HLL cheatsheet

2024–03–22

Complete specification available on hal-03294999 or hal-03356342, or Prover website.

Meta variables

- a, b range over boolean expressions;
- T, U, V range over types;
- e, t, u range over expressions;
- v, x, y range over variables.

Names

Names stand for variables (may they be streams or terms), functions, constants, types or namespaces.

`foo` refer to the name `foo`.

`bar::foo` refer to the name `foo` in the namespace `bar`.

Types

`bool` Booleans

`int` Integers

`int [ -4, 32]` Integer between -4 and 32.

`int signed 8` 8-bit signed integer.

`int unsigned 8` 8-bit unsigned integers

`tuple {`T_1`, `T_2`}` tuples containing one element of type T_1 and one of type T_2

`struct {`e_1`: `T_1`, `e_2`: `T_2`}` structure with two fields e_1 and e_2 respectively of types T_1 and T_2

T `^ (`t_1`, `t_2`)` multi-dimensional arrays of size $t_1 \times t_2$ of elements of type T

T_1 `*` T_2 `->` U functions with two arguments of types T_1 and T_2 and range U

Type declarations and definitions

`enum {red, blue} colour;` declare type `colour` which contains two values

`sort {flower, grass} < herbs` declare sort `herbs` with two elements

`sort herbs, trees < plants` declare sorts `herbs` and `trees` as sub-sorts of the sort `plants`

T v`;` declare v as an alias for type T

Boolean connectives

Connectives are sorted by decreasing precedence: if $\diamond$ occurs before $\star$, then $a \star b \diamond c$ is $a \star (b \diamond c)$.

`~` a negation, true when a is false

a `&` b conjunction, true if a is true *and* b is true

a `#` b disjunction, true if a is true *or* b is true

a `->` b implication, true if either a is false or b is true. Right associative: a `->` b `->` c is a `->` $(b$ `->` $c)$.

a `<->` b equivalence, true when a is the same as b

a `#!` b exclusive disjunction (a.k.a. xor), true if a isn't the same as b (same precedence as `<->`)

`true` truth, also spelled `True` or `TRUE`

`false` falsity, also spelled `False` or `FALSE`

Integer connectives

Connectives are sorted by decreasing precedence: if $\diamond$ occurs before $\star$, then $a \star b \diamond c$ is $a \star (b \diamond c)$.

a `>` b greater than, true when a is strictly greater than b

a `>=` b greater than or equal to, true when a is when greater or equal to b

a `<` b less than, true when a is strictly lower than b

a `<=` b less than or equal to, true when a is lower or equal to b

a `>>` b right shift

a `<<` b left shift

`-` a negation or additive inverse of a

a `*` b multiplication

a `+` b addition

a `-` b subtraction

a `/` b integer division (fractional part is omitted)

a `/>` b floor division (the greatest integer less than or equal to a divided by b)

a `/<` b ceiling division (the least integer greater than or equal to a divided by b)

a `%` b division remainder of a / b

a `^` b exponentiation

`42` integer literal with the value 42

Temporal operators

`X(`e`)` shift the stream e one cycle forward, read *next*

`pre(`e`)` shift the stream e one cycle backward, read *previous*

`pre(`e`, `t`)` like `pre(`e`)` but takes value t for initialisation

Quantifiers

`ALL` v`:`T `(`a`)` universal quantification, also known as $\forall v \in T, a$

`SOME` v`:`T `(`a`)` existential quantification, also known as $\exists v \in T, a$

`SELECT` v`:`T `(`a`)` the unique element x of domain T, such that expression a, with v substituted by x, becomes true. Undefined, if there is no such a single element. Also known as $\exists! v \in T, a$.

`SUM k:T (e)` also known as $\sum_{k \in T} e$. If T is empty, evaluates to 0.
`PROD k:T (e)` also known as $\prod_{k \in T} e$. If T is empty, evaluates to 1.
`$min v:T (e)` the minimal element among the set $\{e \mid v \in T\}$
`$max v:T (e)` the maximal element among the set $\{e \mid v \in T\}$

Operators

`f(t)` application of function f to expression t
`t = u` expression equality
`t != u` expression inequality
`if a then t else u` conditional
`if a then t1 elif b then t2 else t3` conditional
`(t | p1 => t1 | p2 => t2)` reduces to t_1 if t matches pattern p_1 and to t_2 if t matches p_2; patterns are terms which may contain the wildcard _
`t.2` third element of tuple t
`t.e` access field e of structure t
`t[2]` third element of array t

Sections

All declarations and definitions must occur in a section.

`Inputs:` free streams
`Constants:` constant streams (bool or int)
`Types:` type declarations and definitions
`Declarations:` stream declarations
`Definitions:` definitions of declared streams
`Constraints:` expressions in this section are assumed to hold in all models
`Proof obligations:` for any expression a in this section, if one exhibits a model in which a is false, then model checking fails.
`Outputs:` of the system modelized

Stream declarations and definitions

`v := e;` define variable `v`
`T foo(U, V);` declare function `foo` taking two arguments of types U and V, returning a value of type T
`foo(x, y) := e;` define function `foo`
`T v;` declare a free stream v of type T
`Namespaces: Foo { … }` declare namespace `Foo`

License