Koch Trainer: A Z Specification

Formal Model of Morse Code Learning Progress

January 2026

Contents

1 Introduction

Koch Trainer teaches Morse code using the *Koch method*—a proven technique developed by German psychologist Ludwig Koch in the 1930s. Rather than learning Morse at slow speeds and gradually increasing, students learn characters at full speed (20 words per minute) from the start, adding one new character at a time after achieving 90% accuracy.

1.1 Design Philosophy

The application follows four core principles:

1. **Audio-first**: Morse code is fundamentally an auditory skill. Visual aids support learning, but audio remains primary.
2. **Spaced repetition**: Practice intervals adapt to performance, maximizing long-term retention while minimizing daily time commitment.
3. **Separate skills**: Receiving (copying audio to text) and sending (keying Morse from text) are distinct skills that progress independently.
4. **Realistic simulation**: QSO practice mirrors real amateur radio operation with authentic timing and protocols.

1.2 Scope of This Specification

This formal specification models the stateful components of the learning system:

- Student progress through 26 Koch-ordered characters
- Per-character accuracy statistics by training direction
- Spaced repetition scheduling with adaptive intervals
- Streak tracking for consecutive practice days

We do not model audio generation, UI state, or real-time session mechanics—only the persistent state that survives between sessions.

2 Basic Types

We introduce given sets for entities whose internal structure is irrelevant to the learning model.

$[SESSIONID, TIMESTAMP]$

SESSIONID uniquely identifies completed training sessions. *TIMESTAMP* represents points in time for scheduling and history tracking.

Note: We do not model individual Morse characters as a given set. The 26 letters follow a fixed Koch order (K M R S U A P T L O W I N J E F Y V G Q Z H B C D X), and level numbers 1–26 implicitly identify the character set unlocked at each level.

3 Free Types

3.1 Training Direction

Training occurs in two distinct directions, each developing a different skill:

$Direction ::= receive \mid send$

In *receive* mode, the student hears Morse audio and types the corresponding letter—training auditory pattern recognition. In *send* mode, the student sees a letter and keys the Morse pattern using dit/dah inputs—training motor memory and timing.

These skills progress independently. A student may excel at receiving while struggling with sending, or vice versa. Separate tracking prevents frustration and allows focused practice on weaker skills.

3.2 Boolean Type

We define a boolean type for input flags, using names that avoid conflicts with ProB's reserved keywords:

$$ZBOOL ::= ztrue \mid zfalse$$

3.3 Radio Mode

Amateur radio is half-duplex: the operator either receives (listens to incoming signals) or transmits (sends outgoing signals), but not both simultaneously. This fundamental constraint shapes the audio experience during training.

$$RadioMode ::= radioOff \mid radioReceiving \mid radioTransmitting$$

- *radioOff*: Radio not active (session not started, paused, or completed)
- *radioReceiving*: Radio on, listening—continuous noise floor with band conditions (QRN, QSB, QRM), incoming Morse signals audible
- *radioTransmitting*: Radio on, sending—sidetone only, receiver muted (no noise or band conditions)

When the radio is on and the operator is not transmitting, they are receiving. This models real amateur radio behavior where the receiver is always active unless explicitly transmitting.

4 Global Constants

The Koch method uses exactly 26 letters introduced in a specific order based on Ludwig Koch's research on optimal learning sequences. Characters are introduced one at a time: level 1 unlocks only K, level 2 adds M, and so on.

$$\begin{array}{|l}
maxLevel : \\
minLevel : \\
maxInterval : \\
minInterval : \\
advanceThreshold : \\
habitFormationDays : \\
habitMaxInterval : \\
missedPracticeMultiplier : \\
\hline
maxLevel = 26 \\
minLevel = 1 \\
maxInterval = 30 \\
minInterval = 1 \\
advanceThreshold = 90 \\
habitFormationDays = 14 \\
habitMaxInterval = 2 \\
missedPracticeMultiplier = 2
\end{array}$$

The advancement threshold of 90% accuracy over a minimum of 20 attempts ensures students have truly mastered current characters before encountering new ones. This high bar is intentional: the Koch method's effectiveness depends on learning characters correctly at full speed from the start.

5 Character Statistics

The system tracks per-character accuracy separately for each training direction. This granular tracking serves multiple purposes:

- Identify specific characters that need more practice
- Display proficiency indicators (color-coded rings) on the character grid

- Enable future features like adaptive character selection

CharacterStat

$receiveAttempts :$
$receiveCorrect :$
$sendAttempts :$
$sendCorrect :$
$earAttempts :$
$earCorrect :$

$receiveCorrect \leq receiveAttempts$
$sendCorrect \leq sendAttempts$
$earCorrect \leq earAttempts$
$receiveAttempts \leq 10000$
$sendAttempts \leq 10000$
$earAttempts \leq 10000$

The invariants ensure that correct counts never exceed attempt counts—a fundamental data integrity property. Upper bounds support finite model checking.

The *earAttempts* and *earCorrect* fields track ear training mode, where students reproduce heard patterns rather than identify letters.

Combined accuracy for display purposes is computed as:

$$accuracy = \frac{receiveCorrect + sendCorrect}{receiveAttempts + sendAttempts}$$

6 Session Result

Each completed training session produces an immutable result record.

SessionResult

$sessionId : SESSIONID$
$direction : Direction$
$totalAttempts :$
$correctCount :$
$timestamp : TIMESTAMP$

$correctCount \leq totalAttempts$
$totalAttempts \leq 1000$

Session results form the historical record used for:

- Calculating streak continuity (did the student practice yesterday?)
- Tracking long-term progress trends
- Supporting future analytics features

Only *learn mode* sessions (not custom practice or vocabulary drills) affect level advancement and spaced repetition intervals.

7 Student Progress

The central state schema captures all persistent student progress. We flatten the schedule fields into the main state for ProB animation compatibility.

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{State}} \\
\hline
receiveLevel : \\
sendLevel : \\
earLevel : \\
sessionCount : \\
receiveInterval : \\
sendInterval : \\
currentStreak : \\
longestStreak : \\
\hline
receiveLevel \geq minLevel \\
receiveLevel \leq maxLevel \\
sendLevel \geq minLevel \\
sendLevel \leq maxLevel \\
earLevel \geq 1 \\
earLevel \leq 5 \\
sessionCount \leq 10000 \\
receiveInterval \geq minInterval \\
receiveInterval \leq maxInterval \\
sendInterval \geq minInterval \\
sendInterval \leq maxInterval \\
currentStreak \leq longestStreak \\
longestStreak \leq 365 \\
\hline
\end{array}$$

7.1 Level Semantics

Each level n unlocks the first n characters in Koch order:

- Level 1: K
- Level 2: K, M
- Level 3: K, M, R
- ⋮
- Level 26: All 26 letters

Receive and send levels progress independently. A student at receive level 15 and send level 8 can copy 15 characters but only send 8.

7.2 Interval Semantics

The *receiveInterval* and *sendInterval* fields represent the number of days until the next scheduled practice for each direction. These intervals adapt based on performance, implementing a simplified spaced repetition algorithm.

7.3 Streak Semantics

The streak tracks consecutive *calendar days* with at least one completed session (either direction counts). The longest streak persists as a personal record for motivation.

8 Initialization

A new student begins at level 1 in all directions with 1-day intervals and no streak history.

$$\begin{array}{|l}
\textit{Init} \\
\hline
\mathit{State}' \\
\hline
\mathit{receiveLevel}' = \mathit{minLevel} \\
\mathit{sendLevel}' = \mathit{minLevel} \\
\mathit{earLevel}' = 1 \\
\mathit{sessionCount}' = 0 \\
\mathit{receiveInterval}' = \mathit{minInterval} \\
\mathit{sendInterval}' = \mathit{minInterval} \\
\mathit{currentStreak}' = 0 \\
\mathit{longestStreak}' = 0 \\
\hline
\end{array}$$

9 Operations

9.1 Level Advancement

The Koch method's core progression: when a student achieves $\geq 90\%$ accuracy over at least 20 attempts in a session, they advance to the next level, unlocking one additional character.

$$\begin{array}{|l}
\textit{AdvanceReceiveLevel} \\
\hline
\Delta \mathit{State} \\
\mathit{accuracy}? : \\
\hline
\mathit{accuracy}? \geq \mathit{advanceThreshold} \\
\mathit{accuracy}? \leq 100 \\
\mathit{receiveLevel} < \mathit{maxLevel} \\
\mathit{receiveLevel}' = \mathit{receiveLevel} + 1 \\
\mathit{sendLevel}' = \mathit{sendLevel} \\
\mathit{earLevel}' = \mathit{earLevel} \\
\mathit{sessionCount}' = \mathit{sessionCount} \\
\mathit{receiveInterval}' = \mathit{receiveInterval} \\
\mathit{sendInterval}' = \mathit{sendInterval} \\
\mathit{currentStreak}' = \mathit{currentStreak} \\
\mathit{longestStreak}' = \mathit{longestStreak} \\
\hline
\end{array}$$

$$\begin{array}{|l}
\textit{AdvanceSendLevel} \\
\hline
\Delta \mathit{State} \\
\mathit{accuracy}? : \\
\hline
\mathit{accuracy}? \geq \mathit{advanceThreshold} \\
\mathit{accuracy}? \leq 100 \\
\mathit{sendLevel} < \mathit{maxLevel} \\
\mathit{sendLevel}' = \mathit{sendLevel} + 1 \\
\mathit{receiveLevel}' = \mathit{receiveLevel} \\
\mathit{earLevel}' = \mathit{earLevel} \\
\mathit{sessionCount}' = \mathit{sessionCount} \\
\mathit{receiveInterval}' = \mathit{receiveInterval} \\
\mathit{sendInterval}' = \mathit{sendInterval} \\
\mathit{currentStreak}' = \mathit{currentStreak} \\
\mathit{longestStreak}' = \mathit{longestStreak} \\
\hline
\end{array}$$

The precondition $\mathit{receiveLevel} < \mathit{maxLevel}$ (or $\mathit{sendLevel} < \mathit{maxLevel}$) ensures we cannot advance beyond the 26 available characters.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{AdvanceEarLevel}} \\
\Delta State \\
accuracy? : \\
\hline
accuracy? \geq advanceThreshold \\
accuracy? \leq 100 \\
earLevel < 5 \\
earLevel' = earLevel + 1 \\
receiveLevel' = receiveLevel \\
sendLevel' = sendLevel \\
sessionCount' = sessionCount \\
receiveInterval' = receiveInterval \\
sendInterval' = sendInterval \\
currentStreak' = currentStreak \\
longestStreak' = longestStreak \\
\hline
\end{array}
$$

Ear training levels (1–5) correspond to pattern length: level 1 covers single-element patterns (E, T), level 5 includes 5-element patterns like digits. Unlike receive/send, ear training does not affect spaced repetition scheduling or streaks.

9.2 Session Recording

Each completed session increments the session count for analytics.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{RecordSession}} \\
\Delta State \\
\hline
sessionCount' = sessionCount + 1 \\
receiveLevel' = receiveLevel \\
sendLevel' = sendLevel \\
earLevel' = earLevel \\
receiveInterval' = receiveInterval \\
sendInterval' = sendInterval \\
currentStreak' = currentStreak \\
longestStreak' = longestStreak \\
\hline
\end{array}
$$

9.3 Spaced Repetition

The interval algorithm adapts practice frequency based on performance:

Accuracy	Interval Change
$\geq$ 90%	Double interval (max 30 days)
70%–89%	No change
< 70%	Reset to 1 day

High accuracy indicates mastery, so we increase the interval to avoid over-practice. Low accuracy signals struggle, so we reset to daily practice. The middle range maintains the current schedule.

$UpdateReceiveIntervalHigh$

$\Delta State$
$accuracy? :$
$daysSinceStart? :$

$accuracy? \geq advanceThreshold$
$accuracy? \leq 100$
$\mathbf{let}\ effectiveMax == \mathbf{if}\ daysSinceStart? < habitFormationDays\ \mathbf{then}\ habitMaxInterval\ \mathbf{else}\ maxInterval \bullet (recei$
$sendInterval' = sendInterval$
$currentStreak' = currentStreak$
$longestStreak' = longestStreak$
$receiveLevel' = receiveLevel$
$sendLevel' = sendLevel$
$earLevel' = earLevel$
$sessionCount' = sessionCount$

$UpdateReceiveIntervalMedium$

$\Delta State$
$accuracy? :$

$accuracy? \geq 70$
$accuracy? < advanceThreshold$
$receiveInterval' = receiveInterval$
$sendInterval' = sendInterval$
$currentStreak' = currentStreak$
$longestStreak' = longestStreak$
$receiveLevel' = receiveLevel$
$sendLevel' = sendLevel$
$earLevel' = earLevel$
$sessionCount' = sessionCount$

$UpdateReceiveIntervalLow$

$\Delta State$
$accuracy? :$

$accuracy? < 70$
$receiveInterval' = minInterval$
$sendInterval' = sendInterval$
$currentStreak' = currentStreak$
$longestStreak' = longestStreak$
$receiveLevel' = receiveLevel$
$sendLevel' = sendLevel$
$earLevel' = earLevel$
$sessionCount' = sessionCount$

The three interval schemas partition the accuracy range: high ($\geq$ 90%), medium (70%–89%), and low ($<$ 70%). The *daysSinceStart?* input allows the habit formation cap to be applied during the first 14 days.

Symmetric operations exist for send training:

UpdateSendIntervalHigh

$\Delta State$
$accuracy? :$
$daysSinceStart? :$

$accuracy? \geq advanceThreshold$
$accuracy? \leq 100$
$\textbf{let } effectiveMax == \textbf{if } daysSinceStart? < habitFormationDays \textbf{ then } habitMaxInterval \textbf{ else } maxInterval \bullet (send$
$receiveInterval' = receiveInterval$
$currentStreak' = currentStreak$
$longestStreak' = longestStreak$
$receiveLevel' = receiveLevel$
$sendLevel' = sendLevel$
$earLevel' = earLevel$
$sessionCount' = sessionCount$

UpdateSendIntervalMedium

$\Delta State$
$accuracy? :$

$accuracy? \geq 70$
$accuracy? < advanceThreshold$
$sendInterval' = sendInterval$
$receiveInterval' = receiveInterval$
$currentStreak' = currentStreak$
$longestStreak' = longestStreak$
$receiveLevel' = receiveLevel$
$sendLevel' = sendLevel$
$earLevel' = earLevel$
$sessionCount' = sessionCount$

UpdateSendIntervalLow

$\Delta State$
$accuracy? :$

$accuracy? < 70$
$sendInterval' = minInterval$
$receiveInterval' = receiveInterval$
$currentStreak' = currentStreak$
$longestStreak' = longestStreak$
$receiveLevel' = receiveLevel$
$sendLevel' = sendLevel$
$earLevel' = earLevel$
$sessionCount' = sessionCount$

9.4 Missed Practice Reset

If a student misses practice for more than twice their current interval, skill decay is assumed and the interval resets to daily practice.

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{ResetReceiveIntervalOnMissedPractice}} \\ \hline
\Delta \mathit{State} \\
\mathit{daysSinceLastPractice}? : \\ \hline
\mathit{daysSinceLastPractice}? > \mathit{receiveInterval} * \mathit{missedPracticeMultiplier} \\
\mathit{receiveInterval}' = \mathit{minInterval} \\
\mathit{sendInterval}' = \mathit{sendInterval} \\
\mathit{currentStreak}' = \mathit{currentStreak} \\
\mathit{longestStreak}' = \mathit{longestStreak} \\
\mathit{receiveLevel}' = \mathit{receiveLevel} \\
\mathit{sendLevel}' = \mathit{sendLevel} \\
\mathit{earLevel}' = \mathit{earLevel} \\
\mathit{sessionCount}' = \mathit{sessionCount} \\ \hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{ResetSendIntervalOnMissedPractice}} \\ \hline
\Delta \mathit{State} \\
\mathit{daysSinceLastPractice}? : \\ \hline
\mathit{daysSinceLastPractice}? > \mathit{sendInterval} * \mathit{missedPracticeMultiplier} \\
\mathit{sendInterval}' = \mathit{minInterval} \\
\mathit{receiveInterval}' = \mathit{receiveInterval} \\
\mathit{currentStreak}' = \mathit{currentStreak} \\
\mathit{longestStreak}' = \mathit{longestStreak} \\
\mathit{receiveLevel}' = \mathit{receiveLevel} \\
\mathit{sendLevel}' = \mathit{sendLevel} \\
\mathit{earLevel}' = \mathit{earLevel} \\
\mathit{sessionCount}' = \mathit{sessionCount} \\ \hline
\end{array}$$

9.5 Streak Tracking

Streaks measure consecutive calendar days with at least one completed session. They provide motivation through gamification without affecting learning outcomes.

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{IncrementStreak}} \\ \hline
\Delta \mathit{State} \\ \hline
\mathit{currentStreak}' = \mathit{currentStreak} + 1 \\
(\mathit{currentStreak} + 1 > \mathit{longestStreak} \land \mathit{longestStreak}' = \mathit{currentStreak} + 1) \lor (\mathit{currentStreak} + 1 \leq \mathit{longestStreak} \land \\
\mathit{receiveInterval}' = \mathit{receiveInterval} \\
\mathit{sendInterval}' = \mathit{sendInterval} \\
\mathit{receiveLevel}' = \mathit{receiveLevel} \\
\mathit{sendLevel}' = \mathit{sendLevel} \\
\mathit{earLevel}' = \mathit{earLevel} \\
\mathit{sessionCount}' = \mathit{sessionCount} \\ \hline
\end{array}$$

The streak increments when the student practices on the calendar day following their last practice. The longest streak updates if the current streak exceeds it.

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{ResetStreak}} \\ \hline
\Delta \mathit{State} \\ \hline
\mathit{currentStreak}' = 0 \\
\mathit{longestStreak}' = \mathit{longestStreak} \\
\mathit{receiveInterval}' = \mathit{receiveInterval} \\
\mathit{sendInterval}' = \mathit{sendInterval} \\
\mathit{receiveLevel}' = \mathit{receiveLevel} \\
\mathit{sendLevel}' = \mathit{sendLevel} \\
\mathit{earLevel}' = \mathit{earLevel} \\
\mathit{sessionCount}' = \mathit{sessionCount} \\ \hline
\end{array}$$

The streak resets to zero when a calendar day passes without any practice. Note that the longest streak is preserved—it represents the all-time record.

9.6 Attempt Recording

During training, each character attempt updates the per-character statistics.

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{RecordReceiveAttempt}} \\
\hline
\Delta \mathit{CharacterStat} \\
\mathit{correct?} : \mathit{ZBOOL} \\
\hline
\mathit{receiveAttempts}' = \mathit{receiveAttempts} + 1 \\
(\mathit{correct?} = \mathit{ztrue} \land \mathit{receiveCorrect}' = \mathit{receiveCorrect} + 1) \lor (\mathit{correct?} = \mathit{zfalse} \land \mathit{receiveCorrect}' = \mathit{receiveCorrect} \\
\mathit{sendAttempts}' = \mathit{sendAttempts} \\
\mathit{sendCorrect}' = \mathit{sendCorrect} \\
\mathit{earAttempts}' = \mathit{earAttempts} \\
\mathit{earCorrect}' = \mathit{earCorrect} \\
\hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{RecordSendAttempt}} \\
\hline
\Delta \mathit{CharacterStat} \\
\mathit{correct?} : \mathit{ZBOOL} \\
\hline
\mathit{sendAttempts}' = \mathit{sendAttempts} + 1 \\
(\mathit{correct?} = \mathit{ztrue} \land \mathit{sendCorrect}' = \mathit{sendCorrect} + 1) \lor (\mathit{correct?} = \mathit{zfalse} \land \mathit{sendCorrect}' = \mathit{sendCorrect}) \\
\mathit{receiveAttempts}' = \mathit{receiveAttempts} \\
\mathit{receiveCorrect}' = \mathit{receiveCorrect} \\
\mathit{earAttempts}' = \mathit{earAttempts} \\
\mathit{earCorrect}' = \mathit{earCorrect} \\
\hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{RecordEarAttempt}} \\
\hline
\Delta \mathit{CharacterStat} \\
\mathit{correct?} : \mathit{ZBOOL} \\
\hline
\mathit{earAttempts}' = \mathit{earAttempts} + 1 \\
(\mathit{correct?} = \mathit{ztrue} \land \mathit{earCorrect}' = \mathit{earCorrect} + 1) \lor (\mathit{correct?} = \mathit{zfalse} \land \mathit{earCorrect}' = \mathit{earCorrect}) \\
\mathit{receiveAttempts}' = \mathit{receiveAttempts} \\
\mathit{receiveCorrect}' = \mathit{receiveCorrect} \\
\mathit{sendAttempts}' = \mathit{sendAttempts} \\
\mathit{sendCorrect}' = \mathit{sendCorrect} \\
\hline
\end{array}$$

10 Session Flow

This section models the state machine for training sessions. While persistent state (levels, streaks, intervals) survives across sessions, session state exists only during active training.

10.1 Session Phase

A training session progresses through distinct phases:

$$\mathit{SessionPhase} ::= \mathit{introduction} \mid \mathit{training} \mid \mathit{paused} \mid \mathit{completed}$$

- *introduction*: Showing unlocked characters with audio playback
- *training*: Active call-and-response practice
- *paused*: Session suspended mid-training
- *completed*: Session ended, results recorded

10.2 Session Constants

$$
\begin{array}{|l}
\mathit{proficiencyThreshold} : \\
\mathit{baseMinimumAttempts} : \\
\mathit{attemptsPerCharacter} : \\
\mathit{responseTimeoutMs} : \\
\hline
\mathit{proficiencyThreshold} = 90 \\
\mathit{baseMinimumAttempts} = 15 \\
\mathit{attemptsPerCharacter} = 5 \\
\mathit{responseTimeoutMs} = 3000
\end{array}
$$

The proficiency threshold (90%) matches the level advancement threshold. Minimum attempts scale with character count: $\max(15, 5 \times \mathit{characterCount})$.

10.3 Session State

Session state is separate from persistent progress state. In the implementation, this lives in the View-Model; here we model it as a distinct schema.

$$
\begin{array}{|l}
\hline
\mathit{SessionState} \\
\mathit{phase} : \mathit{SessionPhase} \\
\mathit{direction} : \mathit{Direction} \\
\mathit{radioMode} : \mathit{RadioMode} \\
\mathit{toneActive} : \mathit{ZBOOL} \\
\mathit{characterCount} : \\
\mathit{introIndex} : \\
\mathit{sessionAttempts} : \\
\mathit{sessionCorrect} : \\
\hline
\mathit{characterCount} \geq 1 \\
\mathit{characterCount} \leq \mathit{maxLevel} \\
\mathit{introIndex} \leq \mathit{characterCount} \\
\mathit{sessionCorrect} \leq \mathit{sessionAttempts} \\
\mathit{sessionAttempts} \leq 1000 \\
\mathit{phase} \in \{\mathit{introduction}, \mathit{paused}, \mathit{completed}\} \Rightarrow \mathit{radioMode} = \mathit{radioOff} \\
(\mathit{phase} = \mathit{training} \land \mathit{direction} = \mathit{receive}) \Rightarrow \mathit{radioMode} = \mathit{radioReceiving} \\
\mathit{radioMode} = \mathit{radioTransmitting} \Rightarrow \mathit{direction} = \mathit{send} \\
\mathit{radioMode} = \mathit{radioOff} \Rightarrow \mathit{toneActive} = \mathit{zfalse} \\
\hline
\end{array}
$$

The *characterCount* is the number of unlocked characters for this session's direction (equal to the current level). The *introIndex* tracks progress through the introduction phase.

The *radioMode* models half-duplex radio behavior:

- During introduction, pause, or completion: radio is off (no audio)
- During receive training: radio always receiving (user listens to incoming Morse)
- During send training: user toggles between receiving (waiting) and transmitting (keying)
- Transmitting is only valid in send mode (cannot transmit during receive training)

10.4 Session Initialization

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{InitSession}} \\
\hline
\mathit{SessionState}' \\
\mathit{direction}? : \mathit{Direction} \\
\mathit{level}? : \\
\hline
\mathit{level}? \geq \mathit{minLevel} \\
\mathit{level}? \leq \mathit{maxLevel} \\
\mathit{phase}' = \mathit{introduction} \\
\mathit{direction}' = \mathit{direction}? \\
\mathit{radioMode}' = \mathit{radioOff} \\
\mathit{toneActive}' = \mathit{zfalse} \\
\mathit{characterCount}' = \mathit{level}? \\
\mathit{introIndex}' = 0 \\
\mathit{sessionAttempts}' = 0 \\
\mathit{sessionCorrect}' = 0 \\
\hline
\end{array}$$

A session begins in the introduction phase with the radio off. The radio activates when training begins (transitioning to receiving mode).

10.5 Introduction Phase Operations

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{NextIntroCharacter}} \\
\hline
\Delta \mathit{SessionState} \\
\hline
\mathit{phase} = \mathit{introduction} \\
\mathit{introIndex} < \mathit{characterCount} \\
\mathit{introIndex}' = \mathit{introIndex} + 1 \\
\mathit{phase}' = \mathit{phase} \\
\mathit{direction}' = \mathit{direction} \\
\mathit{radioMode}' = \mathit{radioMode} \\
\mathit{toneActive}' = \mathit{toneActive} \\
\mathit{characterCount}' = \mathit{characterCount} \\
\mathit{sessionAttempts}' = \mathit{sessionAttempts} \\
\mathit{sessionCorrect}' = \mathit{sessionCorrect} \\
\hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{CompleteIntroduction}} \\
\hline
\Delta \mathit{SessionState} \\
\hline
\mathit{phase} = \mathit{introduction} \\
\mathit{introIndex} = \mathit{characterCount} \\
\mathit{phase}' = \mathit{training} \\
\mathit{radioMode}' = \mathit{radioReceiving} \\
\mathit{toneActive}' = \mathit{zfalse} \\
\mathit{introIndex}' = \mathit{introIndex} \\
\mathit{direction}' = \mathit{direction} \\
\mathit{characterCount}' = \mathit{characterCount} \\
\mathit{sessionAttempts}' = \mathit{sessionAttempts} \\
\mathit{sessionCorrect}' = \mathit{sessionCorrect} \\
\hline
\end{array}$$

When all characters have been shown ($\mathit{introIndex} = \mathit{characterCount}$), the session transitions to the training phase and the radio turns on in receiving mode.

10.6 Training Phase Operations

During training, the student responds to character prompts. Each response (correct, incorrect, or time-out) is recorded.

$$\begin{array}{|l}
\textit{RecordCorrectResponse} \\
\Delta \textit{SessionState} \\
\hline
\textit{phase} = \textit{training} \\
\textit{sessionAttempts}' = \textit{sessionAttempts} + 1 \\
\textit{sessionCorrect}' = \textit{sessionCorrect} + 1 \\
\textit{phase}' = \textit{phase} \\
\textit{direction}' = \textit{direction} \\
\textit{radioMode}' = \textit{radioMode} \\
\textit{toneActive}' = \textit{toneActive} \\
\textit{characterCount}' = \textit{characterCount} \\
\textit{introIndex}' = \textit{introIndex}
\end{array}$$

$$\begin{array}{|l}
\textit{RecordIncorrectResponse} \\
\Delta \textit{SessionState} \\
\hline
\textit{phase} = \textit{training} \\
\textit{sessionAttempts}' = \textit{sessionAttempts} + 1 \\
\textit{sessionCorrect}' = \textit{sessionCorrect} \\
\textit{phase}' = \textit{phase} \\
\textit{direction}' = \textit{direction} \\
\textit{radioMode}' = \textit{radioMode} \\
\textit{toneActive}' = \textit{toneActive} \\
\textit{characterCount}' = \textit{characterCount} \\
\textit{introIndex}' = \textit{introIndex}
\end{array}$$

A timeout is treated as an incorrect response—the student failed to respond within *responseTimeoutMs* (3 seconds).

10.7 Pause and Resume

$$\begin{array}{|l}
\textit{PauseSession} \\
\Delta \textit{SessionState} \\
\hline
\textit{phase} = \textit{training} \\
\textit{phase}' = \textit{paused} \\
\textit{radioMode}' = \textit{radioOff} \\
\textit{toneActive}' = \textit{zfalse} \\
\textit{direction}' = \textit{direction} \\
\textit{characterCount}' = \textit{characterCount} \\
\textit{introIndex}' = \textit{introIndex} \\
\textit{sessionAttempts}' = \textit{sessionAttempts} \\
\textit{sessionCorrect}' = \textit{sessionCorrect}
\end{array}$$

$$\begin{array}{|l}
\textit{ResumeSession} \\
\Delta \textit{SessionState} \\
\hline
\textit{phase} = \textit{paused} \\
\textit{phase}' = \textit{training} \\
\textit{radioMode}' = \textit{radioReceiving} \\
\textit{toneActive}' = \textit{zfalse} \\
\textit{direction}' = \textit{direction} \\
\textit{characterCount}' = \textit{characterCount} \\
\textit{introIndex}' = \textit{introIndex} \\
\textit{sessionAttempts}' = \textit{sessionAttempts} \\
\textit{sessionCorrect}' = \textit{sessionCorrect}
\end{array}$$

Pausing turns the radio off. Resuming returns to receiving mode (the radio turns back on, listening for incoming signals).

10.8 Radio Mode Transitions

During send training, the student toggles between receiving (waiting to key) and transmitting (actively keying). These transitions model the half-duplex nature of amateur radio.

$$
\begin{array}{|l}
\textit{EnterTransmitMode} \\
\hline
\Delta \textit{SessionState} \\
\hline
\textit{phase} = \textit{training} \\
\textit{direction} = \textit{send} \\
\textit{radioMode} = \textit{radioReceiving} \\
\textit{radioMode}' = \textit{radioTransmitting} \\
\textit{toneActive}' = \textit{toneActive} \\
\textit{phase}' = \textit{phase} \\
\textit{direction}' = \textit{direction} \\
\textit{characterCount}' = \textit{characterCount} \\
\textit{introIndex}' = \textit{introIndex} \\
\textit{sessionAttempts}' = \textit{sessionAttempts} \\
\textit{sessionCorrect}' = \textit{sessionCorrect} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\textit{ExitTransmitMode} \\
\hline
\Delta \textit{SessionState} \\
\hline
\textit{phase} = \textit{training} \\
\textit{direction} = \textit{send} \\
\textit{radioMode} = \textit{radioTransmitting} \\
\textit{radioMode}' = \textit{radioReceiving} \\
\textit{toneActive}' = \textit{zfalse} \\
\textit{phase}' = \textit{phase} \\
\textit{direction}' = \textit{direction} \\
\textit{characterCount}' = \textit{characterCount} \\
\textit{introIndex}' = \textit{introIndex} \\
\textit{sessionAttempts}' = \textit{sessionAttempts} \\
\textit{sessionCorrect}' = \textit{sessionCorrect} \\
\hline
\end{array}
$$

EnterTransmitMode occurs when the student begins keying (presses dit or dah). *ExitTransmitMode* occurs when keying completes (timeout after last element). Note that transmitting is only valid during send training—the precondition $\textit{direction} = \textit{send}$ enforces this constraint.

10.9 Tone Activation

During training, Morse tones are generated by activating and deactivating the tone generator. The *toneActive* flag controls whether audio is currently being produced. The audio output depends on the current radio mode:

- *radioReceiving*: Tone mixed with noise floor and band conditions (QRN, QSB, QRM)
- *radioTransmitting*: Clean sidetone only (no noise or band conditions)
- *radioOff*: No audio output (toneActive must be false)

$$\begin{array}{l}
\textit{ActivateTone} \\
\hline
\Delta \mathit{SessionState} \\
\hline
\mathit{radioMode} \neq \mathit{radioOff} \\
\mathit{toneActive} = \mathit{zfalse} \\
\mathit{toneActive}' = \mathit{ztrue} \\
\mathit{phase}' = \mathit{phase} \\
\mathit{direction}' = \mathit{direction} \\
\mathit{radioMode}' = \mathit{radioMode} \\
\mathit{characterCount}' = \mathit{characterCount} \\
\mathit{introIndex}' = \mathit{introIndex} \\
\mathit{sessionAttempts}' = \mathit{sessionAttempts} \\
\mathit{sessionCorrect}' = \mathit{sessionCorrect} \\
\hline
\end{array}$$

ActivateTone requires the radio to be on (either receiving or transmitting). The precondition $\mathit{toneActive} = \mathit{zfalse}$ prevents double activation.

$$\begin{array}{l}
\textit{DeactivateTone} \\
\hline
\Delta \mathit{SessionState} \\
\hline
\mathit{toneActive} = \mathit{ztrue} \\
\mathit{toneActive}' = \mathit{zfalse} \\
\mathit{phase}' = \mathit{phase} \\
\mathit{direction}' = \mathit{direction} \\
\mathit{radioMode}' = \mathit{radioMode} \\
\mathit{characterCount}' = \mathit{characterCount} \\
\mathit{introIndex}' = \mathit{introIndex} \\
\mathit{sessionAttempts}' = \mathit{sessionAttempts} \\
\mathit{sessionCorrect}' = \mathit{sessionCorrect} \\
\hline
\end{array}$$

DeactivateTone stops tone generation. The precondition $\mathit{toneActive} = \mathit{ztrue}$ ensures we only deactivate an active tone.

In the implementation, `playToneElement(duration:)` performs:

1. *ActivateTone* (start generating tone at configured frequency)
2. Wait for the specified duration (dit: 60ms, dah: 180ms)
3. *DeactivateTone* (stop tone generation)

The continuous audio engine ensures smooth transitions—the audio buffer runs continuously, and only the tone generation flag changes, avoiding clicks or pops from starting/stopping the audio engine.

10.10 Proficiency Check

The session ends automatically when the student achieves proficiency: $\geq 90\%$ accuracy over at least $\max(15, 5 \times \mathit{characterCount})$ attempts.

$$\begin{array}{l}
\textit{CheckProficiencyMet} \\
\hline
\Xi \mathit{SessionState} \\
\mathit{proficiencyMet}! : \mathit{ZBOOL} \\
\hline
\mathit{phase} = \mathit{training} \\
\textbf{let } \mathit{minAttempts} == \textbf{if } \mathit{baseMinimumAttempts} > \mathit{attemptsPerCharacter} * \mathit{characterCount} \textbf{ then } \mathit{baseMinimumAt} \\
\hline
\end{array}$$

This is a query operation (Ξ indicates no state change). The implementation calls this after each response to determine whether to end the session.

10.11 Session Completion

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{CompleteSession}} \\
\Delta \mathit{SessionState} \\
\hline
\mathit{phase} = \mathit{training} \\
\mathit{phase}' = \mathit{completed} \\
\mathit{radioMode}' = \mathit{radioOff} \\
\mathit{toneActive}' = \mathit{zfalse} \\
\mathit{direction}' = \mathit{direction} \\
\mathit{characterCount}' = \mathit{characterCount} \\
\mathit{introIndex}' = \mathit{introIndex} \\
\mathit{sessionAttempts}' = \mathit{sessionAttempts} \\
\mathit{sessionCorrect}' = \mathit{sessionCorrect} \\
\hline
\end{array}$$

When a session completes (either by proficiency or user action), the radio turns off and the accumulated statistics are transferred to persistent state via *RecordSession* and the relevant *Advance* and *UpdateInterval* operations.

10.12 Session-to-Progress Integration

The session flow connects to persistent progress through these operations:

1. On session completion, calculate accuracy: $\mathit{sessionCorrect}\ \mathsf{div}\ \mathit{sessionAttempts}$
2. If accuracy $\geq 90\%$: invoke *AdvanceReceiveLevel* or *AdvanceSendLevel*
3. Based on accuracy tier: invoke appropriate *UpdateInterval* operation
4. Invoke *RecordSession* to increment session count
5. Update streak via *IncrementStreak* or *ResetStreak* based on calendar day

This integration is orchestrated by the application layer (ProgressStore in the implementation) rather than modeled as a single composite operation.

11 System Invariants

The following properties hold for any reachable state:

1. **Level bounds**: $1 \leq \mathit{receiveLevel} \leq 26$ and $1 \leq \mathit{sendLevel} \leq 26$
2. **Correctness bounds**: For any character, $\mathit{receiveCorrect} \leq \mathit{receiveAttempts}$ and $\mathit{sendCorrect} \leq \mathit{sendAttempts}$
3. **Streak monotonicity**: $\mathit{currentStreak} \leq \mathit{longestStreak}$
4. **Interval bounds**: $1 \leq \mathit{interval} \leq 30$ for both directions
5. **Non-negative counts**: All attempt and session counts are non-negative natural numbers

These invariants are encoded directly in the schema predicates, ensuring type-correct operations preserve them.

12 Properties Not Modeled

This specification focuses on state that affects learning outcomes. The following aspects are intentionally omitted:

- **Audio timing**: Dit/dah durations (60ms/180ms at 20 WPM), Farnsworth spacing adjustments
- **QSO simulation**: Virtual station exchanges, protocol phases, message templates

- **UI state**: Current screen, selected characters, input buffers
- **Notifications**: Practice due reminders, streak alerts, anti-nag policies

These could be modeled in future extensions or separate specifications.

13 Validation

This specification has been validated with:

- **fuzz**: Type-checking passes with no errors
- **probcli -init**: Initialization succeeds
- **probcli -animate**: All operations covered

The flattened state schema and bounded inputs ensure ProB can explore the state space without unbounded enumeration warnings.