1 Rate of Completions

As learners engage in activities supported by a learning ecosystem, they will build up a history of learning experiences. When the digital resources of that learning ecosystem adhere to a framework dedicated to supporting and understanding the learner, such as the Total Learning Architecture (TLA), it becomes possible to retell their learning story through data and data visualization. One important aspect of that story is the rate of completion of the various digital resources within the learning ecosystem.

$$
\begin{array}{|l}
\hline
\textit{RateOfCompletions}[KV, Collection, KV] \\
\textit{Algorithm} \\
\textit{RateOfCompletionsInit} \\
\textit{RateOfCompletionsRelevant?} \\
\textit{RateOfCompletionsAccept?} \\
\textit{RateOfCompletionsStep} \\
\textit{RateOfCompletionsResult} \\
rateOfCompletions\_ : KV \times Collection \times KV \twoheadrightarrow KV \\
state?, opt?, rateOfCompletions! : KV \\
S? : Collection \\
\hline
Algorithm.algorithm.algorithmIter = \langle relevant?\_, accept?\_, step\_\rangle \\
ROCalgorithmIter = \langle RateOfCompletionsRelevant?.relevant?\_, \\
\qquad\qquad RateOfCompletionsAccept?.accept?\_, \\
\qquad\qquad RateOfCompletionsStep.step\_\rangle \\
\\
Algorithm.algorithm.algorithmIter\_ \leadsto ROCalgorithmIter\_ \Rightarrow \\
\quad (Algorithm.algorithm.algorithmIter.relevant?\_ \leadsto \\
\qquad\quad RateOfCompletionsRelevant?.relevant?\_) \land \\
\quad (Algorithm.algorithm.algorithmIter.accept?\_ \leadsto \\
\qquad\quad RateOfCompletionsAccept?.accept?\_) \land \\
\quad (Algorithm.algorithm.algorithmIter.step\_ \leadsto \\
\qquad\quad RateOfCompletionsStep.step\_) \\
\\
rateOfCompletions! = rateOfCompletions(state?, S?, opt?) = algorithm(state?, S?, opt?) \bullet \\
\quad (Algorithm.algorithm.init\_ \leadsto RateOfCompletionsInit.init\_) \land \\
\quad (Algorithm.algorithm.algorithmIter\_ \leadsto ROCalgorithmIter\_) \land \\
\quad (Algorithm.algorithm.result\_ \leadsto RateOfCompletionsResult.result\_) \\
\hline
\end{array}
$$

- FIXME: explain usage of $\leadsto$
- FIXME: explan usage of $SchemaName.component.subcomponent$

1.1 Initialization

$RateOfCompletionsInit[KV]$

$Init$
$state?, state! : KV$
$init\_ : KV \twoheadrightarrow KV$

$Init.init = \langle body \rangle$
$\langle body \rangle \leadsto init = \langle FIXME : what\,primitives\,are\,needed? \rangle$
$state! = init(state?) \bullet FIXME : what\,happens?$

1.2 Relevant?

$RateOfCompletionsRelevant?[KV, STATEMENT]$

$Relevant?$
$state? : KV$
$stmt? : STATEMENT$
$relevant! : Boolean$
$relevant?\_ : KV \times STATEMENT \to Boolean$

$Relevant.relevant? = \langle body \rangle$
$\langle body \rangle \leadsto relevant? = \langle FIXME : what\,primitives\,are\,needed? \rangle$
$relevant! = relevant?(state?, stmt?) \bullet FIXME : what\,happens?$

$relevant?(state, stmt)$ determines if $stmt$ is valid for use within $step$ of $rateOfCompletions$ and does so by looking into various $k \to v$ within $stmt$. The following Primitives are used as the $body$ of $relevant?(state, stmt)$

- is the Object of the Statement an Activity?

$$activityType = atKey(stmt, < object, objectType >)$$

$$activity?(activityType) = true \iff activityType = Activity \vee activityType = nil$$

- is the Verb indicative of a completion event?

$$verbId = atKey(stmt, < verb, id >)$$

$$completionVerb?(verbId) = true$$

$$\iff$$

$$verbId = http://adlnet.gov/expapi/verbs/passed$$

$$\vee$$

$$verbId = https://w3id.org/xapi/dod-isd/verbs/answered$$

$$\vee$$

$$verbId = http://adlnet.gov/expapi/verbs/completed$$

- does the *stmt* indicate completion using Result?

$$result = atKey(stmt, < result, completion >)$$

$$resultCompletion = true \iff result = true$$

such that the body of *relevant*? contains

$$p_a(stmt) = activity?\,(atKey(stmt, < object, objectType >))$$

and

$$p_v(stmt) = completionVerb?\,(atKey(stmt, < verb, id >))$$

and

$$p_r(stmt) = resultCompletion(atKey(stmt, < result, completion >))$$

which are used to form higher level Primitives

$$p_{continue}(stmt) = stmt \iff p_a(stmt) = true$$

and

$$p_{completed?}(stmt) = stmt \iff p_v(stmt) = true \lor p_r(stmt) = true$$

which results in a final Primitive $p_{return?}$

$$p_{return?}(stmt) = object?\,(p_{completed?}(p_{continue}(stmt)))$$

which defines the *body* of *relevant*?

$$relevant?\,(stmt) = p_{return?}(stmt) \Rightarrow object?\,(p_{completed?}(p_{continue}(stmt)))$$

and can be summarized as

$$relevant?\,(state, stmt) = true$$

$$\iff$$

$$activity?\,(activitType) = true$$

$$\land$$

$$completionVerb?\,(verbId) = true \lor resultCompletion = true$$

1.3 Accept?

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{RateOfCompletionsAccept?}\,[KV, STATEMENT]} \\
Accept? \\
state? : KV \\
stmt? : STATEMENT \\
accept! : Boolean \\
accept?\ \_ : KV \times STATEMENT \rightarrow Boolean \\
\hline
Accept?\,.accept? = \langle body \rangle \\
\langle body \rangle \leadsto accept? = \langle FIXME:\ what\,primitives\,are\,needed?\,\rangle \\
accept! = accept?\,(state?\,, stmt?\,) \bullet FIXME:\ what\,happens? \\
\hline
\end{array}$$

$rateOfCompletions$ does not require further boolean logic to determine if $stmt$ and $state$ can be passed to $step$

$$accept?\,(state, stmt) = object?\,(stmt)$$

which should always return true assuming valid xAPI Statements are passed to $rateOfCompletions$

1.4 Step

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{RateOfCompletionsStep}[KV, STATEMENT]} \\
Step \\
state?\,, step! : KV \\
stmt? : STATEMENT \\
step\,\_ : KV \times STATEMENT \twoheadrightarrow KV \\
\hline
Step.step = \langle body \rangle \\
\langle body \rangle \leadsto step = \langle FIXME:\ what\,primitives\,are\,needed?\,\rangle \\
step! = step(state?\,, stmt?\,) \bullet FIXME:\ what\,happens? \\
\hline
\end{array}$$

1.4.1 summary

$step(state, stmt)$ updates $state$ to include

$$\$.object.id \mapsto < domain, statementCount, name >$$

where

$$domain \mapsto < start, end >$$

$$statementCount \mapsto \mathbb{R}$$

$$name \mapsto < \$.object.definition.name >$$

at

$$< state, completions, \$.object.id >$$

1.4.2 processing

step starts by extracting the relevant information from *stmt*

- *currentTime*

$$currentTime = atKey(stmt, timestamp)$$

- *name*

$$name_{stmt} = atKey(stmt, < object, definition, name >)$$

- *objectId*

$$objectId = atKey(stmt, < object, id >)$$

which allows for the previous *state* to be resolved using *objectId*

- *domain*

$$domain_{state} = atKey(state, < state, completions, objectId, domain >)$$

$$start_{state} = first(domain_{state})$$
$$end_{state} = last(domain_{state})$$

- *statementCount*

$$statementCount_{state} = atKey(state, < state, completions, objectId, statementCount >)$$

- *name*

$$name_{state} = atKey(state, < state, completions, objectId, name >)$$

so that the previous state can be used along side the information parsed from *stmt*

- does $start_{state}$ need to be updated to *currentTime*?

  where
  $$inSeconds_{stmt} = isoToUnixEpoch(currentTime)$$
  $$inSeconds_{start} = isoToUnixEpoch(start_{state}) \iff start_{state} \neq nil$$
  such that
  $$start(state, stmt) = currentTime$$
  $$\iff$$
  $$start_{state} = nil$$
  $$\vee$$
  $$inSeconds_{stmt} \leq inSeconds_{start}$$
  otherwise
  $$start(state, stmt) = start_{state}$$

- does end_{state} need to be updated to $currentTime$?

  where
  $$inSeconds_{stmt} = isoToUnixEpoch(currentTime)$$
  $$inSeconds_{end} = isoToUnixEpoch(end_{state}) \iff end_{state} \neq nil$$
  such that
  $$end(state, stmt) = currentTime$$
  $$\iff$$
  $$end_{state} = nil$$
  $$\vee$$
  $$inSeconds_{stmt} \geq inSeconds_{end}$$
  otherwise
  $$end(state, stmt) = end_{state}$$

- what should $statementCount$ be?

  $$nStmts(state) = 1 \iff statementCount_{state} = 0 \ \vee \ nil$$
  $$\vee$$
  $$nStmts(state) = 1 + statementCount_{state} \iff statementCount_{state} \geq 1$$

- do we need to add a new $name$?

  $$allNames(state, stmt) = append(name_{state}, name_{stmt}, count(name_{state}))$$
  $$\iff$$
  $$name_{stmt} \notin name_{state}$$
  otherwise
  $$allNames(state, stmt) = name_{state}$$

which allows for the following primitives to be defined

$$p_{start}(state, stmt) = start(state, stmt)$$
$$p_{end}(state, stmt) = end(state, stmt)$$
$$p_{stmtCount}(state, stmt) = nStmts(state)$$
$$p_{names}(state, stmt) = allNames(state, stmt)$$

and establish relevant paths into $state$

$$K_{domain} =< state, completions, objectId, domain >$$
$$K_{stmtCount} =< state, completions, objectId, statementCount >$$

$$K_{names} =< state, completions, objectId, name >$$

which are used within higher level primitives concerned with updating *state*

$$p_{updateStart}(state, stmt)$$

$$\equiv$$

$$associate(state, K_{domain}, append(remove(domain_{state}, 0), p_{start}(state, stmt), 0))$$

and

$$p_{updateEnd}(state, stmt)$$

$$\equiv$$

$$associate(state, K_{domain}, append(remove(domain_{state}, 1), p_{end}(state, stmt), 1))$$

and

$$p_{updatedCount}(state, stmt)$$

$$\equiv$$

$$associate(state, K_{stmtCount}, p_{stmtCount}(state, stmt))$$

and

$$p_{updatedNames}(state, stmt)$$

$$\equiv$$

$$associate(state, K_{names}, p_{names}(state, stmt))$$

such that *body* of *step* is defined as

$$step(state, stmt) = p_{updateNames}(p_{updateCount}(p_{updateEnd}(p_{updateStart}(state, stmt), stmt), stmt), stmt)$$

where

$$state' = p_{updateStart}(state, stmt)$$

and

$$state'' = p_{updateEnd}(state', stmt)$$

and

$$state''' = p_{updateCount}(state'', stmt)$$

such that

$$step(state, stmt) = p_{updateNames}(state''', stmt)$$

1.5 Result

$$
\begin{array}{|l}
\textit{RateOfCompletionsResult}[KV, KV] \\
\textit{Result} \\
opt?, state?, result! : KV \\
result\_ : KV \times KV \twoheadrightarrow KV \\
\hline
Result.result = \langle body \rangle \\
\langle body \rangle \rightsquigarrow result = \langle FIXME: \ what\, primitives\, are\, needed? \rangle \\
result! = result(state?, opt?) \bullet FIXME: \ what\, happens? \\
\hline
\end{array}
$$

The only *opts* used by *rateOfCompletions* is *timeUnit*

$$timeUnit = second \ \vee minute \ \vee \ hour \ \vee day \ \vee month \ \vee year$$

and will default to *day* if not passed to *rateOfCompletions*

$$result(state) = result(state, < timeUnit \mapsto day >)$$

which is passed to *rateOf* along with the arguments parsed from *state*

$$unit = atKey(opts, timeUnit)$$

$$allCompletions(state) = atKey(state, < state, completions >)$$

such that

$$\forall k_n : i..n..j \in allCompletions(state)$$

the following primitives are called each iteration

$$getCount(state, k_n) = atKey(allCompletions(state), < k_n, statementCount >)$$

$$getStart(state, k_n) = atKey(allCompletions(state), < k_n, domain, start >)$$

$$getEnd(state, k_n) = atKey(allCompletions(state), < k_n, domain, end >)$$

$$getName(state, k_n) = atKey(allCompletions(state), < k_n, name >)$$

which allows for

$$rate_n(state, k_n, unit) = rateOf(getCount(state, k_n), getStart(state, k_n), getEnd(state, k_n), unit)$$

such that

$$value_n(state, k_n, unit) =< x_n, y_n >$$

where

$$name_n(state, k_n) = first(getName(state, k_n))$$

$$x_n = x \mapsto name_n(state, k_n) \iff name_n(state, k_n) \neq nil$$

otherwise

$$x_n = x \mapsto k_n$$

and

$$y_n = y \mapsto rate_n(state, k_n, unit)$$

such that

$$value_n(state, k_n, unit) =< name_n(state, k_n),\ rate_n(state, k_n, unit) >$$

and

$$value(state, unit) = \forall k_n : i..n..j \in allCompletions(state) \exists!\, value_n(state, k_n, unit) =< x_n, y_n >$$

$$\Rightarrow$$

$$value(state, unit) =< value_i(state, k_i, unit)..value_n(state, k_n, unit)..value_j(state, k_j, unit) >$$

which allows the body of *result* to be defined using

$$unit = atKey(opts, timeUnit)$$

$$K_{store} =< state, completions, values, unit >$$

so that *result* returns an updated *state* with the rate of completions per *unit* located at K_{store}

$$result(state, opts) = associate(state, K_{store}, value(state, unit))$$