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`
| (P) `// "parenthesis" form`
| **not** P `// "negation" form`
| (P **or** P) `// "or" form`
| (P **and** P) `// "and" form`
| (R `|->` P) `// "implication" form`

WITH

The abstract grammar for unclocked properties is

$P ::=$ ~~R `// "sequence" form`~~ **strong** (R) `// "strong sequence" form`
| **weak** (R) `// "weak sequence" form`
| (P) `// "parenthesis" form`
| **not** P `// "negation" form`
| (P **or** P) `// "or" form`
| (P **and** P) `// "and" form`
| (R `|->` P) `// "implication" form`
| **nexttime** P `// "nexttime" form`
| (P **until** P) `// "until" form`
| **accept_on** (b) P `// "accept_on" form` // if 1757 pass

REPLACE

The abstract grammar for clocked properties is

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

WITH

The abstract grammar for clocked properties is

```
Q ::= @(c) P                          // "clock" form
    ~~| S // "sequence" form~~
    | strong(S)                       // "strong sequence" form
    | weak(S)                         // "weak sequence" form
    | (Q)                             // "parenthesis" form
    | not Q                           // "negation" form
    | ( Q or Q )                      // "or" form
    | ( Q and Q )                     // "and" form
    | ( S |-> Q )                     // "implication" form
    | nexttime Q                      // "nexttime" form
    | ( Q until Q )                   // "until" form
    | accept_on(b) Q                  // "accept_on" form// if 1757 pass
```

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

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

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

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

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

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

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

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

WITH

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

~~F.2.3.1 Derived nonoverlapping implication operator~~

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

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

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 $m \le n$. $R[*m:n] \equiv R[*m]$ **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 $m > 1$. $R[*m:\$] \equiv (R[*m{-}1]$ ##1 $R[*1:\$])$.

F.2.3.31.2 Derived delay and concatenation operators

Let $m < n$.

— (##\nlb $m:n$\nrb R) $\equiv$ (1\nlb $*m:n$\nrb ##1 R).

— (##\nlb $m:\$$\nrb R) $\equiv$ (1\nlb $*m:\$$\nrb ##1 R).

— (##m R) $\equiv$ (1\nlb $*m$\nrb ##1 R).

— Let $m > 0$. (R_1##\nlb $m:n$\nrb R_2) $\equiv$ (R_1##1 1\nlb $m-1:n-1$\nrb ##1 R_2).

— Let $m > 0$. (R_1##\nlb $m:\$$\nrb R_2) $\equiv$ (R_1##1 1\nlb $m-1:\$$\nrb ##1 R_2).

— Let $m > 1$. (R_1##m R_2) $\equiv$ (R_1##1 1\nlb $m-1$\nrb ##1 R_2).

— (R_1 ##\nlb 0:0\nrb R_2) $\equiv$ (R_1 ##0 R_2).

— Let $n > 0$. (R_1##\nlb 0:n\nrb R_2) $\equiv$ ((R_1 ##0 R_2) **or** ($R1$ ##\nlb 1:n\nrb R_2)).

— (R_1##\nlb 0:$\$$\nrb R_2) $\equiv$ ((R_1 ##0 R_2) **or** ($R1$ ##\nlb 1:$\$$\nrb R_2)).

F.2.3.41.3 Derived nonconsecutive repetition operators

Let $m \le n$.

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

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

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

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

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

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

F.2.3.51.4 Other derived operators

— $(R_1$ **and** $R_2) \equiv (((R_1$ `##1 1\nlb *0:$\nrb`) **intersect** $R_2)$ **or** $(R_1$ **intersect** $(R_2$ `##1 1\nlb *0:$

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

— $(b$ **throughout** $R) \equiv ((b$`\nlb *0:$\nrb`) **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 sequential property

— $R \equiv$ **strong**(R) when used in a **cover property** or **expect** statement. $R \equiv$ **weak**(R) when used in an **assert property** or **assume property** statement.

F.2.3.2.2 Derived boolean operators

— p_1 **implies** $p_2 \equiv ($**not** p_1 **or** $p_2)$.

— p_1 **iff** $p_2 \equiv ((p_1$ **implies** $p_2)$ **and** $(p_2$ **implies** $p_1))$.

F.2.3.2.3 Derived nonoverlapping implication operator

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

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

F.2.3.2.4 Derived conditional operators

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

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

F.2.3.2.5 Derived followed_by operators

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

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

F.2.3.2.6 Derived reset operator // if 1757 pass

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

F.2.3.2.7 Derived unbounded temporal operators

- — `(always` p`)` $\equiv$ `(`p `until` 0`)`.
- — `(s_eventually` p`)` $\equiv$ `(not (always (not` p`)))`.
- — `(`p `s_until` q`)` $\equiv$ `((`p `until` q`) and s_eventually` q`)`.
- — `(`p `until_with` q`)` $\equiv$ `((`p `until (`p `and` q`))`.
- — `(`p `s_until_with` q`)` $\equiv$ `((`p `s_until (`p `and` q`))`.

F.2.3.2.8 Derived bounded temporal operators

- — `(s_nexttime` p`)` $\equiv$ `(not nexttime not` p`)`.
- — `(nexttime [`0`]` p`)` $\equiv$ `(`1 `|->` p`)`.
- — Let $m > 0$. `(nexttime [`m`]` p`)` $\equiv$ `(nexttime (nexttime [`m-1`]` p`))`.
- — Let $m \geq 0$. `(s_nexttime [`m`]` p`)` $\equiv$ `(not nexttime [`m`] not` p`)`.
- — Let $m \geq 0$. `(eventually [`m:m`]` p`)` $\equiv$ `(nexttime [`m`]` p`)`;
- — Let $m < n$. `(eventually [`m:n`]` p`)` $\equiv$ `(eventually [`m:n-1`]` p `or nexttime [`n`]` p`)`;
- — Let $m \geq 0$. `(always [`m:m`]` p`)` $\equiv$ `(nexttime [`m`]` p`)`;
- — Let $m < n$. `(always [`m:n`]` p`)` $\equiv$ `(always [`m:n-1`]` p `and nexttime [`n`]` p`)`;
- — Let $m \geq 0$. `(always [`m:$`]` p $\equiv$ `(nexttime [`m`] always` p`)`
- — Let $m \leq n$. `(s_eventually [`m:n`]` p`)` $\equiv$ `(not always [`m:n`] not` p`)`.
- — Let $m \geq 0$. `(s_eventually [`m:$`]` p`)` $\equiv$ `(s_nexttime [`m`] s_eventually` p`)`.
- — Let $m \leq n$. `(s_always [`m:n`]` p`)` $\equiv$ `(not eventually [`m:n`] 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.

— ~~$@(c)b \longmapsto (!c[*0:\$]$ `##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 $@(c)P_2)$.~~

F.3.1.1 Rewrite rules for sequences

The transformation $\mathcal{T}^s(S,c)$ recursively defined below produces a sequence R from a sequence S and a clock c:

- $\mathcal{T}^s(b,c)$ = (!c[*0:$] ##1 c & b).
- $\mathcal{T}^s$((1, $v=e$),c) = ($\mathcal{T}^s(1,c)$ ##0 (1, $v=e$)).
- $\mathcal{T}^s$((@(c_2) r),c_1) = ($\mathcal{T}^s(r,c_2)$).
- $\mathcal{T}^s$((r_1 ##1 r_2),c) = ($\mathcal{T}^s(r_1,c)$ ##1$\mathcal{T}^s(r_2,c)$).
- $\mathcal{T}^s$((r_1 ##0 r_2),c) = ($\mathcal{T}^s(r_1,c)$ ##0 $\mathcal{T}^s(r_2,c)$).
- $\mathcal{T}^s$((r_1 **or** r_2),c) = ($\mathcal{T}^s(r_1,c)$ **or** $\mathcal{T}^s(r_2,c)$).
- $\mathcal{T}^s$((r_1 **intersect** r_2),c) = ($\mathcal{T}^s(r_1,c)$ **intersect** $\mathcal{T}^s(r_2,c)$).
- $\mathcal{T}^s$((**first_match** (r)),c) = (**first_match** ($\mathcal{T}^s(r,c)$)).
- $\mathcal{T}^s$((r[*0]),c) = ($\mathcal{T}^s(r,c)$[*0]).
- $\mathcal{T}^s$((r[*1:$]),$c$) = ($\mathcal{T}^s(r,c)$[*1:$]).

F.3.1.2 Rewrite rules for properties

The transformation $\mathcal{T}^p(p,c)$ recursively defined below produces a property P from a property p and a clock c:

- $\mathcal{T}^p$(**strong**(r),c) = (**strong**($\mathcal{T}^s(r,c)$)).
- $\mathcal{T}^p$(**weak**(r),c) = (**weak**($\mathcal{T}^s(r,c)$)).
- $\mathcal{T}^p$((@(c_2) p),c_1) = $\mathcal{T}^p(p,c_2)$.
- $\mathcal{T}^p$((**disable iff**(b) p),c) = (**disable iff**(b) $\mathcal{T}^p(p,c)$).
- $\mathcal{T}^p$((**accept_on**(b) p),c) = (**accept_on**(b) $\mathcal{T}^p(p,c)$). // if 1757 pass
- $\mathcal{T}^p$((**not** p),c) = (**not** $\mathcal{T}^p(p,c)$).
- $\mathcal{T}^p$((r |-> p),c) = ($\mathcal{T}^s(r,c)$ |-> $\mathcal{T}^p(p,c)$).
- $\mathcal{T}^p$((p_1 **or** p_2),c) = ($\mathcal{T}^p(p_1,c)$ **or** $\mathcal{T}^p(p_2,c)$).
- $\mathcal{T}^p$((p_1 **and** p_2),c) = ($\mathcal{T}^p(p_1,c)$ **and** $\mathcal{T}^p(p_2,c)$).
- $\mathcal{T}^p$((**nexttime** p),c) = (!c **until** (c **and** **nexttime** (!c **until** (c **and** $\mathcal{T}^p(p,c)$)))).
- $\mathcal{T}^p$((p_1 **until** p_2),c) = ((**not** (c **and not** $\mathcal{T}^p(p_1,c)$)) **until** (c **and** $\mathcal{T}^p(p_2,c)$)).

F.3.3 Satisfaction without local variables

F.3.3.1 Neutral satisfaction

REPLACE

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.

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

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

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

— $w \models (P_1\ \texttt{or}\ P_2)$ iff $w \models P_1$ or $w \models P_2$.

— $w \models (P_1\ \texttt{and}\ P_2)$ iff $w \models P_1$ and $w \models P_2$.

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

WITH

Neutral satisfaction of properties is defined as follows:

— $w \models (P)$ iff $w \models P$.

— $w \models Q$ iff $w \models$ ~~Q'~~ $\mathcal{T}^p(Q,1)$ ~~, where Q' is the unclocked property that results from Q by applying the rewrite rules~~.

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

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

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

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

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

— $w \models (P_1\ \texttt{or}\ P_2)$ iff $w \models P_1$ or $w \models P_2$.

— $w \models (P_1\ \texttt{and}\ P_2)$ iff $w \models P_1$ and $w \models P_2$.

— $w \models (\texttt{nexttime}\ P)$ iff either $|w| = 0$ or $w^{1..} \models P$.

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

— $w \models (\texttt{accept\_on}\ (b)\ P)$ iff either // if 1757 pass

  — $w \models P$, or

  — For some $0 \leq i < |w|$, $w^i \models b$ and $w^{0,i-1}\top^\omega \models P$.
  Here, $w^{0,-1}$ denotes the empty word.

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

F.3.3.3 Vacuity

REPLACE

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

— Base:

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

— Induction:

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

    1. For every $0 \leq i < |w|, w^i \not\models b$.
    2. For some prefix $x \preceq w$, 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 $w \models^{\mathrm{non}} P$.

WITH

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

— Base:

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

— Induction:

  — $w \models^{\mathrm{non}} (P)$ iff $w \models^{\mathrm{non}} P$.
  — $w \models^{\mathrm{non}} R$ `|->` P iff there exists $i \geq 0$ such that $w^{0..i} \models R$ and $w^{i..} \models^{\mathrm{non}} P$.
  — $w \models^{\mathrm{non}} P_1$ **and** P_2 iff $w \models^{\mathrm{non}} P_1$ or $w \models^{\mathrm{non}} P_2$.
  — $w \models^{\mathrm{non}} P_1$ **or** P_2 iff $w \models^{\mathrm{non}} P_1$ or $w \models^{\mathrm{non}} P_2$.
  — $w \models^{\mathrm{non}} P_1$ **iff** P_2 iff $w \models^{\mathrm{non}} P_1$ or $w \models^{\mathrm{non}} P_2$.
  — $w \models^{\mathrm{non}} P_1$ **implies** P_2 iff $w \models^{\mathrm{non}} P_1$.
  — $w \models^{\mathrm{non}}$ **not** P iff $w \models^{\mathrm{non}} P$.
  — $w \models^{\mathrm{non}}$ **nexttime** P iff $|w| > 0$ and $w^{1..} \models^{\mathrm{non}} P$.

- $w \models^{\text{non}} P_1$ **until** P_2 iff there exists $0 \leq i < |w|$ such that the following holds:
  - Either $w^{i..} \models^{\text{non}} P_1$ or $w^{i..} \models^{\text{non}} P_2$ and
  - For all $0 \leq j < i$, $w^{j..} \models P_1$ **and not** P_2.
- $w \models^{\text{non}} P_1$ **s_until** P_2 iff there exists $0 \leq i < |w|$ such that the following holds:
  - Either $w^{i..} \models^{\text{non}} P_1$ or $w^{i..} \models^{\text{non}} P_2$ and
  - For all $0 \leq j < i$, $w^{j..} \models P_1$ **and not** P_2.
- $w \models^{\text{non}}$ **always** P iff there exists $0 \leq i < |w|$ such that the following holds:
  - $w^{i..} \models^{\text{non}} P$ and
  - For all $0 \leq j < i$, $w^{j..} \models P$.
- $w \models^{\text{non}}$ **always** $[m : n]P$ iff there exists $m \leq i \leq n$ such that the following holds:
  - $w^{i..} \models^{\text{non}} P$ and
  - For all $m \leq j < i$, $w^{j..} \models P$.
- $w \models^{\text{non}}$ **s_always** $[m : n]P$ iff there exists $m \leq i \leq n$ such that the following holds:
  - $w^{i..} \models^{\text{non}} P$ and
  - For all $m \leq j < i$, $w^{j..} \models P$.
- $w \models^{\text{non}}$ **s_eventually** P iff there exists $0 \leq i < |w|$ such that the following holds:
  - $w^{i..} \models^{\text{non}} P$ and
  - For all $0 \leq j < i$, $w^{j..} \models$ **not** P.
- $w \models^{\text{non}}$ **eventually** $[\texttt{m:n}]P$ iff there exists $m \leq i \leq n$ such that the following holds:
  - $w^{i..} \models^{\text{non}} P$ and
  - For all $m \leq j < i$, $w^{j..} \models$ **not** P.
- $w \models^{\text{non}}$ **s_eventually** $[\texttt{m:n}]P$ iff there exists $m \leq i \leq n$ such that the following holds:
  - $w^{i..} \models^{\text{non}} P$ and
  - For all $m \leq j < i$, $w^{j..} \models$ **not** P.
- $w \models^{\text{non}}$ **disable iff** $(b)P$ iff $w \models^{\text{non}} P$ and one of the following holds:
  1. For every $0 \leq i < |w|$, $w^i \not\models b$.
  2. For some prefix $x \preceq$ of w, we have that for every $0 \leq i < |x|$, $x^i \not\models b$,and either $x\bot^\omega \models P$ or $x\top^\omega \not\models P$.
- $w \models^{\text{non}}$ **accept_on** (b) P iff $w \models^{\text{non}} P$ and one of the following holds:
  1. For every $0 \leq i < |w|$, $w^i \not\models b$.
  2. For some prefix x of w, we have that for every $0 \leq i < |x|$, $x^i \not\models b$,and either $x\bot^\omega \models P$ or $x\top^\omega \not\models P$.
- $w \models^{\text{non}}$ **reject_on** (b) P iff $w \models^{\text{non}} P$ and one of the following holds:
  1. For every $0 \leq i < |w|$, $w^i \not\models b$.
  2. For some prefix x of 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 $w \models^{\text{non}} P$.

The $\models^{\text{non}}$ relation is not explicitly defined for all the derived operators. For these operators the $\models^{\text{non}}$ relation is implicitly defined by unrolling their derivation.

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.
- $w, L_0 \models$ **not** P iff $\overline{w}, L_0 \not\models P$.
- $w, L_0 \models R$ iff there exist $0 \leq j < |w|$ and L_1 so that $w^{0,j}, L_0, L_1 \models\!\equiv R$.
- $w, L_0 \models (R$ `|->` $P)$ iff for every $0 \leq j < |w|$ and L_1 so that $\overline{w}^{0,j}, L_0, L_1 \models\!\equiv R, w^{j..}, L_1 \models P$.
- $w, L_0 \models (P)$ 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 $w, L_0 \models P_2$.

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'~~ $\mathcal{T}^p(Q, 1)$ ~~, where Q' is the unclocked property that results from Q by applying the rewrite rules~~.
- w, L_0**not** P iff $\overline{w}, L_0 \not\models P$.
- ~~$w, L_0 \models R$ iff there exist $0 \leq j < |w|$ and L_1 so that $w^{0,j}, L_0, L_1 \models\!\equiv R$.~~
- $w, L_0 \models$ **strong** (R) iff there exist $0 \leq j < |w|$ and L_1 so that $w^{0,j}, L_0, L_1 \models\!\equiv R$.
- $w, L_0 \models$ **weak** (R) iff for every $0 \leq j < |w|, w^{0,j}\top^\omega, L_0 \models$ **strong** (R).
- $w, L_0 \models (R$ `|->` $P)$ iff for every $0 \leq j < |w|$ and L_1 so that $\overline{w}^{0,j}, L_0, L_1 \models\!\equiv R, w^{j..}, L_1 \models P$.
- $w, L_0 \models (P)$ 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 $w, L_0 \models P_2$.
- $w, L_0 \models ($ **nexttime** $P)$ iff either $|w| = 0$ or $w^{1..}, L_0 \models P$.
- $w, L_0 \models (P_1$ **until** $P_2)$ iff either there exists $0 \leq j < |w|$ so that $w^{j..}, L_0 \models P_2$ and for every $0 \leq i < j, w^{i..}, L_0 \models P_1$, or for every $0 \leq i < |w|, w^{i..}, L_0 \models P_1$.
- $w, L_0 \models ($ **accept_on** $(b)\, P)$ iff either // if 1757 pass
  - $w, L_0 \models P$ and no letter of w satisfies b, or
  - For some $0 \leq i < |w|$, $w^i \models b$ and $w^{0,i-1}\top^\omega, L_0 \models P$. Here, $w^{0,-1}$ denotes the empty word.