

conventions	
$\sigma, \sigma_1, \sigma_2$	boolean expressions
τ, π	general expressions
n, m	compile-time constant expressions
α_1, α_2	data types
ℓ, ℓ_1, ℓ_2	control flow locations
compiler directives	
package name	package membership
import module	import module
macro $f(x_1, \dots, x_n) = \tau$	macro definition
//	single line comment
/* comment */	block comment
modules	
module declaration	
module $m(vdcl)\{$ <i>stat</i> }	module m with variable declarations $vdcl$, body statement $stat$ and optional task list
[<i>task list</i>]	
variable declarations $vdcl ::=$	
general syntax is a comma-separated list of single declarations [<i>storage</i>] <i>type</i> [<i>flow</i>] x_1, \dots, x_n	
storage $storage ::=$	
mem	memorized variable (store last values)
event	event variable (reset to default value)
hybrid	continuous variable for hybrid systems
information flow $flow ::=$	
?	input variable (only readable)
!	output variable (only writable)
	inout variable (readable and writable)
tasks declaration $task ::=$	
drivenby [<i>name</i>] { <i>stat</i> }	simulation with stimuli generator $stat$ (writing inputs, reading outputs)
satisfies [<i>name</i>] { [<i>obs</i>] [<i>goal</i>] list }	verification using optional observer and proof goals
observer $obs ::=$	
observer ($vdcl$) { <i>stat</i> }	observer with local declarations $vdcl$ and body statement $stat$

data types	
types $type ::=$	
bool	booleans
nat	unbounded unsigned integers
nat { n }	integers in $\{0, \dots, n - 1\}$
int	unbounded signed integers
int { n }	integers in $\{-n, \dots, 0, \dots, n - 1\}$
real	real numbers
bv	unbounded bitvectors
bv { n }	bitvector of length n
[n] α	array having n elements of type α
$\alpha_1 * \dots * \alpha_n$	tuple type
expressions	
constants	
false	boolean constant false
true	boolean constants true
instant	holds if execution takes no time
inside	holds if control flow is inside statement
terminate	holds if control flow leaves statement
time	physical time for hybrid systems
conversions	
nat2bv (τ, n)	convert nat to n -bit radix-2 number
int2bv (τ, n)	convert int to n -bit 2-complement
arr2bv (x)	convert boolean array x to bitvector
tup2bv (τ)	convert boolean tuple to bitvector
bv2nat (τ)	interpret bitvector as radix-2 number
bv2int (τ)	interpret bitvector as 2-complement num.
nat2real (τ)	convert nat to real number
int2real (τ)	convert int to real number
bitvector operations	
$\tau\{n\}$	bit τ_n of bitvector $\tau = (\tau_\ell, \dots, \tau_0)$
$\tau\{m:n\}$	segment $\tau_m \dots \tau_n$ (with $m \geq n$)
$\tau\{m:\}$	segment $\tau_m \dots \tau_0$ (with $m \geq 0$)
$\tau\{:\!n\}$	segment $(\tau_\ell, \dots, \tau_n)$ (with $\ell \geq n$)
reverse (τ)	reverse bitvector
$\tau_1 @ \tau_2$	bitvector concatenation
$\{\tau : : n\}$	concatenate n instances of boolean τ
constructing and accessing compound types	
$\tau[n]$	array access
$[\tau_0, \dots, \tau_n]$	array of $n + 1$ values
$\tau.n$	tuple access
(τ_0, \dots, τ_n)	tuple of $n + 1$ values

equality	
$\tau_1 == \tau_2$	equality
$\tau_1 != \tau_2$	inequality
numeric relations	
$\tau_1 < \tau_2$	less than
$\tau_1 \leq \tau_2$	less than or equal to
$\tau_1 > \tau_2$	greater than
$\tau_1 \geq \tau_2$	greater than or equal to
boolean operators	
! σ , not σ	negation
$\sigma_1 \& \sigma_2$, σ_1 and σ_2	conjunction
$\sigma_1 \sigma_2$, σ_1 or σ_2	disjunction
$\sigma_1 \wedge \sigma_2$, σ_1 xor σ_2	exclusive or
$\sigma_1 \rightarrow \sigma_2$, σ_1 imp σ_2	implication
$\sigma_1 \leftrightarrow \sigma_2$, σ_1 eqv σ_2	equivalence
arithmetic operators	
$+$ π	unary plus (converts nat to type int)
$-$ π	unary minus
$\tau + \pi$	addition
$\tau - \pi$	subtraction
$\tau * \pi$	multiplication
τ / π	division
$\tau \% \pi$	modulo
abs (τ)	absolute value
sat { n }(τ)	saturate τ to type nat { n } or int { n }
hybrid systems	
drv (τ)	derivation of τ by physical time
cont (τ)	switch between continuous and discrete value
other operators	
sin (π)	sinus
cos (π)	cosinus
exp (τ, π)	τ^π
log (π)	logarithm to base 2
sizeof (π)	
generic expressions	
exists ($i = m..n$) σ_i	denotes $\bigvee_{i=m}^n \sigma_i$
forall ($i = m..n$) σ_i	denotes $\bigwedge_{i=m}^n \sigma_i$
sum ($i = m..n$) τ_i	denotes $\sum_{i=m}^n \tau_i$
misc. expressions	
$(\tau ? \tau_1 : \tau_0)$	if τ then τ_1 else τ_0
next (τ)	value of τ in next step
$f(\tau_1, \dots, \tau_n)$	macro function application

statements $stat ::=$	
discrete actions	
$x = \tau$	immediate assignment
$\text{next}(x) = \tau;$	delayed assignment
$\text{emit}(x);$	immediate emission
$\text{emit next}(x);$	delayed emission
$[name :] \text{assume}(\sigma);$	assumption
$[name :] \text{assert}(\sigma);$	assertion
wait statements	
$\text{nothing};$	empty statement
$[\ell :] \text{pause};$	separate macro steps
$[\ell :] \text{halt};$	halt forever
$[\ell :] \text{[immediate] await}(\sigma);$	wait until σ holds
conditional statements	
$\text{if}(\sigma) S_1 [\text{else } S_2]$	if σ holds, execute S_1 [otherwise S_2]
$\text{choose } S_1 \text{ else } S_2$	nondeterministic choice
case	equivalent to
$(\sigma_1) \text{ do } S_1$	$\text{if}(\sigma_1) S_1$
$(\sigma_2) \text{ do } S_2$	$\text{else if}(\sigma_2) S_2$
...	...
$(\sigma_n) \text{ do } S_n$	$\text{else if}(\sigma_n) S_n$
$\text{default } S_{n+1}$	$\text{else } S_{n+1}$
sequential and parallel control flow	
$S_1 S_2$	sequential execution
$S_1 S_2$	synchronous -parallel
$S_1 S_2$	asynchronous -parallel
$S_1 S_2$	interleaved -parallel
$S_1 \&\& S_2$	synchronous &-parallel
$S_1 \&\&\& S_2$	asynchronous &-parallel
$S_1 \& S_2$	interleaved &-parallel
loops	
$\text{loop } S$	infinite loop of S
$\text{do } S \text{ while}(\sigma)$	repeat S while σ holds
$\text{while}(\sigma) S$	while σ holds, repeat S
$\text{always } S$	infinite loop of $\text{pause}; S;$
$\text{immediate always } S$	infinite loop of $S; \text{pause};$

local declarations	
$\{ \alpha x; S \}$	declare variable x of type α with scope S
$\text{let } (x = \tau) S$	abbreviate τ by x in S
generic statements	
$\text{for } (i=m .. n) S$	generic sequence
$\text{for } (i=m .. n) \text{ do } \eta S$	generic parallel with $\eta \in \{!, \&, , \&\&, , \&\&\&\}$
$\text{choose } (i=m .. n) S$	generic choice
module call	
$[\text{iname}:] m(\tau_1, \dots, \tau_n);$	means: instance iname of call to module m ;
	<ul style="list-style-type: none"> inputs of m must be readable expressions τ_i outputs of m must be writable lhs-expressions τ_i undesired outputs of m can be skipped by $_$
abortion statements	
$[\text{weak}] [\text{immediate}] \text{abort } stat \text{ when}(\sigma);$	aborts $stat$ when σ holds
suspension statements	
$[\text{weak}] [\text{immediate}] \text{suspend } stat \text{ when}(\sigma);$	suspends $stat$ when σ holds
during statements	
$[\text{immediate}] [\text{final}] \text{during } stat1 \text{ do } stat2;$	in each step of $stat1$ do also instantaneous $stat2$
hybrid systems $stat ::=$	
flow statements	
$\text{flow}[(i=m .. n)] \{ S_1; \dots; S_n \}$	perform continuous actions S_i until interrupted
$\text{flow}[(i=m .. n)] \{ S_1; \dots; S_n \} \text{until}(\sigma);$	perform continuous actions S_i until σ holds
continuous actions	
$x <- \tau$	continuous assignment
$\text{drv}(x) <- \tau;$	derivative assignment
$[name :]$	continuous assertion with
$\text{constrainSME}(\sigma);$	at least one of S, M, E

proof goals $goal ::=$	
assumption	
$name: \text{assume } spec;$	
assertion goal	
$name [vtask] [cl]: \text{assert } spec [\text{with } \{al\}];$	
	<ul style="list-style-type: none"> cl is the list of controllable variables al is the list of assumptions
verification task $vtask ::=$	
ProveE	property is true in one initial state
ProveA	property is true in all initial states
DisProveE	property is false in one initial state
DisProveA	property is false in all initial states
specifications $spec ::=$	
path quantifiers	
A φ	φ holds on all inf. computation paths
E φ	φ holds on one inf. computation path
linear time future operators	
X φ	φ holds in the next point
G φ	always φ in the future
F φ	eventual φ in the future
$[\varphi \text{SU } \psi]$	φ until ψ holds and ψ must hold
$[\varphi \text{SB } \psi]$	φ before ψ holds and φ must hold
$[\varphi \text{SW } \psi]$	φ when first ψ holds and ψ must hold
$[\varphi \text{WU } \psi]$	φ until ψ holds or φ holds forever
$[\varphi \text{WB } \psi]$	φ before ψ holds or ψ never holds
$[\varphi \text{WW } \psi]$	φ when first ψ holds or ψ never holds
linear time past operators	
PSX φ	φ holds in the previous point and there is a previous point
PWX φ	φ holds in the previous point or no previous point
PG,PF φ	past time G,F
PSU,PSB,PSW	past time SU,SB,SW
PWU,PWB,PWW	past time WU,WB,WW
mu calculus operators	
nu $z. \varphi$	greatest fixpoint wrt. z
mu $z. \varphi$	least fixpoint wrt. z
$\langle \rangle \varphi$	φ holds in one successor state
$[\] \varphi$	φ holds in all successor states
$\langle : \rangle \varphi$	φ holds in one predecessor state
$[\ :] \varphi$	φ holds in all predecessor states