Annex F

Formal semantics of concurrent assertions

F.2 Abstract syntax

F.2.1 Abstract grammars

REPLACE

The abstract grammar for unclocked properties is

$P ::= R$	// "sequence" form
$\mid (P)$	// "parenthesis" form
$\mid$ **not** P	// "negation" form
$\mid (P$ **or** $P)$	// "or" form
$\mid (P$ **and** $P)$	// "and" form
$\mid (R$ —-¿ $P)$	// "implication" form

WITH

The abstract grammar for unclocked properties is

$P ::= R$	// "sequence" form
$\mid$**strong**(R)	// "strong sequence" form
$\mid (P)$	// "parenthesis" form
$\mid$ **not** P	// "negation" form
$\mid (P$ **or** $P)$	// "or" form
$\mid (P$ **and** $P)$	// "and" form
$\mid (R$ |-> $P)$	// "implication" form
$\mid <$ **next** $> P$	// "<next>" form
$\mid (P <$ **until** $> P)$	// "<until>" form
$\mid$ **accept_on**$(b)\ P$	// "<accept_on>" form

REPLACE

The abstract grammar for clocked properties is

$Q ::=$ `@(`c`)` P	`// "clock" form`
$\mid S$	`// "sequence" form`
$\mid (Q)$	`// "parenthesis" form`
$\mid$ **not** Q	`// "negation" form`
$\mid$ (Q **or** Q)	`// "or" form`
$\mid$ (Q **and** Q)	`// "and" form`
$\mid$ (S `|->` Q)	`// "implication" form`

WITH

The abstract grammar for clocked properties is

$Q ::=$ `@(`c`)` P	`// "clock" form`
$\mid S$	`// "sequence" form`
$\mid$ **strong**(S)	`// "strong sequence" form`
$\mid (Q)$	`// "parenthesis" form`
$\mid$ **not** Q	`// "negation" form`
$\mid$ (Q **or** Q)	`// "or" form`
$\mid$ (Q **and** Q)	`// "and" form`
$\mid$ (S `|->` Q)	`// "implication" form`
$\mid <$ **next** $> Q$	`// "<next>" form`
$\mid$ ($Q <$ **until** $>$ Q)	`// "<until>" form`
$\mid$ **accept_on**(b) Q	`// "<accept_on>" form`

F.2.3 Derived forms

REPLACE

Internal parentheses are omitted in compositions of the (associative) operators `##1` and `or` .

F.2.3.1 Derived nonoverlapping implication operator

— $(R_1$ `|=>` $P) \equiv ((R_1$ `##1 1)` `|->` $P)$.

— $(S_1$ `|=>` $Q) \equiv ((S_1$ `##1 @(1) 1)` `|->` $Q)$.

F.2.3.2 Derived consecutive repetition operators

— Let $m > 0$. $R[*m] \equiv (R$ `##1` R `##1` $\ldots$ `##1` $R)$ `//`m copies of R.

— $R[*0 : \$] \equiv (R[*0]$ **or** $R[1 : \$])$.

— Let $m \leq n$. $R[*m : n] \equiv R[*m]$ **or** $R[*m + 1]$ **or** $\ldots$ **or** $R[*n])$.

— Let $m > 1$. $R[*m : \$] \equiv (R[*m-1]$ `##1` $R[*1 : \$])$.

F.2.3.3 Derived delay and concatenation operators

Let $m < n$.

— $(\texttt{\#\#[}m:n\texttt{]}\ R) \equiv (\texttt{1[}*m:n\texttt{]}\ \ \texttt{\#\#1}\ \ R)$.

— $(\texttt{\#\#[}m:\$\texttt{]}\ R) \equiv (\texttt{1[}*m:\$\texttt{]}\ \ \texttt{\#\#1}\ \ R)$.

— $(\texttt{\#\#}m\ R) \equiv (\texttt{1[}*m\texttt{]}\ \ \texttt{\#\#1}\ \ R)$.

— Let $m > 0$. $(R_1\texttt{\#\#[}m:n\texttt{]}\ R_2) \equiv (R_1\texttt{\#\#1}\ \ \texttt{1[}m-1:n-1\texttt{]}\ \ \texttt{\#\#1}\ R_2)$.

— Let $m > 0$. $(R_1\texttt{\#\#[}m:\$\texttt{]}\ R_2) \equiv (R_1\texttt{\#\#1}\ \ \texttt{1[}m-1:\$\texttt{]}\ \ \texttt{\#\#1}\ R_2)$.

— Let $m > 1$. $(R_1\texttt{\#\#}m\ R_2) \equiv (R_1\texttt{\#\#1}\ \ \texttt{1[}m-1\texttt{]}\ \ \texttt{\#\#1}\ R_2)$.

— $(R_1\ \ \texttt{\#\#[0:0]}\ R_2) \equiv (R_1\ \ \texttt{\#\#0}\ \ R_2)$.

— Let $n > 0$. $(R_1\texttt{\#\#[0:}n\texttt{]}\ R_2) \equiv ((R_1\ \ \texttt{\#\#0}\ \ R_2)\ \textbf{or}\ (R1\ \ \texttt{\#\#[1:}n\texttt{]}\ \ R_2))$.

— $(R_1\texttt{\#\#[0:\$]}\ \ R_2) \equiv ((R_1\ \ \texttt{\#\#0}\ \ R_2)\ \textbf{or}\ (R1\ \ \texttt{\#\#[1:\$]}\ \ R_2))$.

F.2.3.4 Derived nonconsecutive repetition operators

Let $m \leq n$.

— $b[\texttt{->}m:n] \equiv (!b[*0:\$]\ \ \texttt{\#\#1}\ \ b)[*m:n])$.

— $b[\texttt{->}m:\$] \equiv (!b[*0:\$]\ \ \texttt{\#\#1}\ \ b)[*m:\$])$.

— $b[\texttt{->}m] \equiv (!b[*0:\$]\ \ \texttt{\#\#1}\ \ b)[*m])$.

— $b[=m:n] \equiv (b[\texttt{->}m:n]\ \ \texttt{\#\#1}\ \ !b[*0:\$])$.

— $b[=m:\$] \equiv (b[\texttt{->}m:\$]\ \ \texttt{\#\#1}\ \ !b[*0:\$])$.

— $b[=m] \equiv (b[\texttt{->}m]\ \ \texttt{\#\#1}\ \ !b[*0:\$])$.

F.2.3.5 Other derived operators

— $(R_1\ \textbf{and}\ R_2) \equiv (((R_1\ \ \texttt{\#\#1}\ \ \texttt{1[*0:\$]})\ \textbf{intersect}\ R_2)\ \textbf{or}\ (R_1\ \textbf{intersect}\ (R_2\ \ \texttt{\#\#1}\ \ \texttt{1[*0:\$]})))$.

— $(R_1\ \textbf{within}\ R_2) \equiv ((\texttt{1[*0:\$]}\ \ \texttt{\#\#1}\ \ R_1\ \ \texttt{\#\#1}\ \ \texttt{1[*0:\$]})\ \textbf{intersect}\ R_2)$.

— $(b\ \textbf{throughout}\ R) \equiv ((b\texttt{1[*0:\$]})\ \ \textbf{intersect}\ \ R)$.

— $(R, v = e) \equiv (R\ \ \texttt{\#\#0}\ (1, v = e))$.

— $(R, v_1 = e_1, \ldots, v_k = e_k) \equiv ((R, v_1 = e1)\ \ \texttt{\#\#0}\ (1, v_2 = e_2, \ldots, v_k = e_k))$ for $k > 1$.

— $(\textbf{if}(b)\ P) \equiv (b\ \ \texttt{|->}\ \ P)$.

— $(\textbf{if}(b)\ P_1\ \textbf{else}\ P_2) \equiv (b\ \ \texttt{|->}\ \ P_1)\ \textbf{and}\ (!b\ \ \texttt{|->}\ \ P_1))$.

WITH

Internal parentheses are omitted in compositions of the (associative) operators `##1` and `or`.

~~**F.2.3.1 Derived nonoverlapping implication operator**~~

— ~~$(R_1 \mid\Rightarrow P) \equiv ((R_1 \#\#11) \mid\rightarrow P)$.~~

— ~~$(S_1 \mid\Rightarrow Q) \equiv ((S_1 \#\#1@(1)1) \mid\rightarrow Q)$.~~

F.2.3.1 Derived sequence operators

F.2.3.21.1 Derived consecutive repetition operators

— ~~Let $m > 0$. $R[*m] \equiv (R \#\#1 R \#\#1 \ldots \#\#1 + R)$ // m copies of R.~~

— Let $m > 0$. $R[*m] \equiv (R[*m - 1] \ \#\#1 \ R)$.

— $R[*0 : \$] \equiv (R[*0]$ **or** $R[1 : \$])$.

— ~~Let $m \leq n$. $R[*m : n] \equiv R[*m]$ **or** $R[*m + 1]$ **or** $\ldots$ **or** $R[*n])$.~~

— $R[*m : m] \equiv R[*m]$.

— Let $m < n$. $R[*m : n] \equiv (R[*m : n - 1]$ **or** $R[*n])$.

— Let $m > 1$. $R[*m : \$] \equiv (R[*m{-}1] \ \texttt{\#\#1} \ R[*1 : \$])$.

F.2.3.31.2 Derived delay and concatenation operators

Let $m < n$.

— $(\texttt{\#\#[}m : n\texttt{]} \ R) \equiv (\texttt{1[} * m : n\texttt{]} \ \texttt{\#\#1} \ R)$.

— $(\texttt{\#\#[}m : \$\texttt{]} \ R) \equiv (\texttt{1[} * m : \$\texttt{]} \ \texttt{\#\#1} \ R)$.

— $(\texttt{\#\#}m \ R) \equiv (\texttt{1[} * m\texttt{]} \ \texttt{\#\#1} \ R)$.

— Let $m > 0$. $(R_1\texttt{\#\#[}m : n\texttt{]} \ R_2) \equiv (R_1\texttt{\#\#1} \ \texttt{1[}m - 1 : n - 1\texttt{]} \ \texttt{\#\#1} \ R_2)$.

— Let $m > 0$. $(R_1\texttt{\#\#[}m : \$\texttt{]} \ R_2) \equiv (R_1\texttt{\#\#1} \ \texttt{1[}m - 1 : \$\texttt{]} \ \texttt{\#\#1} \ R_2)$.

— Let $m > 1$. $(R_1\texttt{\#\#}m \ R_2) \equiv (R_1\texttt{\#\#1} \ \texttt{1[}m - 1\texttt{]} \ \texttt{\#\#1} \ R_2)$.

— $(R_1 \ \texttt{\#\#[0:0]} \ R_2) \equiv (R_1 \ \texttt{\#\#0} \ R_2)$.

— Let $n > 0$. $(R_1\texttt{\#\#[0:}n\texttt{]} \ R_2) \equiv ((R_1 \ \texttt{\#\#0} \ R_2)$ **or** $(R1 \ \texttt{\#\#[1:}n\texttt{]} \ R_2))$.

— $(R_1\texttt{\#\#[0:\$]} \ R_2) \equiv ((R_1 \ \texttt{\#\#0} \ R_2)$ **or** $(R1 \ \texttt{\#\#[1:\$]} \ R_2))$.

F.2.3.41.3 Derived nonconsecutive repetition operators

Let $m \leq n$.

— $b[{-}{>}m : n] \equiv (!b[*0 : \$] \ \texttt{\#\#1} \ b)[*m : n])$.

— $b[{-}{>}m : \$] \equiv (!b[*0 : \$] \ \texttt{\#\#1} \ b)[*m : \$])$.

— $b[{-}{>}m] \equiv (!b[*0 : \$] \ \texttt{\#\#1} \ b)[*m])$.

— $b[= m : n] \equiv (b[{-}{>}m : n] \ \texttt{\#\#1} \ !b[*0 : \$])$.

— $b[= m : \$] \equiv (b[{-}{>}m : \$] \ \texttt{\#\#1} \ !b[*0 : \$])$.

— $b[= m] \equiv (b[{-}{>}m] \ \texttt{\#\#1} \ !b[*0 : \$])$.

F.2.3.~~5~~1.4 Other derived operators

— $(R_1$ **and** $R_2) \equiv (((R_1$ `##1 1[*0:$]`$)$ **intersect** $R_2)$ **or** $(R_1$ **intersect** $(R_2$ `##1 1[*0:$]`$)))$.

— $(R_1$ **within** $R_2) \equiv ((1$`[*0:$]` `##1` R_1 `##1 1[*0:$]`$)$ **intersect** $R_2)$.

— $(b$ **throughout** $R) \equiv ((b1$`[*0:$]`$)$ **intersect** $R)$.

— $(R, v = e) \equiv (R$ `##0` $(1, v = e))$.

— $(R, v_1 = e_1, \ldots, v_k = e_k) \equiv ((R, v_1 = e1)$ `##0` $(1, v_2 = e_2, \ldots, v_k = e_k))$ for $k > 1$.

— ~~(**if**(b) $P) \equiv (b|\!-\!>$ $P)$.~~

— ~~(**if**(b) P_1 **else** $P_2) \equiv ((b|\!-\!>$ $P_1)$ **and** $(!b|\!-\!>$ $P_1))$.~~

F.2.3.2 Derived property operators

F.2.3.2.1 Derived boolean operators

— P_1 **implies** $P_2 \equiv$ (**not** P_1 **or** P_2).

— P_1 **iff** $P_2 \equiv ((P_1$ `->` $P_2)$ **and** $(P_2$ `->` $P_1))$.

F.2.3.2.2 Derived nonoverlapping implication operator

— $(R$ `|=>` $P) \equiv ((R$ `##1 1)` `|->` $P)$.

— $(S$ `|=>` $Q) \equiv ((S$ `##1 @(1) 1)` `|->` $Q)$.

F.2.3.2.3 Derived conditional operators

— **if**(b) $P \equiv (b$ `|->` $P)$.

— **if**(b) P_1 **else** $P_2) \equiv ((b$ `|->` $P_1)$ **and** $(!b$ `|->` $P_1))$.

F.2.3.2.4 Derived followed_by operators

— $(r$ `#-#` $p) \equiv$ (**not**$(r$ `|->` **not** $p))$.

— $(r$ `#=#` $p) \equiv$ (**not**$(r$ `|=>` **not** $p))$.

F.2.3.2.5 Derived reset operator

— (**reject_on**(b) $p) \equiv$ (**not**(**accept_on**(b) **not** $p))$.

F.2.3.2.6 Derived unbounded temporal operators

— (**eventually** $p) \equiv (1 <$ **until** $> p)$.

— (**always** $p) \equiv$ (**not** $(1 <$ **until** $>$ **not** $p))$.

— $(p$ **until** $q) \equiv ((p <$ **until** $> q)$ **or always** $p)$.

— $(p$ **until with** $q) \equiv ((p$ **until** $(p$ **and** $q))$.

— $(p <$ **until with** $> q) \equiv ((p <$ **until** $> (p$ **and** $q))$.

F.2.3.2.7 Derived bounded temporal operators

— $(\textbf{next}\ p)\ \equiv\ (\textbf{not}\ <\textbf{next}>\ \textbf{not}\ p)$.

— $(<\textbf{next}>[1]\ p)\ \equiv\ <\textbf{next}>p$.

— Let $m > 1$. $(<\textbf{next}>[m]\ p)\ \equiv\ (<\textbf{next}>(<\textbf{next}>[n-1]\ p))$.

— Let $m > 1$. $(\textbf{next}[m]\ p)\ \equiv\ (\textbf{not}\ <\textbf{next}>[m]\ \textbf{not}\ p)$.

— $(<\textbf{next}>[0:\$]\ p)\ \equiv\ (\textbf{eventually}\ p)$.

— Let $m \geq 0$. $(<\textbf{next}>[m:m]\ p)\ \equiv\ (<\textbf{next}>[m]\ p)$;

— Let $m < n$. $(<\textbf{next}>[m:n]\ p)\ \equiv\ (<\textbf{next}>[m:n-1]\ \textbf{or}\ <\textbf{next}>[n]\ p)$;

— Let $m > 0$. $(<\textbf{next}>[m:\$]\ p)\ \equiv\ (<\textbf{next}>[m]\ \textbf{eventually}\ p)$.

— Let $m \geq 0$. $(<\textbf{always}>[m:m]\ p)\ \equiv\ (<\textbf{next}>[m]\ p)$;

— Let $m < n$. $(<\textbf{always}>[m:n]\ p)\ \equiv\ (<\textbf{always}>[m:n-1]\ \textbf{and}\ <\textbf{next}>[n]\ p)$;

— Let $m \geq 0$. $(\textbf{always}[m:\$]\ p\ \equiv\ (\textbf{next}[m]\ \textbf{always}\ p)$

— Let $m < n$. $(\textbf{next}[m:n]\ p)\ \equiv\ (\textbf{not}\ <\textbf{always}>[m:n]\ \textbf{not}\ p)$.

— Let $m < n$. $(\textbf{always}[m:n]\ p)\ \equiv\ (\textbf{not}\ <\textbf{next}>[m:n]\ \textbf{not}\ p)$.

F.3 Semantics

F.3.1 Rewrite rules for clocks

REPLACE

The semantics of clocked sequences and properties is defined in terms of the semantics of unclocked sequences and properties. The following rewrite rules define the transformation of a clocked sequence or property into an unclocked version that is equivalent for the purposes of defining the satisfaction relation. In this transformation, it is required that the conditions in event controls not be dependent upon any local variables.

— @(c) $b \longmapsto$ (!c[*0:$] ##1 c & b).

— @(c) (1, $v = e$) $\longmapsto$ (@(c)1 ##0 (1, $v = e$)).

— @(c) (P) $\longmapsto$ (@(c)P).

— @(c) (R_1 ##1 R_2) $\longmapsto$ (@(c)R_1 ##1 @(c)R_2).

— @(c) (R_1 ##0 R_2) $\longmapsto$ (@(c)R_1 ##0 @(c)R_2).

— @(c) (R_1 or R_2) $\longmapsto$ (@(c)R_1 or @(c)R_2).

— @(c) (R_1 intersect R_2) $\longmapsto$ (@(c)R_1 intersect @(c)R_2).

— @(c) first_match(R) $\longmapsto$ (first_match(@(c)R)).

— @(c) R[*0] $\longmapsto$ (@(c)R)[*0]).

— @(c) R[*1:$] $\longmapsto$ (@(c)R)[*1:$]).

— @(c) disable iff(b) P $\longmapsto$ disable iff(b) (@(c)P).

— @(c) not P $\longmapsto$ not @(c)P.

— @(c) (P_1 —¿ P_2) $\longmapsto$ (@(c)P_1 —¿ @(c)P_2).

— @(c) (P_1 or P_2) $\longmapsto$ (@(c)P_1 or @(c)P_2).

— @(c) (P_1 and P_2) $\longmapsto$ (@(c)P_1 and @(c)P_2).

WITH

The semantics of clocked sequences and properties is defined in terms of the semantics of unclocked sequences and properties. The following rewrite rules define the transformation of a clocked sequence or property into an unclocked version that is equivalent for the purposes of defining the satisfaction relation. In this transformation, it is required that the conditions in event controls not be dependent upon any local variables.

— ~~@(e)b $\longmapsto$ (!e[$*0 : $]##1$e$&$b$).~~

— ~~@(e) (1, $v = e$) $\longmapsto$ (@(e)1 ##0 (1, $v = e$)).~~

— ~~@(e) (P) $\longmapsto$ (@(e)P).~~

— ~~@(e) (R_1 ##1 R_2) $\longmapsto$ (@(e)R_1 ##1 @(e)R_2).~~

— ~~@(e) (R_1 ##0 R_2) $\longmapsto$ (@(e)R_1 ##0 @(e)R_2).~~

— ~~@(e) (R_1 or R_2) $\longmapsto$ (@(e)R_1 or @(e)R_2).~~

— ~~@(e) (R_1 intersect R_2) $\longmapsto$ (@(e)R_1 intersect @(e)R_2).~~

— ~~@(e) first_match(R) $\longmapsto$ (first_match(@(e)R)).~~

— ~~@(e) R[*0] $\longmapsto$ (@(e)R)[*0]).~~

— ~~@(e) R[*1:$] $\longmapsto$ (@(e)R)[*1:$]).~~

— ~~@(e) disable iff(b) P $\longmapsto$ disable iff(b) (@(e)P).~~

— ~~@(e) not P $\longmapsto$ not @(e)P.~~

— ~~@(e) (P_1 |–> P_2) $\longmapsto$ (@(e)P_1 |–> @(e)P_2).~~

— ~~@(e) (P_1 or P_2) $\longmapsto$ (@(e)P_1 or @(e)P_2).~~

— ~~@(e) (P_1 and P_2) $\longmapsto$ (@(e)P_1 and @(e)P_2).~~

F.3.1.1 Rewrite rules for sequences

The transformation $\mathcal{T}^s$ recursively defined below produces an unclocked sequence R equivalent to a given clocked sequence S:

— $\mathcal{T}^s(@\,(c)\ b) = (!c[\star 0:\$]\ \#\#1\ c\ \&\ b)$.

— $\mathcal{T}^s(@\,(c)\ (1,\ v = e)) = (\mathcal{T}^s(@\,(c)\ 1)\ \#\#0\ (1,\ v = e))$.

— $\mathcal{T}^s(@\,(c_1)\ (@\,(c_2)\ S)) = (\mathcal{T}^s(@\,(c_2)\ S))$.

— $\mathcal{T}^s(@\,(c)\ (S_1\ \#\#1\ S_2)) = (\mathcal{T}^s(@\,(c)\ S_1)\ \#\#1\ \mathcal{T}^s(@\,(c)\ S_2))$.

— $\mathcal{T}^s(@\,(c)\ (S_1\ \#\#0\ S_2)) = (\mathcal{T}^s(@\,(c)\ S_1)\ \#\#0\ \mathcal{T}^s(@\,(c)\ S_2))$.

— $\mathcal{T}^s(@\,(c)\ (S_1\ \mathbf{or}\ S_2)) = (\mathcal{T}^s(@\,(c)\ S_1)\ \mathbf{or}\ \mathcal{T}^s(@\,(c)\ S_2))$.

— $\mathcal{T}^s(@\,(c)\ (S_1\ \mathbf{intersect}\ S_2)) = (\mathcal{T}^s(@\,(c)\ S_1)\ \mathbf{intersect}\ \mathcal{T}^s(@\,(c)\ S_2))$.

— $\mathcal{T}^s(@\,(c)\ (\mathbf{first\_match}\ S)) = (\mathbf{first\_match}\ \mathcal{T}^s(@\,(c)\ S))$.

— $\mathcal{T}^s(@\,(c)\ (S[\star 0])) = ((\mathcal{T}^s(@\,(c)\ S))[\star 0])$.

— $\mathcal{T}^s(@\,(c)\ (S[\star 1:\$])) = ((\mathcal{T}^s(@\,(c)\ S))[\star 1:\$])$.

F.3.1.2 Rewrite rules for properties

The transformation $\mathcal{T}^p$ recursively defined below produces an unclocked property P equivalent to a given clocked property Q:

— $\mathcal{T}^p(S) = \mathcal{T}^s(S)$.

— $\mathcal{T}^p(\mathbf{strong}(S)) = (\mathbf{strong}(\mathcal{T}^s(S)))$.

— $\mathcal{T}^p(@\,(c_1)\ (@\,(c_2)\ P)) = \mathcal{T}^p(@\,(c_2)\ P)$.

— $\mathcal{T}^p(@\,(c_1)\ (@\,(c_2)\ Q)) = \mathcal{T}^p(@\,(c_2)\ Q)$.

— $\mathcal{T}^p(@\,(c)\ \mathbf{disable\ iff}(b)\ Q) = (\mathbf{disable\ iff}(b)\ \mathcal{T}^p(@\,(c)\ Q))$.

— $\mathcal{T}^p(@\,(c)\ \mathbf{accept\_on}(b)\ Q) = (\mathbf{accept\_on}(b)\ \mathcal{T}^p(@\,(c)\ Q))$.

— $\mathcal{T}^p(@\,(c)\ \mathbf{not}\ Q) = (\mathbf{not}\ \mathcal{T}^p(@\,(c)\ Q))$.

— $\mathcal{T}^p(@\,(c)\ (S\ \mid\!\!\rightarrow\ Q) = (\mathcal{T}^s(@\,(c)\ S)\ \mid\!\!\rightarrow\ \mathcal{T}^p(@\,(c)\ Q))$.

— $\mathcal{T}^p(@\,(c)\ (Q_1\ \mathbf{or}\ Q_2)) = (\mathcal{T}^p(@\,(c)\ Q_1)\ \mathbf{or}\ \mathcal{T}^p(@\,(c)\ Q_2))$.

— $\mathcal{T}^p(@\,(c)\ (Q_1\ \mathbf{and}\ Q_2)) = (\mathcal{T}^p(@\,(c)\ Q_1)\ \mathbf{and}\ \mathcal{T}^p(@\,(c)\ Q_2))$.

— $\mathcal{T}^p(@\,(c)\ <\mathbf{next}>\ Q) = (!c <\mathbf{until}> (c\ \mathbf{and} <\mathbf{next}> (!c <\mathbf{until}> (c\ \mathbf{and}\ \mathcal{T}^p(Q)))))$.

— $\mathcal{T}^p(@\,(c)\ (Q_1 <\mathbf{until}> Q_2) = ((\mathbf{not}(c\ \mathbf{and\ not}\ \mathcal{T}^p(Q_1)) <\mathbf{until}> (c\ \mathbf{and}\ \mathcal{T}^p(Q_2)))$.

F.3.3 Satisfaction without local variables

REPLACE

F.3.3.1 Neutral satisfaction

Neutral satisfaction of properties is defined as follows:

— $w \models (P)$ iff $w \models P$.

— $w \models Q$ iff $w \models Q'$, where Q' is the unclocked property that results from Q by applying the rewrite rules.

— $w \models$ `not` P iff $\overline{w} \not\models P$.

— $w \models R$ iff there exists $0 \leq j < |w|$ so that $w^{0,j} \models\!\models R$.

— $w \models (R$ `|->` $P)$ iff for every $0 \leq j < |w|$ so that $\overline{w}^{0,j} \models\!\models R, w^{j..} \models P$.

— $w \models (P_1$ `or` $P_2)$ iff $w \models P_1$ or $w \models P_2$.

— $w \models (P_1$ `and` $P_2)$ iff $w \models P_1$ and $w \models P_2$.

Remark: Because w is nonempty, it can be proved that $w \models$ `not` b iff $w \models\!\models\! !b$.

WITH

Neutral satisfaction of properties is defined as follows:

— $w \models (P)$ iff $w \models P$.

— $w \models Q$ iff $w \models$ ~~Q'~~ $\mathcal{T}^p(Q)$ ~~, where Q' is the unclocked property that results from Q by applying the rewrite rules~~.

— $w \models$ `not` P iff $\overline{w} \not\models P$.

— ~~$w \models R$ iff there exists $0 \leq j < |w|$ so that $w^{0,j} \models\!\models R$.~~

— $w \models$ `strong` (R) iff there exists $0 \leq j < |w|$ so that $w^{0,j} \models\!\models R$.

— $w \models R$ iff for every $0 \leq j < |w|, w^{0,j}\top^\omega \models$ `strong` (R).

— $w \models (R$ `|->` $P)$ iff for every $0 \leq j < |w|$ so that $\overline{w}^{0,j} \models\!\models R, w^{j..} \models P$.

— $w \models (P_1$ `or` $P_2)$ iff $w \models P_1$ and $w \models P_2$.

— $w \models (P_1$ `and` $P_2)$ iff $w \models P_1$ and $w \models P_2$.

— $w \models$ (`<next>` $P)$ iff $|w| > 1$ and $w^{1..} \models P$.

— $w \models (P_1$ `<until>` $P_2)$ iff there exists $0 \leq j < |w|$ so that $w^{k..} \models P_2$ and for every $0 \leq i < j, w^{i..} \models P_1$.

— $w \models$ (`accept_on` $(b)P)$ iff either

  — $w \models P$ and no letter of w satisfies b, or

  — Some letter of w satisfies b and $w^{0,i-1}\top^\omega \models P$ for i the least index such that $w^i \models b, 0 \leq i < |w|$. Here, $w^{0,-1}$ denotes the empty word.

Remark: Because w is nonempty, it can be proved that $w \models$ `not` b iff $w \models\!\models\! !b$.

F.3.3.3 Vacuity

REPLACE

This subclause defines the relation of non-vacuity, denoted $\models^{\text{non}}$, between a word w and a property P. An evaluation of P on w is non-vacuous provided $w \models^{\text{non}} P$.

— Base:

— Let R be a sequence. Then $w \models^{\text{non}} R$.

— Induction:

— $w \models^{\text{non}} (P_1)$ iff $w \models^{\text{non}} P_1$.
— $w \models^{\text{non}} R \ \text{|->} \ P_1$ iff there exists $i \geq 0$ such that $w^{0..i} \models R$ and $w^{i..} \models^{\text{non}} P_1$.
— $w \models^{\text{non}} P_1$ **and** P_2 iff $w \models^{\text{non}} P_1$ or $w \models^{\text{non}} P_2$.
— $w \models^{\text{non}} P_1$ **or** P_2 iff $w \models^{\text{non}} P_1$ or $w \models^{\text{non}} P_2$.
— $w \models^{\text{non}} \textbf{not} P_1$ iff $w \models^{\text{non}} P_1$.
— $w, L \models^{\text{non}} \textbf{disable iff}(b) P_1$ iff $w \models^{\text{non}} P_1$ and one of the following holds:

1. For every $0 \leq i < |w|, w^i \not\models b$.
2. For some prefix $x \leq w$, we have that for every $0 \leq i < |x|, x^i \not\models b$,and either $x\bot^\omega \models P$ or $x\top^\omega \not\models P$.

A word w satisfies property P non-vacuously iff $w \models P$ and $w \models^{\text{non}} P$.

WITH

This subclause defines the relation of non-vacuity, denoted $\models^{\text{non}}$, between a word w and a property P. An evaluation of P on w is non-vacuous provided $w \models^{\text{non}} P$.

— Base:

— ~~Let R be a sequence. Then~~ $w \models^{\text{non}} R$.
— $w \models^{\text{non}} \textbf{strong}(R)$.

— Induction:

— $w \models^{\text{non}} (P_{\text{T}})$ iff $w \models^{\text{non}} P_{\text{T}}$.
— $w \models^{\text{non}} R \ \text{|->} \ P_{\text{T}}$ iff there exists $i \geq 0$ such that $w^{0..i} \models R$ and $w^{i..} \models^{\text{non}} P_{\text{T}}$.
— $w \models^{\text{non}} P_1$ **and** P_2 iff $w \models^{\text{non}} P_1$ or $w \models^{\text{non}} P_2$.
— $w \models^{\text{non}} P_1$ **or** P_2 iff $w \models^{\text{non}} P_1$ or $w \models^{\text{non}} P_2$.
— $w \models^{\text{non}} \textbf{not} P_{\text{T}}$ iff $w \models^{\text{non}} P_{\text{T}}$.
— $w \models^{\text{non}} < \textbf{next} > P$ iff $w^{1..} \models^{\text{non}} P$.
— $w \models^{\text{non}} P_1 < \textbf{until} > P_2$ iff there exists $0 \leq k < |w|$ such that $w^{k..} \models P_2$, $w^k \models^{\text{non}} P_2$ and for every $i < k$ $w^{i..} \models P_1$ and there exists $j < k$ such that $w^{j..} \models^{\text{non}} P_1$.
— w ~~, L~~ $\models^{\text{non}} \textbf{disable iff}(b) P_{\text{T}}$ iff $w \models^{\text{non}} P_{\text{T}}$ and one of the following holds:

1. For every $0 \leq i < |w|, w^i \nvdash b$.
2. For some prefix $x \leq$ of w, we have that for every $0 \leq i < |x|, x^i \nvdash b$,and either $x\bot^\omega \models P$ or $x\top^\omega \nvdash P$.

— $w \models^{\text{non}} \textbf{accept\_on}(b)P$ iff $w \models^{\text{non}} P$.

A word w satisfies property P non-vacuously iff $w \models P$ and $w \models^{\text{non}} P$.

F.3.6 Satisfaction with local variables

F.3.6.1 Neutral satisfaction

REPLACE

Neutral satisfaction of properties is defined as follows:

— $w \models Q$ iff $w, \{\} \models Q$.

— $w, L_0 \models Q$ iff $w, L_0 \models Q'$, where Q' is the unclocked property that results from Q by applying the rewrite rules.

— $w, L_0 \models \textbf{not}\ P$ iff $\overline{w}, L_0 \nvDash P$.

— $w, L_0 \models R$ iff there exist $0 \leq j < |w|$ and L_1 so that $w^{0,j}, L_0, L_1 \models\!\models R$.

— $w, L_0 \models (R \ \texttt{|->}\ P)$ iff for every $0 \leq j < |w|$ and L_1 so that $\overline{w}^{0,j}, L_0, L_1 \models\!\models R, w^{j\cdot\cdot}, L_1 \models P$.

— $w, L_0 \models (P)$ iff $w, L_0 \models P$.

— $w, L_0 \models (P_1\ \textbf{or}\ P_2)$ iff $w, L_0 \models P_1$ or $w, L_0 \models P_2$.

— $w, L_0 \models (P_1\ \textbf{and}\ P_2)$ iff $w, L_0 \models P_1$ and $w, L_0 \models P_2$.

WITH

Neutral satisfaction of properties is defined as follows:

— $w \models Q$ iff $w, \{\} \models Q$.

— $w, L_0 \models Q$ iff $w, L_0 \models ~~Q'~~\ \mathcal{T}^p(Q)$ ~~, where Q' is the unclocked property that results from Q by applying the rewrite rules~~.

— $w, L_0 \models \textbf{not}\ P$ iff $\overline{w}, L_0 \nvDash P$.

— ~~$w, L_0 \models R$ iff there exist $0 \leq j < |w|$ and L_1 so that $w^{0,j}, L_0, L_1 \models\!\models R$.~~

— $w, L_0 \models$ `strong` (R) iff there exist $0 \leq j < |w|$ and L_1 so that $w^{0,j}, L_0, L_1 \models\!\models R$.

— $w, L_0 \models R$ iff for every $0 \leq j < |w|, w^{0,j}\top^\omega, L_0 \models$ `strong` (R).

— $w, L_0 \models (R \ \texttt{|->}\ P)$ iff for every $0 \leq j < |w|$ and L_1 so that $\overline{w}^{0,j}, L_0, L_1 \models\!\models R, w^{j\cdot\cdot}, L_1 \models P$.

— $w, L_0 \models (P)$ iff $w, L_0 \models P$.

— $w, L_0 \models (P_1$ **or** $P_2)$ iff $w, L_0 \models P_1$ or $w, L_0 \models P_2$.

— $w, L_0 \models (P_1$ **and** $P_2)$ iff $w, L_0 \models P_1$ and $w, L_0 \models P_2$.

— $w, L_0 \models$ (`<next>` $P)$ iff $|w| > 1$ and $w^{1\cdot\cdot}, L_0 \models P$.

— $w, L_0 \models (P_1$ `<until>` $P_2)$ iff there exists $0 \leq j < |w|$ so that $w^{k\cdot\cdot}, L_0 \models P_2$ and for every $0 \leq i < j, w^{i\cdot\cdot}, L_0 \models P_1$.

— $w, L_0 \models$ (`accept_on` $(b)P)$ iff either

  — $w, L_0 \models P$ and no letter of w satisfies b, or

  — Some letter of w satisfies b and $w^{0,i-1}\top^\omega, L_0 \models P$ for i the least index such that $w^i, L_0 \models b, 0 \leq i < |w|$. Here, $w^{0,-1}$ denotes the empty word.

F.5 Recursive properties

REPLACE

— RESTRICTION 1: The negation operator `not` cannot be applied to any property expression that instantiates a property from which a recursive property can be reached in the dependency digraph.

WITH

— RESTRICTION 1: ~~The negation operator~~ The operators `not` and `until` cannot be applied to any property expression that instantiates a property from which a recursive property can be reached in the dependency digraph.