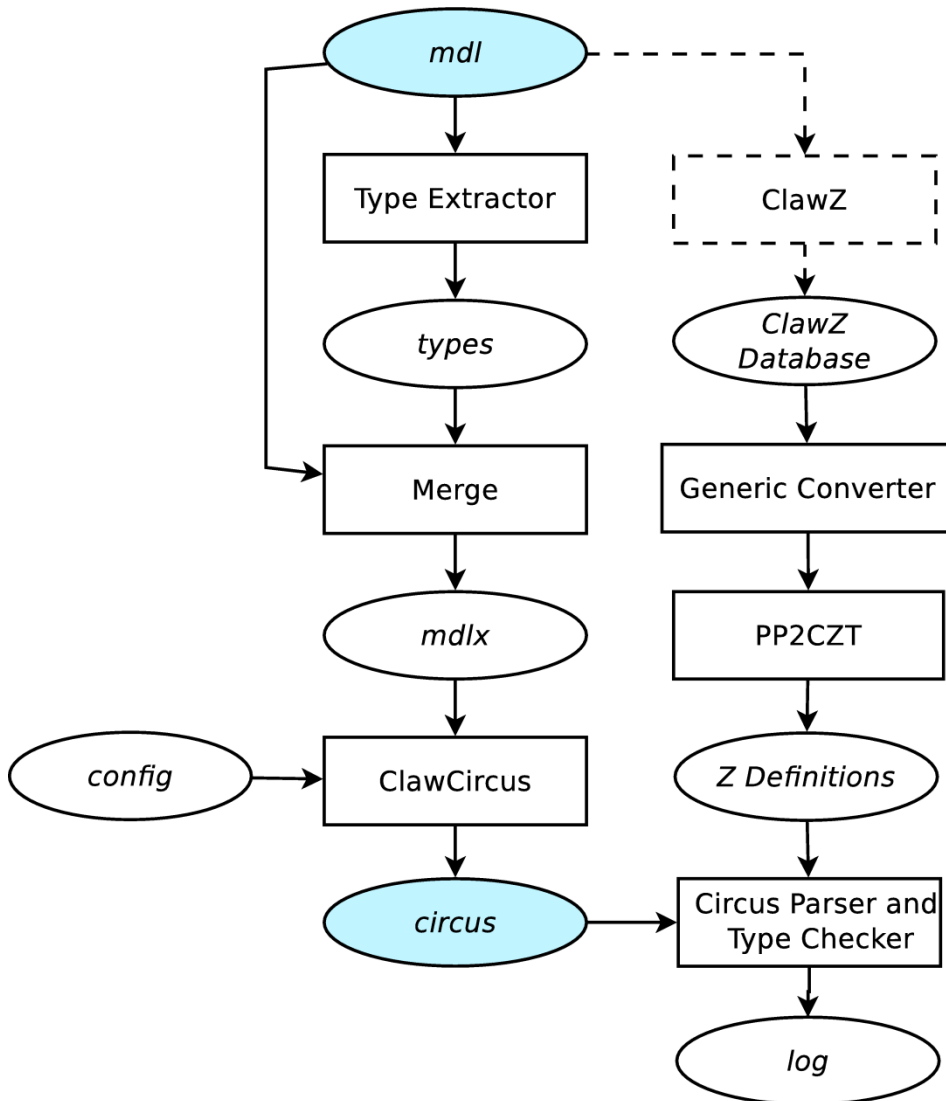
Generic Definitions Converter

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

```
Test _____  
In1? : U;  
Out1! : U  
-----  
Out1! = In1?
```

```
Test_Generic[X] —  
In1? : X;  
Out1! : X  
-----  
Out1! = In1?
```

Tool Chain Automation

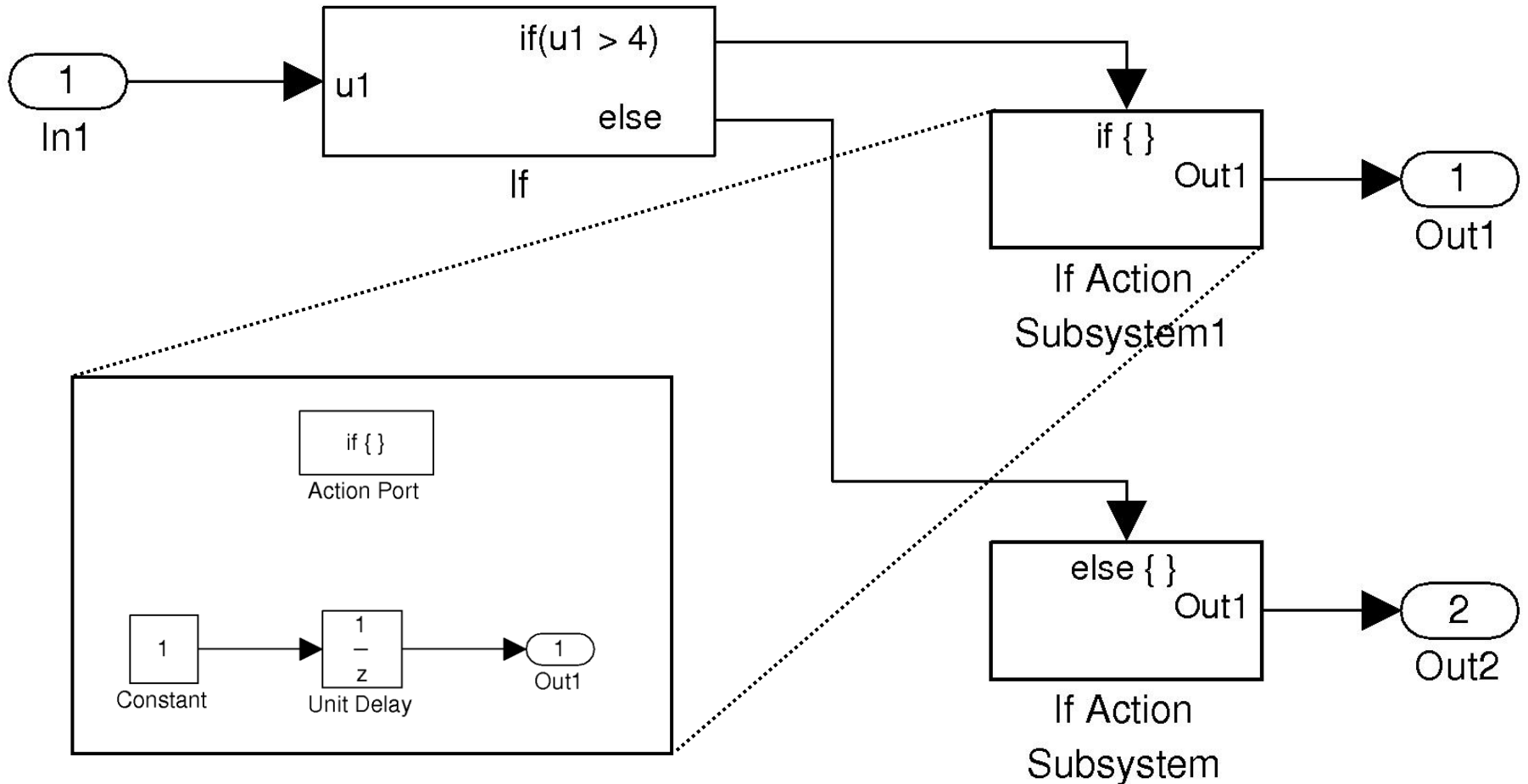


- Tool developed
- Certain tools not automated
 - Z Producer (ClawZ)
- No user input required

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



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

		Sub-System	
		Held	Reset
Internal Blocks	Reset	<ul style="list-style-type: none">• Output = initial value (disabled)	<ul style="list-style-type: none">• Output = initial value (disabled)• Block states reset (on enable)
	Held	<ul style="list-style-type: none">• Output = previous state (disabled)	<ul style="list-style-type: none">• Output = previous state (disabled)• Block states reset (on enable)

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