

Appendix B

(normative)

Formal Syntax and Semantics of Accellera PSL

This appendix formally describes the syntax and semantics of the temporal layer.

B.1 Syntax

The logic Accellera PSL is defined with respect to a non-empty set of atomic propositions P and a given set of boolean expressions B over P . We assume two designated boolean expression *true* and *false* belong to B .

Definition 1 (Sequential Extended Regular Expressions (SEREs)).

- Every boolean expression $b \in B$ is a SERE.
- If r , r_1 , and r_2 are SEREs, and c is a boolean expression, then the following are SEREs:

• $\{r\}$	• $r_1 ; r_2$	• $\{r_1\} : \{r_2\}$	• $\{r_1\} \{r_2\}$
• $\{r_1\} \&\& \{r_2\}$	• $r[*0]$	• $r[*]$	• $r@c$

Definition 2 (Formulas of the Foundation Language (FL formulas)).

- If b is a boolean expression then both b and $b!$ are FL formulas.
- If φ and ψ are FL formulas, r, r_1, r_2 are SEREs, and b a boolean expression, then the following are FL formulas:

• (φ)	• $\neg\varphi$	• $\varphi \wedge \psi$	• $\{r\}!$	• $\{r\}$
• $X! \varphi$	• $[\varphi U \psi]$	• $\varphi \text{ abort } b$	• $\varphi@b$	• $\{r\} \mapsto \varphi$

Definition 3 (Formulas of the Optional Branching Extension (OBE)).

- Every boolean expression is an OBE formula.
- If f , f_1 , and f_2 are OBE formulas, then so are the following:
 - (f)
 - $\neg f$
 - $f_1 \wedge f_2$
 - EXf
 - $E[f_1 U f_2]$
 - EGf

Additional OBE operators are derived from these as follows:

- $f_1 \vee f_2 = \neg(\neg f_1 \wedge \neg f_2)$
- $f_1 \rightarrow f_2 = \neg f_1 \vee f_2$
- $f_1 \leftrightarrow f_2 = (f_1 \rightarrow f_2) \wedge (f_2 \rightarrow f_1)$
- $EFf = E[\text{true} U f]$

- $AXf = \neg EX\neg f$
- $A[f_1 U f_2] = \neg(E[\neg f_2 U (\neg f_1 \wedge \neg f_2)] \vee EG\neg f_2)$
- $AGf = \neg E[\text{true} U \neg f]$
- $AFf = A[\text{true} U f]$

Definition 4 (Accellera PSL Formulas).

- Every FL formula is an Accellera PSL formula.
- Every OBE formula is an Accellera PSL formula.

In Section B.3, we show additional operators which provide syntactic sugaring to the ones above.

B.2 Semantics

B.2.1 Semantics of FL formulas

The semantics of FL is defined with respect to finite and infinite words over $\Sigma = 2^P \cup \{\top, \perp\}$. We denote a letter from Σ by ℓ and an empty, finite, or infinite word from Σ by u , v , or w (possibly with subscripts). We denote the length of word v as $|v|$. An empty word $v = \epsilon$ has length 0, a finite word $v = (\ell_0 \ell_1 \ell_2 \dots \ell_n)$ has length $n + 1$, and an infinite word has length ∞ . We use i , j , and k to denote non-negative integers. We denote the i^{th} letter of v by v^{i-1} (since counting of letters starts at zero). We denote by $v^{i..}$ the suffix of v starting at v^i . That is, for every $i < |v|$, $v^{i..} = v^i v^{i+1} \dots v^n$ or $v^{i..} = v^i v^{i+1} \dots$. We denote by $v^{i..j}$ the finite sequence of letters starting from v^i and ending in v^j . That is, for $j \geq i$, $v^{i..j} = v^i v^{i+1} \dots v^j$ and for $j < i$, $v^{i..j} = \epsilon$. We use ℓ^ω to denote an infinite-length word, each letter of which is ℓ .

We use \bar{v} to denote the word obtained by replacing every \top with a \perp and vice versa. We call \bar{v} the *complement* of v .

The semantics of FL *formulas* over *words* is defined inductively, using as the base case the semantics of *boolean expressions* over *letters* in Σ . The semantics of boolean expression is assumed to be given as a relation $\models \subseteq \Sigma \times B$ relating letters in Σ with boolean expressions in B . If $(\ell, b) \in \models$ we say that the letter ℓ *satisfies* the boolean expression b and denote it $\ell \models b$. We assume the two special letters \top and \perp behave as follows: for every boolean expression b , $\top \models b$ and $\perp \not\models b$. We assume that otherwise the boolean relation \models behaves in the usual manner. In particular, that for every letter $\ell \in 2^P$, atomic proposition $p \in P$ and boolean expressions $b, b_1, b_2 \in B$ (i) $\ell \models p$ iff $p \in \ell$, (ii) $\ell \models \neg b$ iff $\ell \not\models b$, and (iii) $\ell \models \text{true}$ and $\ell \not\models \text{false}$. Finally, we assume that for every letter $\ell \in \Sigma$, $\ell \models b_1 \wedge b_2$ iff $\ell \models b_1$ and $\ell \models b_2$.

B.2.1.1 Unlocked Semantics

B.2.1.1.1 Semantics of unlocked SEREs

Unlocked SEREs are defined over finite words from the alphabet Σ . The notation $v \models r$, where r is a SERE and v a finite word means that v *models tightly* r . The semantics of unlocked SEREs are defined as follows, where b denotes a boolean expression, and r , r_1 , and r_2 denote unlocked SEREs.

- $v \models \{r\} \iff v \models r$
- $v \models b \iff |v| = 1 \text{ and } v \models b$
- $v \models r_1; r_2 \iff \exists v_1, v_2 \text{ s.t. } v = v_1v_2, v_1 \models r_1, \text{ and } v_2 \models r_2$
- $v \models \{r_1\}:\{r_2\} \iff \exists v_1, v_2, \text{ and } \ell \text{ s.t. } v = v_1\ell v_2, v_1\ell \models r_1, \text{ and } \ell v_2 \models r_2$
- $v \models \{r_1\}|\{r_2\} \iff v \models r_1 \text{ or } v \models r_2$
- $v \models \{r_1\} \&\& \{r_2\} \iff v \models r_1 \text{ and } v \models r_2$
- $v \models r[*0] \iff v = \epsilon$
- $v \models r[*] \iff \text{either } v \models r[*0] \text{ or } \exists v_1, v_2 \text{ s.t. } v_1 \neq \epsilon, v = v_1v_2, v_1 \models r \text{ and } v_2 \models r[*]$

B.2.1.1.2 Semantics of unlocked FL

We refer to a formula of FL with no $\mathcal{@}$ operator as an *unlocked formula*. Let v be a finite or infinite word, b be a boolean expression, r, r_1, r_2 unlocked SEREs, and φ, ψ unlocked FL formulas. We use \models to define the semantics of unlocked FL formulas:

If $v \models \varphi$ we say that v *models* (or *satisfies*) φ .

1. $v \models (\varphi) \iff v \models \varphi$
2. $v \models \neg\varphi \iff \bar{v} \not\models \varphi$
3. $v \models \varphi \wedge \psi \iff v \models \varphi \text{ and } v \models \psi$
4. $v \models b! \iff |v| > 0 \text{ and } v^0 \models b$
5. $v \models b \iff |v| = 0 \text{ or } v^0 \models b$
6. $v \models \{r\}! \iff \exists j < |v| \text{ s.t. } v^{0..j} \models r$
7. $v \models \{r\} \iff \forall j < |v|, v^{0..j}\top^\omega \models \{r\}!$
8. $v \models X! \varphi \iff |v| > 1 \text{ and } v^{1..} \models \varphi$
9. $v \models [\varphi U \psi] \iff \exists k < |v| \text{ s.t. } v^{k..} \models \psi, \text{ and } \forall j < k, v^{j..} \models \varphi$
10. $v \models \varphi \text{ abort } b \iff \text{either } v \models \varphi \text{ or } \exists j < |v| \text{ s.t. } v^j \models b \text{ and } v^{0..j-1}\top^\omega \models \varphi$
11. $v \models \{r\} \mapsto \varphi \iff \forall j < |v| \text{ s.t. } \bar{v}^{0..j} \models r, v^{j..} \models \varphi$

Notes:

1. The semantics given here for the LTL operator and the **abort** operator is equivalent to the truncated semantics given in [1] which is interpreted over 2^P rather than over $2^P \cup \{\top, \perp\}$. Using \models_\bullet for the semantics in [1] the following proposition states the equivalence: Let w be a finite word over 2^P , and let φ be a formula of LTL^{trunc}. Then the three following equivalences hold:

$$\begin{aligned} w \models_\bullet^- \varphi &\iff w\top^\omega \models \varphi \\ w \models_\bullet^+ \varphi &\iff w\perp^\omega \models \varphi \\ w \models_\bullet \varphi &\iff w \models \varphi \end{aligned}$$

2. There is a subtle difference between boolean negation and formula negation. For instance, consider the formula $\neg b$. If \neg is boolean negation, then $\neg b$ holds on an empty path. If \neg is formula negation, then $\neg b$ does not hold on an empty path. Rather than introduce distinct operators for boolean and formula negation, we instead adopt the convention that negation applied to a boolean expression is boolean negation. This does not restrict expressivity, as formula negation of b can be expressed as $(\neg b)!$.

B.2.1.2 Clocked Semantics

We say that finite word v is a clock tick of c iff $|v| > 0$ and $v^{|v|-1} \models c$ and for every natural number $i < |v| - 1$, $v^i \models \neg c$.

B.2.1.2.1 Semantics of clocked SEREs

Clocked SEREs are defined over finite words from the alphabet Σ and a boolean expression that serves as the clock context. The notation $v \models^c r$, where r is a SERE and c is a boolean expression, means that v models tightly r in context of clock c . The semantics of clocked SEREs are defined as follows, where b, c , and c_1 denote boolean expressions, r, r_1 , and r_2 denote clocked SEREs.

- $v \models^c \{r\} \iff v \models^c r$
- $v \models^c b \iff v$ is a clock tick of c and $v^{|v|-1} \models b$
- $v \models^c r_1; r_2 \iff \exists v_1, v_2$ s.t. $v = v_1 v_2$, $v_1 \models^c r_1$, and $v_2 \models^c r_2$
- $v \models^c \{r_1\} : \{r_2\} \iff \exists v_1, v_2$, and ℓ s.t. $v = v_1 \ell v_2$, $v_1 \ell \models^c r_1$, and $\ell v_2 \models^c r_2$
- $v \models^c \{r_1\} | \{r_2\} \iff v \models^c r_1$ or $v \models^c r_2$
- $v \models^c \{r_1\} \&\& \{r_2\} \iff v \models^c r_1$ and $v \models^c r_2$
- $v \models^c r[*0] \iff v = \epsilon$
- $v \models^c r[*] \iff$ either $v \models^c r[*0]$ or $\exists v_1, v_2$ s.t. $v_1 \neq \epsilon$, $v = v_1 v_2$, $v_1 \models^c r$ and $v_2 \models^c r[*]$
- $v \models^c r@c_1 \iff v \models^c r$

B.2.1.2.2 Semantics of clocked FL

The semantics of (clocked) FL formulas is defined with respect to finite/infinite words over Σ and a boolean expression c which serves as the clock context. Let v be a finite or infinite word, b, c, c_1 boolean expressions, r, r_1, r_2 SEREs, and φ, ψ FL formulas. We use \models^c to define the semantics of FL formulas. If $v \models^c \varphi$ we say that v models (or satisfies) φ in the context of clock c .

1. $v \models^c (\varphi) \iff v \models^c \varphi$
2. $v \models^c \neg \varphi \iff \bar{v} \not\models^c \varphi$
3. $v \models^c \varphi \wedge \psi \iff v \models^c \varphi$ and $v \models^c \psi$

4. $v \models^c b! \iff \exists j < |v|$ s.t. $v^{0..j}$ is a clock tick of c and $v^j \models b$
5. $v \models^c b \iff \forall j < |v|$ s.t. $\bar{v}^{0..j}$ is a clock tick of c , $v^j \models b$
6. $v \models^c \{r\}! \iff \exists j < |v|$ s.t. $v^{0..j} \models^c r$
7. $v \models^c \{r\} \iff \forall j < |v|$, $v^{0..j} \top^\omega \models^c \{r\}!$
8. $v \models^c X! f \iff \exists j < k < |v|$ s.t. $v^{0..j}$ and $v^{j+1..k}$ are clock ticks of c and $v^{k..} \models^c f$
9. $v \models^c [\varphi U \psi] \iff \exists k < |v|$ s.t. $v^k \models c$, $v^{k..} \models^c \psi$, and $\forall j < k$ s.t. $\bar{v}^j \models c$, $v^{j..} \models^c \varphi$
10. $v \models^c \varphi \text{ abort } b \iff$ either $v \models^c \varphi$ or $\exists j < |v|$ s.t. $v^j \models b$ and $v^{0..j-1} \top^\omega \models^c \varphi$
11. $v \models^c \varphi @_{c_1} \iff v \models^{c_1} \varphi$
12. $v \models^c \{r\} \mapsto \varphi \iff \forall j < |v|$ s.t. $\bar{v}^{0..j} \models^c r$, $v^{j..} \models^c \varphi$

Note:

The clocked semantics for the LTL subset follows the clocks paper [2], with the exception that strength is applied at the boolean level rather than at the propositional level.

B.2.2 Semantics of OBE formulas

The semantics of OBE formulas are defined over states in the *model*, rather than finite or infinite words. A model is a quintuple (S, S_0, R, P, L) , where S is a finite set of states, $S_0 \subseteq S$ is a set of initial states, $R \subseteq S \times S$ is the transition relation, P is a non-empty set of atomic propositions, and L is the valuation, a function $L : S \rightarrow 2^P$, mapping each state with a set of atomic propositions valid in that state.

A *path* π is a finite (or infinite) sequence of states $\pi = (\pi_0, \pi_1, \pi_2, \dots, \pi_n)$ (or $\pi = (\pi_0, \pi_1, \pi_2, \dots)$). A *computation path* π of a model M is a finite (or infinite) path π such that for every $i < n$, $R(\pi_i, \pi_{i+1})$ and for no s , $R(\pi_n, s)$ (or such that for every i , $R(\pi_i, \pi_{i+1})$). Given a finite (or infinite) path π , we define \hat{L} , an extension of the valuation function L from states to paths as follows: $\hat{L}(\pi) = L(\pi_0)L(\pi_1) \dots L(\pi_n)$ (or $\hat{L}(\pi) = L(\pi_0)L(\pi_1) \dots$). Thus we have a mapping from states in M to letters of 2^P , and from finite (or infinite) sequences of states in M to finite (or infinite) words over 2^P .

The semantics of OBE formulas are defined inductively, using as the base case the semantics of *boolean expressions* over *letters* in 2^P . The semantics of boolean expression is assumed to be given as a relation $\models \subseteq 2^P \times B$ relating letters in 2^P with boolean expressions in B . If $(\ell, b) \in \models$ we say that the letter ℓ *satisfies* the boolean expression b and denote it $\ell \models b$. We assume that the boolean relation \models behaves in the usual manner. In particular, that for every letter $\ell \in 2^P$, atomic proposition $p \in P$ and boolean expressions $b, b_1, b_2 \in B$ (i) $\ell \models p$ iff $p \in \ell$, (ii) $\ell \models \neg b$ iff $\ell \not\models b$, (iii) $\ell \models b_1 \wedge b_2$ iff $\ell \models b_1$ and $\ell \models b_2$, and (iv) $\ell \models \text{true}$ and $\ell \not\models \text{false}$.

The notation $M, s \models f$ means that formula f holds in state s of model M . The notation $M \models f$ is equivalent to $\forall s \in S_0 : M, s \models f$. In other words, f is valid for every initial state of M . The semantics of an OBE formula are defined as follows¹, where b denotes a boolean expression and f, f_1 , and f_2 denote OBE formulas.

¹ The semantics are those of standard CTL.

- $M, s \models b \iff L(s) \models b$
- $M, s \models (f) \iff M, s \models f$
- $M, s \models \neg f \iff M, s \not\models f$
- $M, s \models f_1 \wedge f_2 \iff M, s \models f_1 \text{ and } M, s \models f_2$
- $M, s \models EX f \iff$ there exists a computation path π of M such that $|\pi| > 1$, $\pi_0 = s$, and $M, \pi_1 \models f$
- $M, s \models E[f_1 U f_2] \iff$ there exists a computation path π of M such that $\pi_0 = s$ and there exists $k < |\pi|$ such that $M, \pi_k \models f_2$ and for every j such that $j < k$: $M, \pi_j \models f_1$
- $M, s \models EG f \iff$ there exists a computation path π of M such that $\pi_0 = s$ and for every j such that $0 \leq j < |\pi|$: $M, \pi_j \models f$

B.3 Syntactic Sugaring

The remainder of the temporal layer is syntactic sugar. In other words, it does not add expressive power, and every piece of syntactic sugar can be defined in terms of the basic FL operators presented above. The syntactic sugar is defined below.

Note: the definitions given here do not necessarily represent the most efficient implementation. In some cases, there is an equivalent syntactic sugaring, or a direct implementation, that is more efficient.

B.3.1 Additional SERE operators

If i, j, k , and l are integer constants such that $i \geq 0, j \geq i, k \geq 1$ and $l \geq k$, then additional SERE operators can be viewed as abbreviations of the basic SERE operators defined above, as follows, where b denotes a boolean expression, and r denotes a SERE.

- $\{r_1\} \& \{r_2\} \stackrel{\text{def}}{=} \{\{r_1\} \&\& \{r_2; \text{true}[*]\}\} \mid \{\{r_1; \text{true}[*]\} \&\& \{r_2\}\}$
- $r[+] \stackrel{\text{def}}{=} r; r[*]$
- $r[*k] \stackrel{\text{def}}{=} \overbrace{r; r; \dots; r}^{k \text{ times}}$
- $r[*i..j] \stackrel{\text{def}}{=} \{r[*i]\} \dots \{r[*j]\}$
- $r[*i..] \stackrel{\text{def}}{=} \{r[*i]\}; \{r[*]\}$
- $r[*..i] \stackrel{\text{def}}{=} \{r[*0]\} \dots \{r[*i]\}$
- $r[*..] \stackrel{\text{def}}{=} r[*0..]$
- $[+] \stackrel{\text{def}}{=} \text{true}[+]$
- $[*] \stackrel{\text{def}}{=} \text{true}[*]$
- $[*i] \stackrel{\text{def}}{=} \text{true}[*i]$
- $[*i..j] \stackrel{\text{def}}{=} \text{true}[*i..j]$
- $[*i..] \stackrel{\text{def}}{=} \text{true}[*i..]$
- $[*..i] \stackrel{\text{def}}{=} \text{true}[*..i]$
- $[*..] \stackrel{\text{def}}{=} \text{true}[*..]$

- $b[= i] \stackrel{\text{def}}{=} \{\neg b[*]; b\}[*i]; \neg b[*]$
- $b[= i..j] \stackrel{\text{def}}{=} \{b[= i]\}|\dots|\{b[= j]\}$
- $b[= i..] \stackrel{\text{def}}{=} b[= i]; [*]$
- $b[= ..i] \stackrel{\text{def}}{=} \{b[= 0]\}|\dots|\{b[= i]\}$
- $b[= ..] \stackrel{\text{def}}{=} b[= 0..]$
- $b[\rightarrow] \stackrel{\text{def}}{=} \neg b[*]; b$
- $b[\rightarrow k] \stackrel{\text{def}}{=} \{\neg b[*]; b\}[*k]$
- $b[\rightarrow k..l] \stackrel{\text{def}}{=} \{b[\rightarrow k]\}|\dots|\{b[\rightarrow l]\}$
- $b[\rightarrow k..] \stackrel{\text{def}}{=} \{b[\rightarrow k]\}|\{b[\rightarrow k]; [*]; b\}$
- $b[\rightarrow ..k] \stackrel{\text{def}}{=} \{b[\rightarrow 1]\}|\dots|\{b[\rightarrow k]\}$
- $b[\rightarrow ..] \stackrel{\text{def}}{=} b[\rightarrow 1..]$
- $r_1 \text{ within } r_2 \stackrel{\text{def}}{=} \{[*]; r_1; [*]\} \&\& \{r_2\}$

B.3.2 Additional FL operators

If i, j, k and l are integers such that $i \geq 0$, $j \geq i$, $k > 0$ and $l \geq k$ then additional operators can be viewed as abbreviations of the basic operators defined above, as follows, where b denotes a boolean expression, r , r_1 , and r_2 denote SEREs, and φ , φ_1 , and φ_2 denote FL formulas.

- $\varphi_1 \vee \varphi_2 \stackrel{\text{def}}{=} \neg(\neg\varphi_1 \wedge \neg\varphi_2)$
- $\varphi_1 \rightarrow \varphi_2 \stackrel{\text{def}}{=} \neg\varphi_1 \vee \varphi_2$
- $\varphi_1 \leftrightarrow \varphi_2 \stackrel{\text{def}}{=} (\varphi_1 \rightarrow \varphi_2) \wedge (\varphi_2 \rightarrow \varphi_1)$

- $F\varphi \stackrel{\text{def}}{=} [\text{true } U \varphi]$
- $G\varphi \stackrel{\text{def}}{=} \neg F\neg\varphi$
- $X\varphi \stackrel{\text{def}}{=} \neg X! \neg\varphi$
- $[\varphi_1 W \varphi_2] \stackrel{\text{def}}{=} [\varphi_1 U \varphi_2] \vee G\varphi_1$

- $\text{always } \varphi \stackrel{\text{def}}{=} G \varphi$
- $\text{never } \varphi \stackrel{\text{def}}{=} G \neg\varphi$
- $\text{next! } \varphi \stackrel{\text{def}}{=} X! \varphi$
- $\text{next } \varphi \stackrel{\text{def}}{=} X \varphi$
- $\text{eventually! } \varphi \stackrel{\text{def}}{=} F\varphi$

- $\varphi_1 \text{ until! } \varphi_2 \stackrel{\text{def}}{=} [\varphi_1 U \varphi_2]$
- $\varphi_1 \text{ until } \varphi_2 \stackrel{\text{def}}{=} [\varphi_1 W \varphi_2]$
- $\varphi_1 \text{ until!}_- \varphi_2 \stackrel{\text{def}}{=} [\varphi_1 U \varphi_1 \wedge \varphi_2]$
- $\varphi_1 \text{ until}_- \varphi_2 \stackrel{\text{def}}{=} [\varphi_1 W \varphi_1 \wedge \varphi_2]$

- $\varphi_1 \text{ before! } \varphi_2 \stackrel{\text{def}}{=} [\neg\varphi_2 U \varphi_1 \wedge \neg\varphi_2]$
- $\varphi_1 \text{ before } \varphi_2 \stackrel{\text{def}}{=} [\neg\varphi_2 W \varphi_1 \wedge \neg\varphi_2]$
- $\varphi_1 \text{ before!}_- \varphi_2 \stackrel{\text{def}}{=} [\neg\varphi_2 U \varphi_1]$
- $\varphi_1 \text{ before}_- \varphi_2 \stackrel{\text{def}}{=} [\neg\varphi_2 W \varphi_1]$

- $X! [i]\varphi \stackrel{\text{def}}{=} \overbrace{X! X! \dots X!}^{i \text{ times}} \varphi$
- $X [i]\varphi \stackrel{\text{def}}{=} \overbrace{X X \dots X}^{i \text{ times}} \varphi$
- $\text{next!}[i] \varphi \stackrel{\text{def}}{=} X! [i] \varphi$
- $\text{next}[i] \varphi \stackrel{\text{def}}{=} X [i] \varphi$
- $\text{next}_a![i..j]\varphi \stackrel{\text{def}}{=} (X![i]\varphi) \wedge \dots \wedge (X![j]\varphi)$
- $\text{next}_a[i..j]\varphi \stackrel{\text{def}}{=} (X [i]\varphi) \wedge \dots \wedge (X [j]\varphi)$
- $\text{next}_e![i..j]\varphi \stackrel{\text{def}}{=} (X![i]\varphi) \vee \dots \vee (X![j]\varphi)$
- $\text{next}_e[i..j]\varphi \stackrel{\text{def}}{=} (X [i]\varphi) \vee \dots \vee (X [j]\varphi)$

- $\text{next_event!}(b)(\varphi) \stackrel{\text{def}}{=} [\neg b U b \wedge \varphi]$
- $\text{next_event}(b)(\varphi) \stackrel{\text{def}}{=} [\neg b W b \wedge \varphi]$

- $\text{next_event!}(b)[k](\varphi) \stackrel{\text{def}}{=} \text{next_event!}(b) \overbrace{(X! \text{next_event!}(b) \dots (X! \text{next_event!}(b)(\varphi)) \dots)}^{k-1 \text{ times}}$
- $\text{next_event}(b)[k](\varphi) \stackrel{\text{def}}{=} \text{next_event}(b) \overbrace{(X \text{next_event}(b) \dots (X \text{next_event}(b)(\varphi)) \dots)}^{k-1 \text{ times}}$
- $\text{next_event}_a!(b)[k..l](\varphi) \stackrel{\text{def}}{=} \text{next_event!}(b)[k](\varphi) \wedge \dots \wedge \text{next_event!}(b)[l](\varphi)$
- $\text{next_event}_a(b)[k..l](\varphi) \stackrel{\text{def}}{=} \text{next_event}(b)[k](\varphi) \wedge \dots \wedge \text{next_event}(b)[l](\varphi)$
- $\text{next_event}_e!(b)[k..l](\varphi) \stackrel{\text{def}}{=} \text{next_event!}(b)[k](\varphi) \vee \dots \vee \text{next_event!}(b)[l](\varphi)$
- $\text{next_event}_e(b)[k..l](\varphi) \stackrel{\text{def}}{=} \text{next_event}(b)[k](\varphi) \vee \dots \vee \text{next_event}(b)[l](\varphi)$

- $\{r\}(\varphi) \stackrel{\text{def}}{=} \{r\} \mapsto \varphi$
- $\{r\} \Rightarrow \varphi \stackrel{\text{def}}{=} \{r; \text{true}\} \mapsto \varphi$

B.3.4 Forall

If f is an Accellera PSL formula, v_0, v_1, \dots, v_n are constants, and j, k, l and m are integers, then the following are Accellera PSL formulas:

- $\text{forall } i \text{ in } \{v_0, v_1, \dots, v_n\} : f$
- $\text{forall } i \text{ in } j..k : f$
- $\text{forall } i \text{ in boolean} : f$
- $\text{forall } i \langle l..m \rangle \text{ in } \{v_0, v_1, \dots, v_n\} : f$
- $\text{forall } i \langle l..m \rangle \text{ in } j..k : f$
- $\text{forall } i \langle l..m \rangle \text{ in boolean} : f$

forall does not add expressive power. Rather, it can be viewed as additional syntactic sugar, as follows:

$$\begin{aligned}
& - \text{forall } i \text{ in } \{v_0, v_1, \dots, v_n\} : f \stackrel{\text{def}}{=} \bigwedge_{u \in \{v_0, v_1, \dots, v_n\}} f[i \leftarrow u] \\
& - \text{forall } i \text{ in } j..k : f \stackrel{\text{def}}{=} \bigwedge_{u=j}^k f[i \leftarrow u] \\
& - \text{forall } i \text{ in boolean} : f \stackrel{\text{def}}{=} \bigwedge_{u=0}^1 f[i \leftarrow u] \\
& - \text{forall } i\langle l..m \rangle \text{ in } \{v_0, v_1, \dots, v_n\} : f \stackrel{\text{def}}{=} \bigwedge_{u_l \in \{v_0, v_1, \dots, v_n\}} \dots \bigwedge_{u_m \in \{v_0, v_1, \dots, v_n\}} f[i\langle l..m \rangle \leftarrow \langle u_l..u_m \rangle] \\
& - \text{forall } i\langle l..m \rangle \text{ in } j..k : f \stackrel{\text{def}}{=} \bigwedge_{u_l=j}^k \dots \bigwedge_{u_m=j}^k f[i\langle l..m \rangle \leftarrow \langle u_l..u_m \rangle] \\
& - \text{forall } i\langle l..m \rangle \text{ in boolean} : f \stackrel{\text{def}}{=} \bigwedge_{u_l=0}^1 \dots \bigwedge_{u_m=0}^1 f[i\langle l..m \rangle \leftarrow \langle u_l..u_m \rangle]
\end{aligned}$$

where $f[i \leftarrow u]$ is the formula obtained from f by replacing every occurrence of i by u and $f[i\langle l..m \rangle \leftarrow \langle u_l..u_m \rangle]$ is the formula obtained from f by replacing every occurrence of i_j with u_j .

B.4 Typed-text representation of symbols

Table 1 shows the mapping of various symbols used in this definition to the corresponding typed-text Sugar representation.

	Verilog	VHDL	EDL
\mapsto	->	->	->
\mapRightarrow	=>	=>	=>
\rightarrow	->	->	->
\leftrightarrow	<->	<->	<->
\neg	!	not	!
\wedge	&&	and	&
\vee		or	
$..$:	to	..
$\langle \rangle$	[]	()	()

Table 1. Typed-text symbols in the Verilog, VHDL, and EDL flavors

B.5 Rewriting rules for clocks

In Section B.2.2 we gave the semantics of clocked FL formulas directly. There is an equivalent definition in terms of unclocked FL formulas, as follows: Starting from the outermost clock, use the following rules to translate clocked SEREs into unclocked SEREs, and clocked FL formulas into unclocked FL formulas.

The rewrite rules for SEREs are:

1. $\mathcal{R}^c(\{r\}) = \{\mathcal{R}^c(r)\}$
2. $\mathcal{R}^c(b) = \{\neg c[*]; c \wedge b\}$
3. $\mathcal{R}^c(r_1 ; r_2) = \mathcal{R}^c(r_1) ; \mathcal{R}^c(r_2)$
4. $\mathcal{R}^c(\{r_1\} : \{r_2\}) = \{\mathcal{R}^c(r_1)\} : \{\mathcal{R}^c(r_2)\}$
5. $\mathcal{R}^c(\{r_1\} \mid \{r_2\}) = \{\mathcal{R}^c(r_1)\} \mid \{\mathcal{R}^c(r_2)\}$
6. $\mathcal{R}^c(\{r_1\} \&\& \{r_2\}) = \{\mathcal{R}^c(r_1)\} \&\& \{\mathcal{R}^c(r_2)\}$
7. $\mathcal{R}^c(r[*0]) = \{\mathcal{R}^c(r)\}[*0]$
8. $\mathcal{R}^c(r[*]) = \{\mathcal{R}^c(r)\}[*]$
9. $\mathcal{R}^c(r@c_1) = \mathcal{R}^{c_1}(r)$

The rewrite rules for FL formulas are:

1. $\mathcal{F}^c((\varphi)) = (\mathcal{F}^c(\varphi))$
2. $\mathcal{F}^c(b!) = [\neg c \ U \ (c \wedge b)]$
3. $\mathcal{F}^c(b) = [\neg c \ W \ (c \wedge b)]$
4. $\mathcal{F}^c(\neg\varphi) = \neg\mathcal{F}^c(\varphi)$
5. $\mathcal{F}^c(\varphi \wedge \psi) = (\mathcal{F}^c(\varphi) \wedge \mathcal{F}^c(\psi))$
6. $\mathcal{F}^c(X!\varphi) = [\neg c \ U \ (c \wedge X! [\neg c \ U \ (c \wedge \mathcal{F}^c(\varphi))])]$
7. $\mathcal{F}^c(\varphi \ U \ \psi) = [(c \rightarrow \mathcal{F}^c(\varphi)) \ U \ (c \wedge \mathcal{F}^c(\psi))]$
8. $\mathcal{F}^c(\varphi \ \text{abort} \ b) = \mathcal{F}^c(\varphi) \ \text{abort} \ b$
9. $\mathcal{F}^c(\varphi@c_1) = \mathcal{F}^{c_1}(\varphi)$
10. $\mathcal{F}^c(\{r\} \mapsto \varphi) = \{\mathcal{R}^c(r)\} \mapsto \mathcal{F}^c(\varphi)$
11. $\mathcal{F}^c(\{r\}!) = \{\mathcal{R}^c(r)\}!$
12. $\mathcal{F}^c(\{r\}) = \{\mathcal{R}^c(r)\}$

References

1. C. Eisner, D. Fisman, J. Havlicek, Y. Lustig, A. McIsaac, and D. Van Campenhout. Reasoning with temporal logic on truncated paths. In *The 15th international conference on computer aided verification (CAV'03)*, LNCS 2725, pages 27–40, Boulder, CO, USA, July 2003. Springer-Verlag.
2. C. Eisner, D. Fisman, J. Havlicek, A. McIsaac, and D. Van Campenhout. The definition of a temporal clock operator. In *Proc. 30th Int. Colloq. Aut. Lang. Prog. (ICALP'03)*, LNCS 2719, pages 857–870. Springer-Verlag, June 2003.