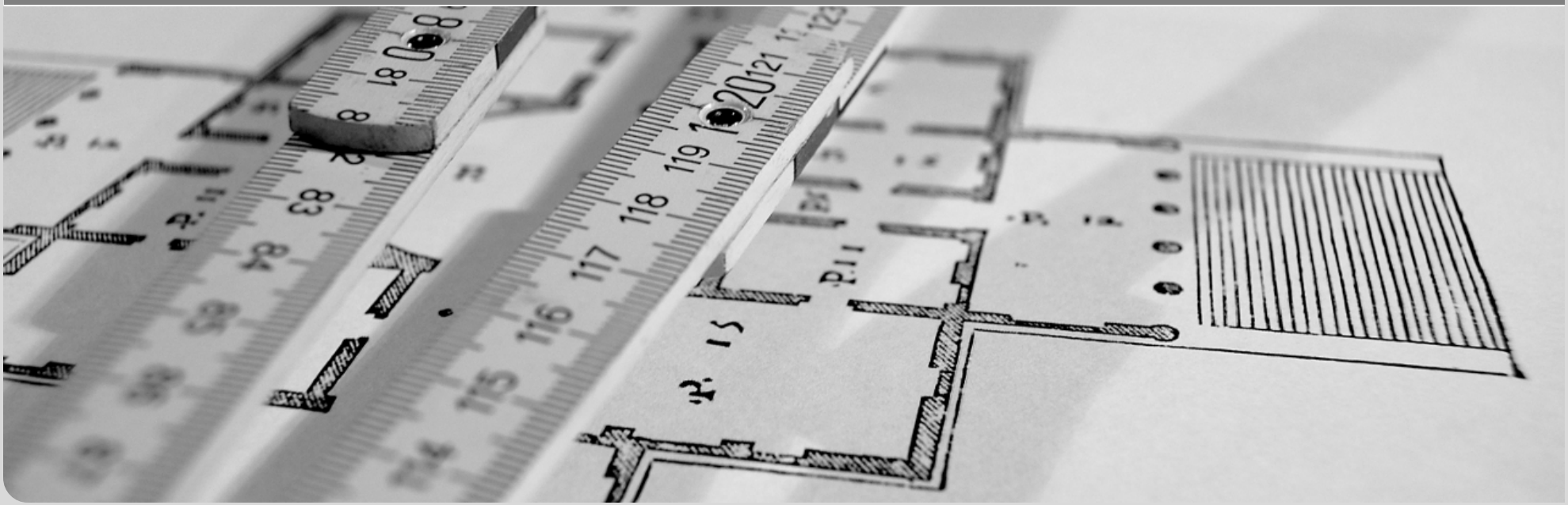
Karlsruher Institut für Technologie

IMPROVE APS

Structured Text and Test Tables

Alexander Weigl | 16.11.2016

INSTITUT FÜR THEORETISCHE INFORMATIK – ANWENDUNGSORIENTIERTE FORMALE VERIFIKATION

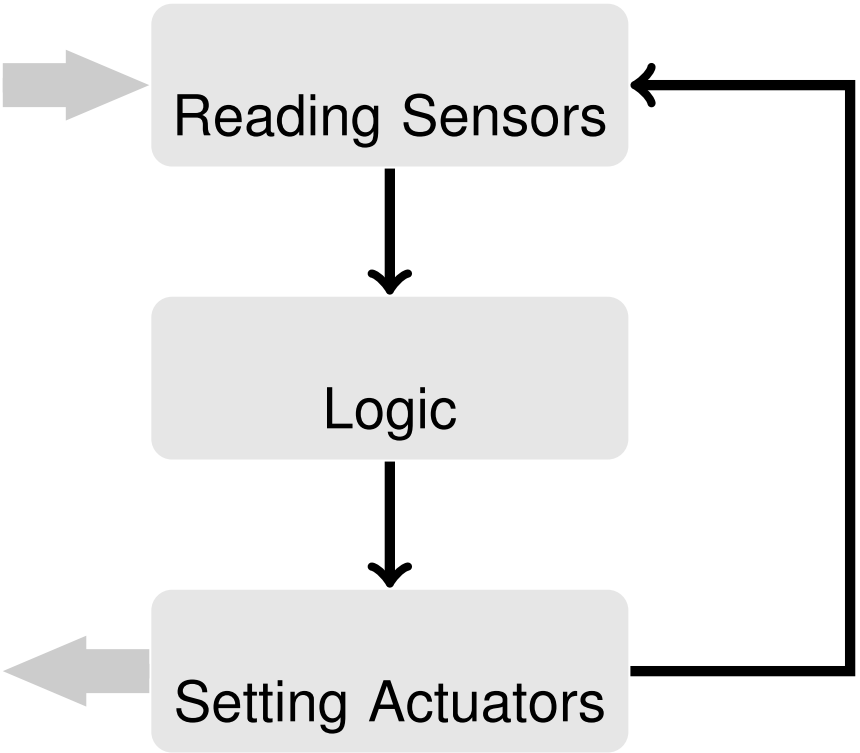


Overview

Disclaimer

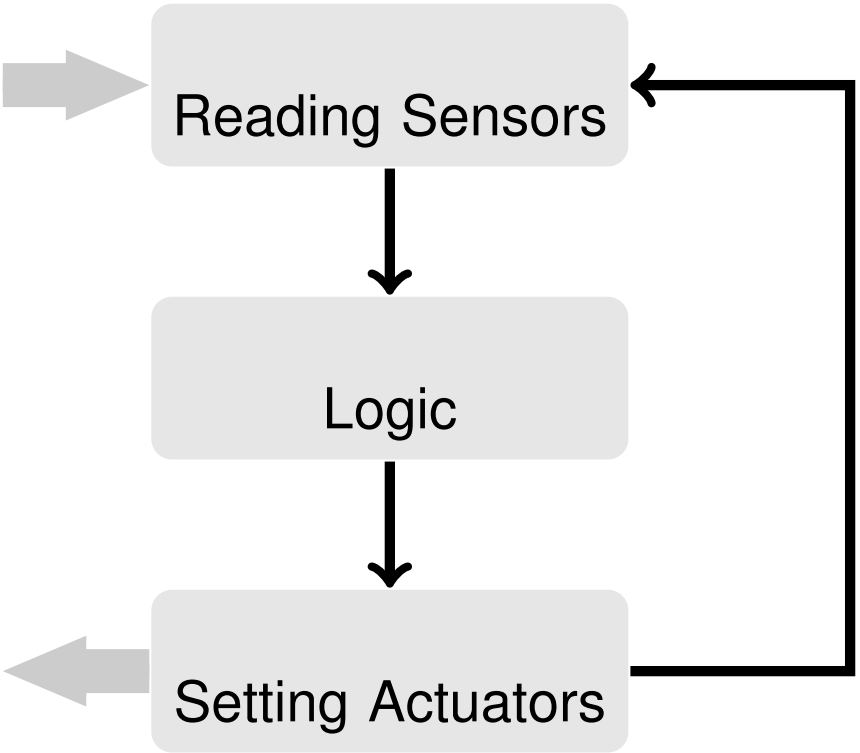
Everything is stripped down to the knowledge needed for this PSE.

Environment



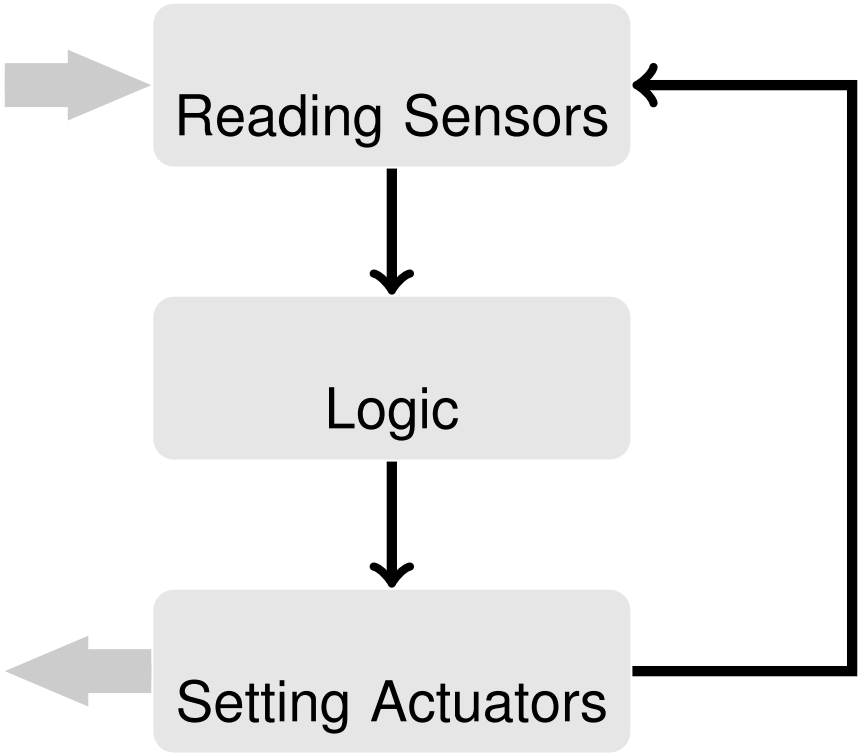
- triggered by timer every n ms
- unlimitedly often & hard realtime

Environment



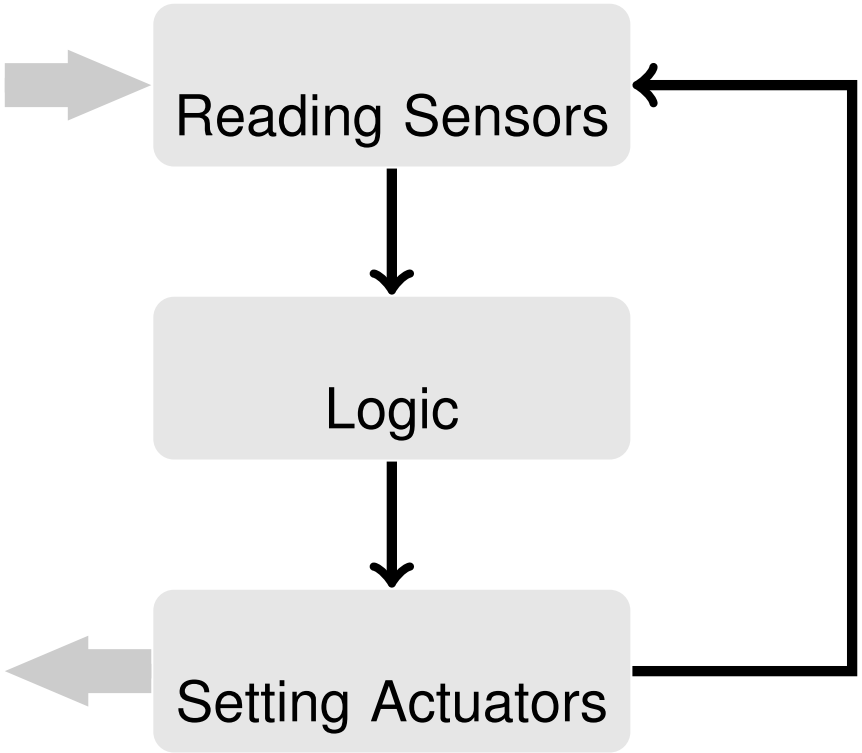
- triggered by timer every n ms
- unlimitedly often & hard realtime
- Logic runtime is limited
  - No Recurrence
  - Bounded loops
  - Bo infinite nested data structures
- Logic built-up from:
  - Functions
  - Function Blocks (Function with state)
  - Program (Function Block + callable by PLC)

Environment



- 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



- 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

```
vardecl :== 'VAR' | 'VAR_INPUT' | 'VAR_OUTPUT' ['CONSTANT']
            ⟨names⟩ ':' ⟨datatype⟩
            [':=' ⟨literal⟩] ';'
```

```
VAR_INPUT
  x, y, z : INT;
  a :INT := 2;
END_VAR
```

- Variables defined in the block scope
- permissions:

	Caller		Callee	
Type	Read	Write	Read	Write
local	-	-	x	x
input	x	x	x	-
output	x	-	x	x

Datatypes

Boolean

- BOOL
- BYTE
- WORD
- DWORD
- LWORD

Integers

- INT
- SINT
- DINT
- LINT
- UINT
- USINT
- UDINT
- ULINT

Floating-point

- REAL
- LREAL

Time, duration, date and character string

- TIME
- DATE
- TIME_OF_DAY
- DATE_AND_TIME
- STRING

Datatypes

Boolean

- BOOL
- BYTE
- WORD
- DWORD
- LWORD

Integers

- INT
- SINT
- DINT
- LINT
- UINT
- USINT
- UDINT
- ULINT

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.

```
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: ⟨*name*⟩ `:=` ⟨*expr*⟩ `;`
  - RETURN, EXIT
  - Function Block call
  - `IF`
  - `CASE`
  - `FOR`
  - `WHILE`
  - `REPEAT`
- (unbounded) loops are avoided

Structured Text

- Similar to PASCAL
- typical expressions (later)
- Statements:
  - assignment: ⟨*name*⟩ `:=` ⟨*expr*⟩ `;`
  - RETURN, EXIT
  - Function Block call
  - `IF`
  - `CASE`
  - `FOR`
  - `WHILE`
  - `REPEAT`
- (unbounded) loops are avoided

Structured Text

- Similar to PASCAL
- typical expressions (later)
- Statements:
  - assignment: ⟨*name*⟩ `:=` ⟨*expr*⟩ `;`
  - RETURN, EXIT
  - Function Block call
  - `IF`
  - `CASE`
  - `FOR`
  - `WHILE`
  - `REPEAT`
- (unbounded) loops are avoided

Statement: Function Block Call

```
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 ⟨expr⟩ THEN
    ⟨statements⟩
[ELSEIF ⟨expr⟩ THEN
    ⟨statements⟩]+
[ELSE
    ⟨statements⟩]
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

```
FOR ⟨intvar⟩ := ⟨init⟩ TO ⟨end⟩ [BY ⟨step⟩] DO
    ⟨statements⟩
END_FOR
```

- Looping over $init \leq intvar \leq end$
- bounds are inclusive
- always counting upwards
  - step positive
- no guarantee for termination

Statement: WHILE/REPEAT

```
WHILE  ⟨boolexpr⟩ DO ⟨statements⟩ END_WHILE
REPEAT ⟨statements⟩ UNTIL ⟨boolexpr⟩ END_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

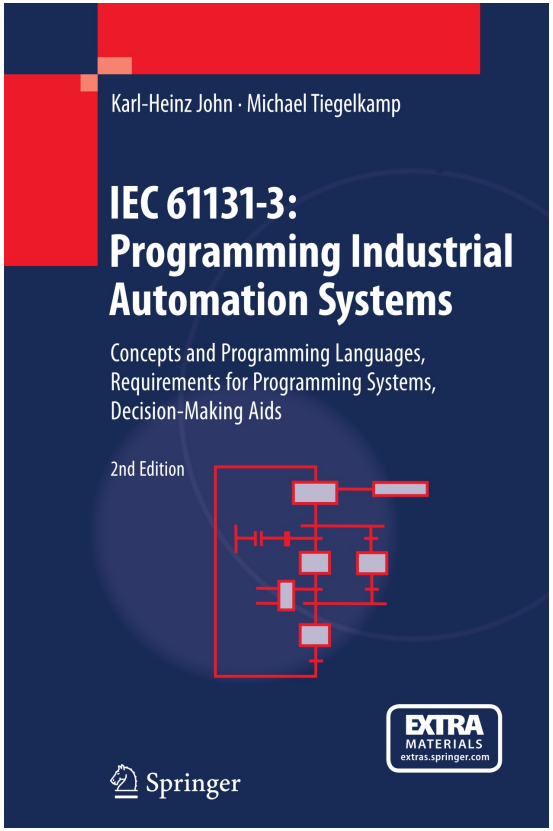
Not mentioned here...

- User-defined datatypes (derived, structs)
- Direct addresses
- Arrays
- Pointer

Example

```
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



https://link.springer.com/book/10.1007%2F978-3-642-12015-2
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

State of the Industry

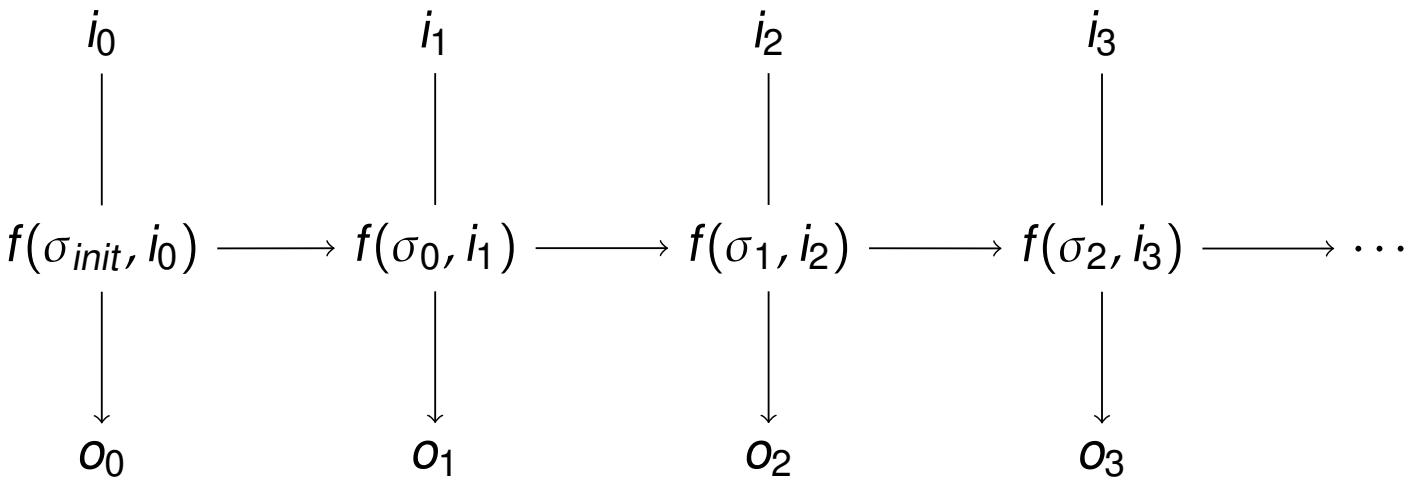
What we know: Functional testing

$$f_\pi(x) = f_p(x)$$

Problem: How to test with state?

Solution

- Description of input and output values



Test table with concrete values.

#	Inputs A	B	C	Outputs X	Y	Z	DURATION
0	1	1	2	0	0	5	1
1	0	3	3	6	6	5	7
2	1	4	2	2	8	5	2

Syntax

- distinguish between input and output variables
- cells contain concrete values
- DURATION: repetition of a line

Test table with concrete values.

#	Inputs A	B	C	Outputs X	Y	Z	DURATION
0	1	1	2	0	0	5	1
1	0	3	3	6	6	5	7
2	1	4	2	2	8	5	2

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

#	Inputs			Outputs			DURATION
	A	B	C	X	Y	Z	
0	1	1	2	0	0	–	1
1	–	p	p	$2 * p$	X	$Z[-1]$	> 5
2	–	$p + 1$	–	$[0, p]$	$> Y[-1]$	$2 * Z > Y$	$*$

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 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 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

Abbrev.	Constraint	
n	$X = n$	
$< n$	$X < n$	(same for $>, \leq, \geq, \neq$)
$[m, n]$	$X \geq m \wedge X \leq n$	
a, b	$a \wedge b$	
–	*true*	(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

Abbrev.	Constraint	
n	$X = n$	
$< n$	$X < n$	(same for $>, \leq, \geq, \neq$)
$[m, n]$	$X \geq m \wedge X \leq n$	
a, b	$a \wedge b$	
–	*true*	(don't care)

Column: DURATION

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).