

Annex F

Formal semantics of concurrent assertions

F.2 Abstract syntax

F.2.1 Abstract grammars

REPLACE

The abstract grammar for unlocked 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 unlocked properties is

```
P ::= R // "sequence" form
| strong(R) // "strong sequence" form
| (P) // "parenthesis" form
| not P // "negation" form
| (P or P) // "or" form
| (P and P) // "and" form
| (R → P) // "implication" form
| <next> P // "<next>" form
| (P <until> P) // "<until>" form
| accept.on(b) P // "<accept_on>" form
```

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
   | (Q)               // "parenthesis" form
   | not Q             // "negation" form
   | ( Q or Q )        // "or" form
   | ( Q and Q )       // "and" form
   | ( S |-> Q )       // "implication" form
   | <next> Q          // "<next>" form
   | ( Q <until> Q )   // "<until>" form
   | 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

- $(R_1 \mid\Rightarrow P) \equiv ((R_1 \#1 1) \mid\rightarrow P)$.
- $(S_1 \mid\Rightarrow Q) \equiv ((S_1 \#1 @ (1) 1) \mid\rightarrow Q)$.

F.2.3.2 Derived consecutive repetition operators

- Let $m > 0$. $R[*m] \equiv (R \#1 R \#1 \dots \#1 R)$ // m copies of R .
- $R[*0 : \$] \equiv (R[*0] \text{ or } R[1 : \$])$.
- Let $m \leq n$. $R[*m : n] \equiv R[*m] \text{ or } R[*m + 1] \text{ or } \dots \text{ or } R[*n]$.
- Let $m > 1$. $R[*m : \$] \equiv (R[*m-1] \#1 R[*1 : \$])$.

F.2.3.3 Derived delay and concatenation operators

Let $m < n$.

- $(\#\#[m : n] R) \equiv (1 [*m : n] \#\#1 R)$.
- $(\#\#[m : \$] R) \equiv (1 [*m : \$] \#\#1 R)$.
- $(\#\#m R) \equiv (1 [*m] \#\#1 R)$.
- Let $m > 0$. $(R_1 \#\#[m : n] R_2) \equiv (R_1 \#\#1 1 [m - 1 : n - 1] \#\#1 R_2)$.
- Let $m > 0$. $(R_1 \#\#[m : \$] R_2) \equiv (R_1 \#\#1 1 [m - 1 : \$] \#\#1 R_2)$.
- Let $m > 1$. $(R_1 \#\#m R_2) \equiv (R_1 \#\#1 1 [m - 1] \#\#1 R_2)$.
- $(R_1 \#\#[0 : 0] R_2) \equiv (R_1 \#\#0 R_2)$.
- Let $n > 0$. $(R_1 \#\#[0 : n] R_2) \equiv ((R_1 \#\#0 R_2) \text{ or } (R_1 \#\#[1 : n] R_2))$.
- $(R_1 \#\#[0 : \$] R_2) \equiv ((R_1 \#\#0 R_2) \text{ or } (R_1 \#\#[1 : \$] R_2))$.

F.2.3.4 Derived nonconsecutive repetition operators

Let $m \leq n$.

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

F.2.3.5 Other derived operators

- $(R_1 \text{ and } R_2) \equiv (((R_1 \#\#1 1 [*0 : \$]) \text{ intersect } R_2) \text{ or } (R_1 \text{ intersect } (R_2 \#\#1 1 [*0 : \$])))$.
- $(R_1 \text{ within } R_2) \equiv ((1 [*0 : \$] \#\#1 R_1 \#\#1 1 [*0 : \$]) \text{ intersect } R_2)$.
- $(b \text{ throughout } R) \equiv ((b1 [*0 : \$]) \text{ intersect } R)$.
- $(R, v = e) \equiv (R \#\#0 (1, v = e))$.
- $(R, v_1 = e_1, \dots, v_k = e_k) \equiv ((R, v_1 = e_1) \#\#0 (1, v_2 = e_2, \dots, v_k = e_k))$ for $k > 1$.
- $(\text{if}(b) P) \equiv (b \mid-> P)$.
- $(\text{if}(b) P_1 \text{ else } P_2) \equiv (b \mid-> P_1) \text{ and } (!b \mid-> P_2)$.

WITH

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

F.2.3.1 Derived nonoverlapping implication operator

- $(R_1 \mid \Rightarrow P) \equiv ((R_1 \#\#\#1) \mid \rightarrow P)$.
- $(S_1 \mid \Rightarrow Q) \equiv ((S_1 \#\#\#1@(1)1) \mid \rightarrow Q)$.

F.2.3.1 Derived sequence operators

F.2.3.21.1 Derived consecutive repetition operators

- ~~Let $m > 0$. $R[*m] \equiv (R \#\#\#1R \#\#\#1 \dots \#\#\#1 + R) // m \text{ copies of } R$.~~
- Let $m > 0$. $R[*m] \equiv (R[*m - 1] \#\#\#1 R)$.
- $R[*0 : \$] \equiv (R[*0] \text{ or } R[1 : \$])$.
- ~~Let $m \leq n$. $R[*m : n] \equiv R[*m] \text{ or } R[*m + 1] \text{ or } \dots \text{ or } R[*n]$.~~
- $R[*m : m] \equiv R[*m]$.
- Let $m < n$. $R[*m : n] \equiv (R[*m : n - 1] \text{ or } R[*n])$.
- Let $m > 1$. $R[*m : \$] \equiv (R[*m - 1] \#\#\#1 R[*1 : \$])$.

F.2.3.31.2 Derived delay and concatenation operators

Let $m < n$.

- $(\#\#[m : n] R) \equiv (1[*m : n] \#\#\#1 R)$.
- $(\#\#[m : \$] R) \equiv (1[*m : \$] \#\#\#1 R)$.
- $(\#\\#m R) \equiv (1[*m] \#\#\#1 R)$.
- Let $m > 0$. $(R_1 \#\#[m : n] R_2) \equiv (R_1 \#\#\#1 1[m - 1 : n - 1] \#\#\#1 R_2)$.
- Let $m > 0$. $(R_1 \#\#[m : \$] R_2) \equiv (R_1 \#\#\#1 1[m - 1 : \$] \#\#\#1 R_2)$.
- Let $m > 1$. $(R_1 \#\\#m R_2) \equiv (R_1 \#\#\#1 1[m - 1] \#\#\#1 R_2)$.
- $(R_1 \#\#[0 : 0] R_2) \equiv (R_1 \#\#\#0 R_2)$.
- Let $n > 0$. $(R_1 \#\#[0 : n] R_2) \equiv ((R_1 \#\#\#0 R_2) \text{ or } (R_1 \#\#[1 : n] R_2))$.
- $(R_1 \#\#[0 : \$] R_2) \equiv ((R_1 \#\#\#0 R_2) \text{ or } (R_1 \#\#[1 : \$] R_2))$.

F.2.3.41.3 Derived nonconsecutive repetition operators

Let $m \leq n$.

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

F.2.3.5.1.4 Other derived operators

- $(R_1 \text{ and } R_2) \equiv (((R_1 \ \#\#1 \ 1 \ [*0:\$]) \text{ intersect } R_2) \text{ or } (R_1 \text{ intersect } (R_2 \ \#\#1 \ 1 \ [*0:\$])))$.
- $(R_1 \text{ within } R_2) \equiv ((1 \ [*0:\$] \ \#\#1 \ R_1 \ \#\#1 \ 1 \ [*0:\$]) \text{ intersect } R_2)$.
- $(b \text{ throughout } R) \equiv ((b \ 1 \ [*0:\$]) \text{ intersect } R)$.
- $(R, v = e) \equiv (R \ \#\#0 \ (1, v = e))$.
- $(R, v_1 = e_1, \dots, v_k = e_k) \equiv ((R, v_1 = e_1) \ \#\#0 \ (1, v_2 = e_2, \dots, v_k = e_k))$ for $k > 1$.
- ~~$(\text{if}(b) P) \equiv (b \ \rightarrow P)$.~~
- ~~$(\text{if}(b) P_1 \text{ else } P_2) \equiv ((b \ \rightarrow P_1) \text{ and } (!b \ \rightarrow P_2))$.~~

F.2.3.2 Derived property operators

F.2.3.2.1 Derived boolean operators

- $P_1 \text{ imply } P_2 \equiv (\text{not } P_1 \text{ or } P_2)$.
- $P_1 \text{ iff } P_2 \equiv ((P_1 \ \rightarrow P_2) \text{ and } (P_2 \ \rightarrow P_1))$.

F.2.3.2.2 Derived nonoverlapping implication operator

- $(R \ \mid\Rightarrow P) \equiv ((R \ \#\#1 \ 1) \ \mid\rightarrow P)$.
- $(S \ \mid\Rightarrow Q) \equiv ((S \ \#\#1 \ \text{@}(1) \ 1) \ \mid\rightarrow Q)$.

F.2.3.2.3 Derived conditional operators

- $(\text{if}(b) P) \equiv (b \ \mid\rightarrow P)$.
- $(\text{if}(b) P_1 \text{ else } P_2) \equiv ((b \ \mid\rightarrow P_1) \text{ and } (!b \ \mid\rightarrow P_2))$.

F.2.3.2.4 Derived followed by operators

- $(r \ \#\# p) \equiv (\text{not}(r \ \mid\rightarrow \text{not } p))$.
- $(r \ \#\# p) \equiv (\text{not}(r \ \mid\Rightarrow \text{not } p))$.

F.2.3.2.5 Derived reset operator

- $(\text{reject_on}(b) p) \equiv (\text{not}(\text{accept_on}(b) \text{ not } p))$.

F.2.3.2.6 Derived unbounded temporal operators

- $(\text{eventually } p) \equiv (1 \ < \text{until } > p)$.
- $(\text{always } p) \equiv (\text{not } (1 \ < \text{until } > \text{not } p))$.
- $(p \text{ until } q) \equiv ((p \ < \text{until } > q) \text{ or } \text{always } p)$.
- $(p \text{ until with } q) \equiv ((p \text{ until } (p \text{ and } q))$.
- $(p \ < \text{until with } > q) \equiv ((p \ < \text{until } > (p \text{ and } q))$.

F.2.3.2.7 Derived bounded temporal operators

- $(\mathbf{next} \ p) \equiv (\mathbf{not} \ < \mathbf{next} \ > \ \mathbf{not} \ p)$.
- $(\ < \mathbf{next} \ > [1] \ p) \equiv \ < \mathbf{next} \ > p$.
- Let $m > 1$. $(\ < \mathbf{next} \ > [m] \ p) \equiv (\ < \mathbf{next} \ > (\ < \mathbf{next} \ > [n-1] \ p))$.
- Let $m > 1$. $(\mathbf{next}[m] \ p) \equiv (\mathbf{not} \ < \mathbf{next} \ > [m] \ \mathbf{not} \ p)$.
- $(\ < \mathbf{next} \ > [0 : \$] \ p) \equiv (\mathbf{eventually} \ p)$.
- Let $m \geq 0$. $(\ < \mathbf{next} \ > [m : m] \ p) \equiv (\ < \mathbf{next} \ > [m] \ p)$;
- Let $m < n$. $(\ < \mathbf{next} \ > [m : n] \ p) \equiv (\ < \mathbf{next} \ > [m : n-1] \ \mathbf{or} \ \ < \mathbf{next} \ > [n] \ p)$;
- Let $m > 0$. $(\ < \mathbf{next} \ > [m : \$] \ p) \equiv (\ < \mathbf{next} \ > [m] \ \mathbf{eventually} \ p)$.
- Let $m \geq 0$. $(\ < \mathbf{always} \ > [m : m] \ p) \equiv (\ < \mathbf{next} \ > [m] \ p)$;
- Let $m < n$. $(\ < \mathbf{always} \ > [m : n] \ p) \equiv (\ < \mathbf{always} \ > [m : n-1] \ \mathbf{and} \ \ < \mathbf{next} \ > [n] \ p)$;
- Let $m \geq 0$. $(\mathbf{always}[m : \$] \ p) \equiv (\mathbf{next}[m] \ \mathbf{always} \ p)$
- Let $m < n$. $(\mathbf{next}[m : n] \ p) \equiv (\mathbf{not} \ \ < \mathbf{always} \ > [m : n] \ \mathbf{not} \ p)$.
- Let $m < n$. $(\mathbf{always}[m : n] \ p) \equiv (\mathbf{not} \ \ < \mathbf{next} \ > [m : n] \ \mathbf{not} \ p)$.

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.

- $\ @ (c) \ b \mapsto (!c[*0:\$] \ \#\#1 \ c \ \& \ b)$.
- $\ @ (c) \ (1, v = e) \mapsto (\ @ (c)1 \ \#\#0 \ (1, v = e))$.
- $\ @ (c) \ (P) \mapsto (\ @ (c)P)$.
- $\ @ (c) \ (R_1 \ \#\#1 \ R_2) \mapsto (\ @ (c)R_1 \ \#\#1 \ @ (c)R_2)$.
- $\ @ (c) \ (R_1 \ \#\#0 \ R_2) \mapsto (\ @ (c)R_1 \ \#\#0 \ @ (c)R_2)$.
- $\ @ (c) \ (R_1 \ \mathbf{or} \ R_2) \mapsto (\ @ (c)R_1 \ \mathbf{or} \ @ (c)R_2)$.
- $\ @ (c) \ (R_1 \ \mathbf{intersect} \ R_2) \mapsto (\ @ (c)R_1 \ \mathbf{intersect} \ @ (c)R_2)$.
- $\ @ (c) \ \mathbf{first_match}(R) \mapsto (\mathbf{first_match}(\ @ (c)R))$.

- $@(c) R[*0] \mapsto (@(c)R)[*0]$.
- $@(c) R[*1:\$] \mapsto (@(c)R)[*1:\$]$.
- $@(c) \text{disable iff}(b) P \mapsto \text{disable iff}(b) (@(c)P)$.
- $@(c) \text{not } P \mapsto \text{not } @(c)P$.
- $@(c) (P_1 \xrightarrow{i} P_2) \mapsto (@(c)P_1 \xrightarrow{i} @(c)P_2)$.
- $@(c) (P_1 \text{ or } P_2) \mapsto (@(c)P_1 \text{ or } @(c)P_2)$.
- $@(c) (P_1 \text{ and } P_2) \mapsto (@(c)P_1 \text{ and } @(c)P_2)$.

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.

- $@(e)b \mapsto (!e[*0:\$]###1e\&b)$.
- $@(e)(1, v = e) \mapsto (@(e)1###0(1, v = e))$.
- $@(e)(P) \mapsto (@(e)P)$.
- $@(e)(R_1###1R_2) \mapsto (@(e)R_1###1@(e)R_2)$.
- $@(e)(R_1###0R_2) \mapsto (@(e)R_1###0@(e)R_2)$.
- $@(e)(R_1 \text{ or } R_2) \mapsto (@(e)R_1 \text{ or } @(e)R_2)$.
- $@(e)(R_1 \text{ intersect } R_2) \mapsto (@(e)R_1 \text{ intersect } @(e)R_2)$.
- $@(e) \text{first_match}(R) \mapsto (\text{first_match}(@(e)R))$.
- $@(e) R[*0] \mapsto (@(e)R)[*0]$.
- $@(e) R[*1:\$] \mapsto (@(e)R)[*1:\$]$.
- $@(e) \text{disable iff}(b) P \mapsto \text{disable iff}(b) @(e)P$.
- $@(e) \text{not } P \mapsto \text{not } @(e)P$.
- $@(e)(P_1 \xrightarrow{\quad} P_2) \mapsto (@(e)P_1 \xrightarrow{\quad} @(e)P_2)$.
- $@(e)(P_1 \text{ or } P_2) \mapsto (@(e)P_1 \text{ or } @(e)P_2)$.
- $@(e)(P_1 \text{ and } P_2) \mapsto (@(e)P_1 \text{ and } @(e)P_2)$.

F.3.1.1 Rewrite rules for sequences

The transformation T^s recursively defined below produces an unlocked sequence R equivalent to a given clocked sequence S :

- $T^s(@ (c) b) = (!c[*0:\$] \#\#1 c \& b)$.
- $T^s(@ (c) (1, v = e)) = (T^s(@ (c) 1) \#\#0 (1, v = e))$.
- $T^s(@ (c_1) (@ (c_2) S)) = (T^s(@ (c_2) S))$.
- $T^s(@ (c) (S_1 \#\#1 S_2)) = (T^s(@ (c) S_1) \#\#1 T^s(@ (c) S_2))$.
- $T^s(@ (c) (S_1 \#\#0 S_2)) = (T^s(@ (c) S_1) \#\#0 T^s(@ (c) S_2))$.
- $T^s(@ (c) (S_1 \text{ or } S_2)) = (T^s(@ (c) S_1) \text{ or } T^s(@ (c) S_2))$.
- $T^s(@ (c) (S_1 \text{ intersect } S_2)) = (T^s(@ (c) S_1) \text{ intersect } T^s(@ (c) S_2))$.
- $T^s(@ (c) (\text{first_match } S)) = (\text{first_match } T^s(@ (c) S))$.
- $T^s(@ (c) (S[*0])) = ((T^s(@ (c) S))[*0])$.
- $T^s(@ (c) (S[*1:\$])) = ((T^s(@ (c) S))[*1:\$])$.

F.3.1.2 Rewrite rules for properties

The transformation T^p recursively defined below produces an unlocked property P equivalent to a given clocked property Q :

- $T^p(S) = T^s(S)$.
- $T^p(\text{strong}(S)) = (\text{strong}(T^s(S)))$.
- $T^p(@ (c_1) (@ (c_2) P)) = T^p(@ (c_2) P)$.
- $T^p(@ (c_1) (@ (c_2) Q)) = T^p(@ (c_2) Q)$.
- $T^p(@ (c) \text{disable iff}(b) Q) = (\text{disable iff}(b) T^p(@ (c) Q))$.
- $T^p(@ (c) \text{accept_on}(b) Q) = (\text{accept_on}(b) T^p(@ (c) Q))$.
- $T^p(@ (c) \text{not } Q) = (\text{not } T^p(@ (c) Q))$.
- $T^p(@ (c) (S \mid\rightarrow Q)) = (T^s(@ (c) S) \mid\rightarrow T^p(@ (c) Q))$.
- $T^p(@ (c) (Q_1 \text{ or } Q_2)) = (T^p(@ (c) Q_1) \text{ or } T^p(@ (c) Q_2))$.
- $T^p(@ (c) (Q_1 \text{ and } Q_2)) = (T^p(@ (c) Q_1) \text{ and } T^p(@ (c) Q_2))$.
- $T^p(@ (c) < \text{next} > Q) = (!c < \text{until} > (c \text{ and } < \text{next} > (!c < \text{until} > (c \text{ and } T^p(Q))))$.
- $T^p(@ (c) (Q_1 < \text{until} > Q_2)) = ((\text{not}(c \text{ and } \text{not } T^p(Q_1)) < \text{until} > (c \text{ and } T^p(Q_2)))$.

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 unlocked property that results from Q by applying the rewrite rules.
- $w \models \text{not } P$ iff $\bar{w} \not\models P$.
- $w \models R$ iff there exists $0 \leq j < |w|$ so that $w^{0,j} \models R$.
- $w \models (R \mid \rightarrow P)$ iff for every $0 \leq j < |w|$ so that $\bar{w}^{0,j} \models R, w^{j,\cdot} \models P$.
- $w \models (P_1 \text{ or } P_2)$ iff $w \models P_1$ or $w \models P_2$.
- $w \models (P_1 \text{ and } P_2)$ iff $w \models P_1$ and $w \models P_2$.

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

WITH

Neutral satisfaction of properties is defined as follows:

- $w \models (P)$ iff $w \models P$.
- $w \models Q$ iff $w \models Q' \text{ } T^p(Q)$, where Q' is the unlocked property that results from Q by applying the rewrite rules.
- $w \models \text{not } P$ iff $\bar{w} \not\models P$.
- ~~$w \models R$ iff there exists $0 \leq j < |w|$ so that $w^{0,j} \models R$.~~
- $w \models \text{strong } (R)$ iff there exists $0 \leq j < |w|$ so that $w^{0,j} \models R$.
- $w \models R$ iff for every $0 \leq j < |w|, w^{0,j} \text{ } T^\omega \models \text{strong } (R)$.
- $w \models (R \mid \rightarrow P)$ iff for every $0 \leq j < |w|$ so that $\bar{w}^{0,j} \models R, w^{j,\cdot} \models P$.
- $w \models (P_1 \text{ or } P_2)$ iff $w \models P_1$ and $w \models P_2$.
- $w \models (P_1 \text{ and } P_2)$ iff $w \models P_1$ and $w \models P_2$.
- $w \models (\langle \text{next} \rangle P)$ iff $|w| > 1$ and $w^{1,\cdot} \models P$.
- $w \models (P_1 \langle \text{until} \rangle P_2)$ iff there exists $0 \leq j < |w|$ so that $w^{k,\cdot} \models P_2$ and for every $0 \leq i < j, w^{i,\cdot} \models P_1$.
- $w \models (\text{accept_on } (b)P)$ iff either
 - $w \models P$ and no letter of w satisfies b , or
 - Some letter of w satisfies b and $w^{0,i-1} \text{ } T^\omega \models P$ 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 $w \models \text{not } b$ iff $w \models !b$.

F.3.3.3 Vacuity

REPLACE

This subclause defines the relation of non-vacuity, denoted \models^{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 $w \models^{\text{non}} P_1$.
 - $w \models^{\text{non}} R \mid \rightarrow P_1$ 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 $w \models^{\text{non}} P_2$.
 - $w \models^{\text{non}} \mathbf{not}P_1$ iff $w \models^{\text{non}} P_1$.
 - $w, L \models^{\text{non}} \mathbf{disable\ iff}(b)P_1$ 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 $x \perp^\omega \models P$ 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^{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 $w \models^{\text{non}} R$.~~
 - $w \models^{\text{non}} \mathbf{strong}(R)$.
- Induction:
 - $w \models^{\text{non}} (P_{\top})$ iff $w \models^{\text{non}} P_{\top}$.
 - $w \models^{\text{non}} R \mid \rightarrow P_{\top}$ iff there exists $i \geq 0$ such that $w^{0..i} \models R$ and $w^{i..} \models^{\text{non}} P_{\top}$.
 - $w \models^{\text{non}} P_1$ **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 $w \models^{\text{non}} P_2$.
 - $w \models^{\text{non}} \mathbf{not}P_{\top}$ iff $w \models^{\text{non}} P_{\top}$.
 - $w \models^{\text{non}} < \mathbf{next} > P$ iff $w^{1..} \models^{\text{non}} P$.
 - $w \models^{\text{non}} P_1 < \mathbf{until} > P_2$ 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 $w^{j..} \models^{\text{non}} P_1$.
 - $w, \bar{L} \models^{\text{non}} \mathbf{disable\ iff}(b)P_{\top}$ iff $w \models^{\text{non}} P_{\top}$ and one of the following holds:

1. For every $0 \leq i < |w|$, $w^i \not\prec b$.
 2. For some prefix $x \preceq_{\text{of}} w$, we have that for every $0 \leq i < |x|$, $x^i \not\prec b$, and either $x \perp^\omega \models P$ or $x \top^\omega \not\models P$.
- $w \models^{\text{non}} \text{accept_on}(b)P$ 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 unlocked property that results from Q by applying the rewrite rules.
- $w, L_0 \models \text{not } P$ iff $\bar{w}, L_0 \not\models P$.
- $w, L_0 \models R$ iff there exist $0 \leq j < |w|$ and L_1 so that $w^{0:j}, L_0, L_1 \models R$.
- $w, L_0 \models (R \mid \rightarrow P)$ iff for every $0 \leq j < |w|$ and L_1 so that $\bar{w}^{0:j}, L_0, L_1 \models R$, $w^{j:\cdot}, L_1 \models P$.
- $w, L_0 \models (P)$ iff $w, L_0 \models P$.
- $w, L_0 \models (P_1 \text{ or } P_2)$ iff $w, L_0 \models P_1$ or $w, L_0 \models P_2$.
- $w, L_0 \models (P_1 \text{ and } P_2)$ 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 unlocked property that results from Q by applying the rewrite rules.
- $w, L_0 \models \text{not } P$ iff $\bar{w}, L_0 \not\models P$.
- ~~$w, L_0 \models R$ iff there exist $0 \leq j < |w|$ and L_1 so that $w^{0:j}, L_0, L_1 \models R$.~~
- $w, L_0 \models \text{strong } (R)$ iff there exist $0 \leq j < |w|$ and L_1 so that $w^{0:j}, L_0, L_1 \models R$.
- $w, L_0 \models R$ iff for every $0 \leq j < |w|$, $w^{0:j} \top^\omega, L_0 \models \text{strong } (R)$.
- $w, L_0 \models (R \mid \rightarrow P)$ iff for every $0 \leq j < |w|$ and L_1 so that $\bar{w}^{0:j}, L_0, L_1 \models R$, $w^{j:\cdot}, L_1 \models P$.
- $w, L_0 \models (P)$ iff $w, L_0 \models P$.

- $w, L_0 \models (P_1 \text{ or } P_2)$ iff $w, L_0 \models P_1$ or $w, L_0 \models P_2$.
- $w, L_0 \models (P_1 \text{ and } P_2)$ iff $w, L_0 \models P_1$ and $w, L_0 \models P_2$.
- $w, L_0 \models (\langle \text{next} \rangle P)$ iff $|w| > 1$ and $w^{1..}, L_0 \models P$.
- $w, L_0 \models (P_1 \langle \text{until} \rangle P_2)$ iff there exists $0 \leq j < |w|$ so that $w^{k..}, L_0 \models P_2$ and for every $0 \leq i < j, w^{i..}, L_0 \models P_1$.
- $w, L_0 \models (\text{accept_on } (b)P)$ 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.