Structured Text and Test Tables

Overview

1. PLC software
   - The two parts...
   - Variable Declaration
   - User-defined Types

2. Structured Text
   - Overview
   - Statements
   - Not mentioned here
   - Type/Var-Example

3. Generalized Test Tables
   - Concrete Table
   - Generalization
Disclaimer

Everything is stripped down to the knowledge needed for this PSE.
Environment

- Reading Sensors
  - Logic
    - Setting Actuators

- Triggered by timer every $n$ ms
- Unlimitedly often & hard realtime
Environment

- triggered by timer every $n$ ms
- unlimitedly often & hard realtime

- Logic runtime is limited
  - No Recurrence
  - Bounded loops
  - Bo infinite nested data structures

- Logic built-up from:
  - Functions
  - Function Blocks (Function with state)
  - Program (Function Block + callable by PLC)
Environment

- **Reading Sensors**
- **Logic**
- **Setting Actuators**

- triggered by timer every $n$ ms
- unlimitedly often & hard realtime

- Logic runtime is limited
  - No Recurrence
  - Bounded loops
  - Bo infinite nested data structures

- Logic built-up from:
  - Functions
  - Function Blocks (Function with state)
  - Program (Function Block + callable by PLC)
Environment

- Reading Sensors
  - Logic
    - Setting Actuators

- triggered by timer every $n$ ms
- unlimitedly often & hard realtime

PLC software can be composed:
- **Structured Text (ST)**
- Sequential Function Chart (SFC)
- Instruction List (IL)
- Ladder Diagram (LD)
- Function Block Diagram (FBD)
The two parts

Every block has a variable declaration and an implementation body.

```
FUNCTION_BLOCK  ⟨name⟩

⟨DECL⟩

⟨BODY⟩

END_FUNCTION_BLOCK
```

The same for PROGRAM and FUNCTION.
Variable Declaration

\[
\text{vardecl} ::= \text{'VAR'} | \text{'VAR\_INPUT'} | \text{'VAR\_OUTPUT'} \ [\text{'CONSTANT'}]\]
\[
\langle \text{names} \rangle \ ':=' \langle \text{datatype} \rangle
\]
\[
[\ ':=' \langle \text{literal} \rangle ] \ ';'
\]

VAR\_INPUT
\[
x, y, z : \text{INT};
\]
\[
a : \text{INT} := 2;
\]
END\_VAR

- Variables defined in the block scope

- permissions:

<table>
<thead>
<tr>
<th>Type</th>
<th>Caller</th>
<th></th>
<th></th>
<th>Callee</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Read</td>
<td>Write</td>
<td>Read</td>
<td>Write</td>
<td></td>
</tr>
<tr>
<td>local</td>
<td>-</td>
<td>-</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>input</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>output</td>
<td>x</td>
<td>-</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Datatypes</td>
<td>Boolean</td>
<td>Integers</td>
<td>Floating-point</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------------------------</td>
<td>------------------</td>
<td>----------------</td>
<td>----------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BOOL</td>
<td>INT</td>
<td>REAL</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BYTE</td>
<td>SINT</td>
<td>LREAL</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>WORD</td>
<td>DINT</td>
<td>TIME</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DWORD</td>
<td>LINT</td>
<td>DATE</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LWORD</td>
<td>UINT</td>
<td>TIME_OF_DAY</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>USINT</td>
<td>DATE_AND_TIME</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>UDINT</td>
<td>STRING</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>LINT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time, duration, date and</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>character string</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Alexander Weigl – Structured Text and Test Tables

16.11.2016  7/26
Datatypes

**Boolean**
- BOOL
- BYTE
- WORD
- DWORD
- LWORD

**Integers**
- INT
- SINT
- DINT
- LINT
- UINT
- USINT
- UDINT
- UINT

**Floating-point**
- REAL
- LREAL

**Time, duration, date and character string**
- TIME
- DATE
- TIME_OF_DAY
- DATE_AND_TIME
- STRING
User-defined Types

Here only enumerations.

```plaintext
TYPE
  ⟨identifier⟩ := ( ⟨name⟩, ... );
END_TYPE
```

- Enumerations have properties of Integer:
  - ordinal scala
  - iterable (for-loop)
  - arithmetic
Structured Text

- Similar to PASCAL
- typical expressions (later)
- Statements:
  - assignment: \( \langle \text{name} \rangle := \langle \text{expr} \rangle ; \)
  - RETURN, EXIT
  - Function Block call
  - IF
  - CASE
  - FOR
  - WHILE
  - REPEAT
- (unbounded) loops are avoided
Structured Text

- Similar to Pascal
- Typical expressions (later)
- Statements:
  - Assignment: \( \langle name \rangle := \langle expr \rangle; \)
  - Return, Exit
  - Function Block call
  - If
  - Case
  - For
  - While
  - Repeat

- (Unbounded) loops are avoided
Structured Text

- Similar to Pascal
- typical expressions (later)

Statements:

- assignment: \langle name \rangle := \langle expr \rangle ;
- RETURN, EXIT
- Function Block call
- IF
- CASE
- FOR
- WHILE
- REPEAT

- (unbounded) loops are avoided
VAR fb : Stamp;
    a,b : INT;
END_VAR
fb.i1 := 1;
fb( i2 := 2, o1 => a);
b := fb.o2;

- application of the function block definitions
- differences between input/output variables
- different ways for assignment
Statement: IF

IF \langle expr \rangle THEN
\langle statements \rangle

[ELSEIF \langle expr \rangle THEN
\langle statements \rangle] +

[ELSE
\langle statements \rangle]
END_IF

- multiple ELSEIF
- optional ELSE
Statement: CASE

CASE ⟨var⟩ OF
  ⟨item⟩:
    ⟨statements⟩
  ⟨item⟩, ⟨item⟩, ⟨item⟩:
    ⟨statements⟩
  ⟨item⟩ . . ⟨item⟩:
    ⟨statements⟩
[ELSE
  ⟨statements⟩]
END_CASE

- distinction on the value of var
- support for enumeration and integers
- cases support constant values, value enumerations and ranges
Statement: FOR

\[
\text{FOR } \langle \text{intvar} \rangle := \langle \text{init} \rangle \text{ TO } \langle \text{end} \rangle \ [\text{BY } \langle \text{step} \rangle] \text{ DO } \\
\langle \text{statements} \rangle \\
\text{END\_FOR}
\]

- Looping over \textit{init} \leq \textit{intvar} \leq \textit{end}
- bounds are inclusive
- always counting upwards
  - step positive
- no guarantee for termination
Statement: WHILE/REPEAT

**While**
- pre-test loop
- iterates as long as `boolexpr` evaluates to true

**Repeat**
- post-test loop
- always one loop iteration
- iterates as long as `boolexpr` evaluates to false
Not mentioned here...

- User-defined datatypes (derived, structs)
- Direct addresses
- Arrays
- Pointer
TYPE
    States : (Red, Yellow, Green);
    LineState : STRUCT
        Running : BOOL;
        Drive : MultiMotState;
    END_STRUCT;
END_TYPE
VAR
    Input AT %IB0 : ARRAY [0..4] OF BYTE;
    Index : UINT := 5;
    Motor1 : MotorState;
    FourMotors : MultiMotState;
    MotorArray : ARRAY [0..3, 0..9] OF MotorState;
    Line : ARRAY [0..2] OF LineState;
END_VAR
Source

and Beckhoff Infosys
Test Tables
State of the Industry

What we know: Functional testing

\[ f_\pi(x) = f_p(x) \]

Problem: How to test with state?

Solution

- Description of input and output values

\[ f(\sigma_{i_0}, i_0) \rightarrow f(\sigma_0, i_1) \rightarrow f(\sigma_1, i_2) \rightarrow f(\sigma_2, i_3) \rightarrow \cdots \]

\[ o_0 \rightarrow o_1 \rightarrow o_2 \rightarrow o_3 \]
State of the Industry

What we know: Functional testing

\[ f_\pi(x) = f_\rho(x) \]

Problem: How to test with state?

Solution

- Description of input and output values

\[
\begin{align*}
  i_0 & \rightarrow f(\sigma_{init}, i_0) \rightarrow f(\sigma_0, i_1) \rightarrow f(\sigma_1, i_2) \rightarrow f(\sigma_2, i_3) \rightarrow \ldots \\
  & \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \ldots \\
  o_0 & \quad o_1 & \quad o_2 & \quad o_3
\end{align*}
\]
## Syntax

- Distinguish between input and output variables
- Cells contain concrete values
- **DURATION**: repetition of a line
Test table with concrete values.

<table>
<thead>
<tr>
<th>#</th>
<th>Inputs</th>
<th>Outputs</th>
<th>DURATION</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

Semantic

A function block \( f \) is conform to a test table \( t \), iff the output is expected if the input described input is given.

\[
\bar{f}_{\downarrow o}(\bar{t}_{\downarrow i}) = \bar{t}_{\downarrow o}
\]
## Example of a generalized test table

<table>
<thead>
<tr>
<th>#</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>DURATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>–</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>–</td>
<td>p</td>
<td>p</td>
<td>2*p</td>
<td>X</td>
<td>Z[-1]</td>
<td>&gt; 5</td>
</tr>
<tr>
<td>2</td>
<td>–</td>
<td>p+1</td>
<td>–</td>
<td>[0, p]</td>
<td>Y[-1]</td>
<td>2*Z &gt; Y</td>
<td>*</td>
</tr>
</tbody>
</table>
Overview

Abstraction
Cells in test tables describes the constraint on the variable.

References to other cells
Cells in tables can contain a reference to values encountered in other cells.

Generalization of row durations.
A table row may be repeated arbitrary often.
Overview

Abstraction
Cells in test tables describe the constraint on the variable.

References to other cells
Cells in tables can contain a reference to values encountered in other cells.

Generalization of row durations.
A table row may be repeated arbitrary often.
Overview

Abstraction
Cells in test tables describes the constraint on the variable.

References to other cells
Cells in tables can contain a reference to values encountered in other cells.

Generalization of row durations.
A table row may be repeated arbitrary often.
Cell Expression

Following ST notation:

**base**
- Literals: `2#11`, "a_string", `TRUE`
- Variable names: `x`
- References: `x[-1]`

**step**
- Unary operators: `NOT`, `-`
- Binary operators: `+`, `-`, `*`, `/`, `**`, `MOD`, `<`, `>`, `<=`, `>=`, `<>`, `=`, `AND`, `OR`, `XOR`
- Function call: `f( ... )`

### Abbreviations

<table>
<thead>
<tr>
<th>Abbrev.</th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>n</code></td>
<td><code>X = n</code></td>
</tr>
<tr>
<td><code>&lt; n</code></td>
<td><code>X &lt; n</code></td>
</tr>
<tr>
<td><code>[m, n]</code></td>
<td><code>X \geq m \land X \leq n</code></td>
</tr>
<tr>
<td><code>a, b</code></td>
<td><code>a \land b</code></td>
</tr>
<tr>
<td><code>–</code></td>
<td><code>true</code></td>
</tr>
</tbody>
</table>

(same for `>`, `\leq`, `\geq`, `\neq`)

(don’t care)
Cell Expression

Following ST notation:

**base**
- Literals: 2#11, "a_string", TRUE
- Variable names: x
- References: x[-1],

**step**
- Unary operators: NOT, -
- Binary operators: +, -, *, /, **, MOD, <, >, <=, >=, <>, =, AND, OR, XOR
- Function call: f( ... )

Abbreviations

<table>
<thead>
<tr>
<th>Abbrev.</th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>X = n</td>
</tr>
<tr>
<td>&lt; n</td>
<td>X &lt; n</td>
</tr>
<tr>
<td>[m, n]</td>
<td>X ≥ m ∧ X ≤ n</td>
</tr>
<tr>
<td>a, b</td>
<td>a ∧ b</td>
</tr>
<tr>
<td>−</td>
<td>true</td>
</tr>
</tbody>
</table>

(same for >, ≤, ≥, ≠)

(don’t care)
<table>
<thead>
<tr>
<th>Column: DURATION</th>
</tr>
</thead>
</table>

### Limitation
- rigid value, not state dependent, isolated constraint
- No variable names
- No references

### Example
- 1, 6, 9
- –
- > 5, [6, –]
## Column: DURATION

### Limitation
- rigid value, not state dependent, isolated constraint
- No variable names
- No references

### Example
- 1, 6, 9
- –
- > 5, [6, –]
Free Variables

- Bind values
  - Let you capture values
  - Reuse previous values

Back to the example...
Semantics

Generalized Test Table

A generalized test table $g$ describes a set of test tables $S(g)$.

The set is
- infinite,
- finite test cases,
- created by unrolling.

Conformity

A program $f$ is conform to a generalized test table $t$ iff every run of $f$:
- (a) satisfies one test table from the set $S(g)$ or
- (b) has no matching input sequence in the set $S(g)$. 