Comparison of the textual description language

	KIELER SyncCharts	YAKINDU SCT	

Declarations	**Pure Signals**: signals have a present status:
	• absent or present
	• must be set for input signals
	• computed for local and output signals:
	  • for each tick absent by default unless signal is emitted
	• Example:
	  • Emit signal S: S
	  • Test for presence: S
	
	Valued Signals:
	• are pure signals than additionally are able to store a value
	• values are persistent across ticks
	• Example:
	  • Emit signal V with value 3: V(3)
	  • Test for presence: V
	  • Get the last emitted value of V: ?V
	
	Variables:
	• Are not shared between concurrent regions
	• Currently implemented by host type variables
		Events:
		• interface scope: events can either be ingoing (in event event) or outgoing (out event event).
		• local scope: events are able to store a value.
		`internal: event localEvent : bool`
		
		Variables:
		• variable:
		`var variable: string`
		• read-only variable:
		`var readonly size: int = 10`
		• external variable: can be referenced by the environment
		`var external variable: int = 44`
		`External variables are not used at the moment.`	

Types	• **int** • **bool** • **string** • **pure**: only makes sense for Signals. Signals are absent or present. • **unsigned** • **float** • **double** • **host:** no actual type is given. The given type in the hostType attribute is used.	• **integer** • **boolean** • **string** • **real** • **void** New types are going to be added. A type system abstraction is present.	
Expressions	• **Logical AND:** var1 && var2 • **Logical OR:** var1 || var2 • **Logical NOT:** !var1 `(A and B) or ((not C) and D)`	• **Logical AND:** var1 && var2 • **Logical OR:** var1 || var2 • **Logical NOT:** !var1 • **Conditional Expression:** var1 ? var2 : var3	

Operations	• **Equal**: '='
	• **Less Than: '**<'
	• **Equal Or Less Than: '**<='
	• **Greater Than:** '>'
	• **Equal Or Greater Than:** '>='
	• **NOT: '**!='
	• **Add**: '+'
	• **Minus** : '-'
	• **Multiply**: '*'
	• **Divide**: '/'
	• **Modulo**: 'mod'
	• **Value**: '?'
	`?B = 3`
	• **PRE**: 'pre':
	`pre(S):gives the presence status of S at`
	`the previous tick.`
	`pre(?S):returns the value of S at the`
	`previous tick.`
	• **NE**: '<>'
		• **Equal**: '=='
		• **less than:** '<'
		• **Equal Or Less Than**: '<='
		• **Greater Than**: '>'
		• **Equal Or Greater Than** '>='
		• **Not Equal:** '!='
		• **Plus:** '+'
		• **Minus:** '-'
		• **Multiply:** '*'
		• **Divide:** '/'
		• **Modulo:** '%'
		• **valueof()**
		• **Shift Left**: '<<'
		• **Shift Right**: '>>'
		• **Positive: '**+'
		• **Negative: '**-'
		• **Complement:** '~'	

Trigger	• Simple signal reference: I / O
	• Boolean expression:
	`(A and B) or ((not C) and D)`
	• Valued Signals and Variables can be used in conditions in these boolean expressions
	`variable > 1`
	`?A = 1`
	• Comparison
	`?A > (variable + 1)`
	`A and (3 > ?B) or ((var5 + 2) = 6)`
	• Pre
	`A and pre(B)`
	`3 < pre(?A)`
	• Immediate, #S: the trigger is satisfied as soon as the state is entered..
	• Count Delays, 3 S
	• Time in SyncCharts: Multiform notion of time,
	e.g., signal SECOND appears every second (depending on the physical tick length).
		• **event:**
		`I / raise O`
		• **after**:
		`after 20 s`
		• **every**
		`every 200 ms`
		• **always:**
		enables a reaction to be executed in every run to completion step
		• **default:**
		enables a reaction to be executed in every run to completion step
		• **else**:
		used in transitions and implies the lowest evaluation priority for that transition.
		• **entry**
		• **exit**
		• **oncycle**	

Effects	• Emission of a simple signal
	`/ A`
	• Emission of value of a valued Signal
	`/ A(3)`
	• Assignment of a variable
	`/ varA := 42`
	• Multiple effects get comma- or whitespace separated
	`/ A, B, C(25), varA := 2`
	• New values may use value expressions as explained above
	`/ A(3 + pre(?B)), varC := (varD + 1)`
		• **Assignment of a variable:** S / varA = 5
		• **raise** myvar
		• **myvar = valueof(event)**: Returns the value of an valued event that it passed to the function as parameter.
		• **mybool = active(StateA)**: Returns „true” if a state is active or „false” otherwise.	