Logic App Development: Coursework Specification

Felix Neubauer

Type of Logic/Game

The game uses ALC description logic (more expressive than propositional logic but less expressive than first order logic). Concept descriptions are interpreted as sets of objects.

Equivalence Lemma: Let I be an interpretation, C and D concepts and r a role. Then

1. $\top^I = (C \sqcup \neg C)^I$ (Tautology)
2. $\bot^I = (C \sqcap \neg C)^I$ (Contradiction)
3. $(\neg\neg C)^I = C^I$ (Double Negation)
4. $(\neg(C \sqcup D))^I = (\neg C \sqcap \neg D)^I$ (De Morgan's Laws)
5. $(\neg(C \sqcap D))^I = (\neg C \sqcup \neg D)^I$
6. $(\neg(\exists r.C))^I = (\forall r.\neg C)^I$ (Quantifier Duality)
7. $(\neg(\forall r.C))^I = (\exists r.\neg C)^I$

(Source: Knowledge Graphs lecture by Steffen Staab)

An ALC knowledge base K consists of an TBox T and an ABox A. T contains conceptual knowledge, while A contains knowledge about objects on instance level.

Example: ABox

$alice$: $Actor$
bob: $Actor$
$bulbFiction$: $Movie$
$eight$: $Movie$
$played\_in(alice, bulbFiction)$
$played\_in(alice, eight)$
$played\_in(bob, eight)$

TBox

$Actor \sqsubseteq \exists played\_in.Movie$
$AwardWinner \sqsubseteq \exists won.Award$

(Source: Knowledge Graphs lecture by Steffen Staab)

Knowledge bases either are *consistent* or they are not.

Example: Consistent KB $K_1 = (A_1, T_1)$

$$T_1 = \{\, Human \sqsubseteq Male \sqcup Female, \bot \sqsubseteq Male \sqcap Female\},$$
$$A_1 = \{\, bob{:}\, \neg Male \sqcap \neg Female \,\}$$

Inconsistent KB $K_2 = (A_2, T_1)$

$$A_2 = \{\, bob{:}\, \neg Male \sqcap \neg Female \sqcap Human \,\}$$

To not make the app too complex, we stick to just the ABox (equivalent to a KB with empty TBox, or KB with TBox already unfolded into ABox).
The player will be given an ABox and he has to decide, whether the ABox is consistent or not.
Example: Consistent ABox

$$A_1 = \{\, a{:}\, \neg B \sqcup (B \sqcap C) \,\}$$

Inconsistent ABoxes

$$A_2 = \{\, a{:}\, \neg B \sqcap (B \sqcap C) \,\} \qquad A_3 = \{\, a{:}\, \exists r_1.C,\;\; a{:}\, \forall r1.(\neg C) \,\}$$

To simplify the user interface even more, we will give the player not one ABox that consists of multiple entries, but instead we give the player exactly one Concept and the player must decide, whether the Concept is satisfiable or not (equivalent to KB consistency with $K = (\{a_1{:}\, C\}, \emptyset)$ and C being the Concept checked for satisfiability).

App Features

- Front-End
    - Screens
        - Main-Screen: here the player sees the quiz and can play the quiz. A menubar leads the player to the other screens. Upon completing one question, the correct answer will be revealed on the same screen. The user can press a button to continue to the next question
        - Statistic screen (see mock screens for more information)
        - User Guide Screen
    - Rest
        - Offline storage of tasks
    - Optional requirements I choose to work on
        - Learning technique
        - Light and dark mode
        - Timer for statistical information
- Server-side
    - Provides tasks to front-end. Has 200+ tasks available.
    - Communication via GraphQl
    - Collects statistics from front-end
    - HTML interface for admin (view statistics, add or remove tasks)
    - Uses local database for data storage
- User management
    - Using third-party libraries
- Task generation
    - See following section on task generation

Task Generation

General idea to generate one task (which can be modified to generate many tasks):

User input: $n_{basicConcepts}, n_{roles}, n_{levels}, requestClash, taskComplexity$

1. Concepts $C = \left\{C_1, \dots, C_{n_{basicConcepts}}\right\} \cup \{\top \cup \bot\}$ and roles $R = \{r_1, \dots, r_{n_{Roles}}\}$
2. $lvls = \left[C, lvl_1, \dots, lvl_{n_{levels}}\right]$,with $lvl_x = combineConcepts(lvl_{x-1})$

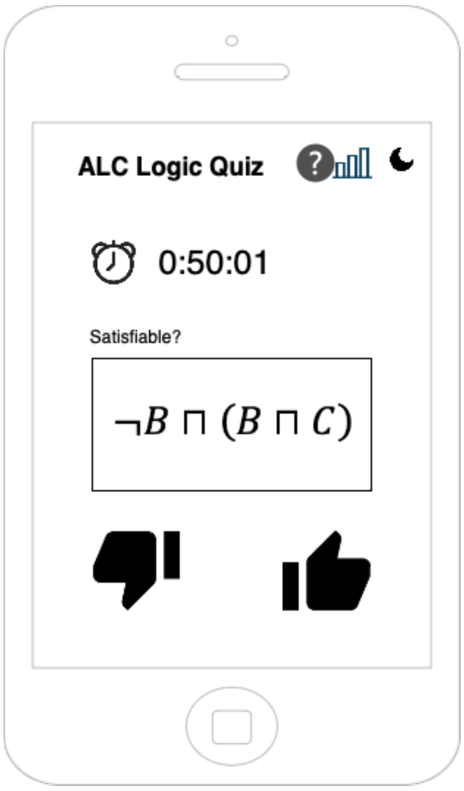
$combineConcepts(lvl_n)$:

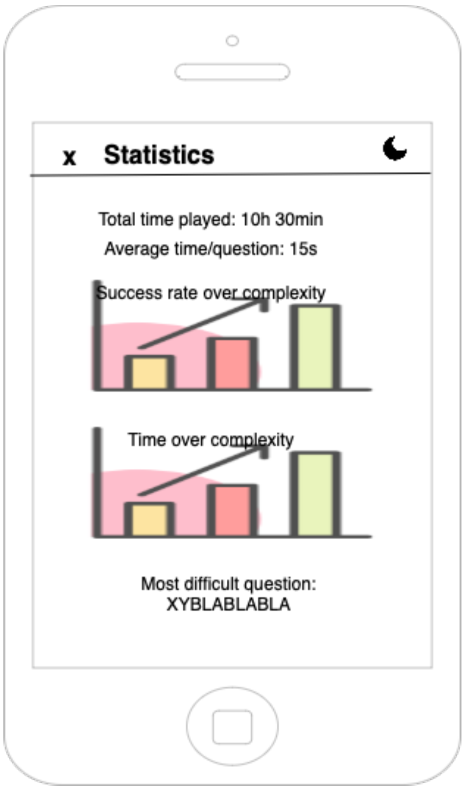
- For every $C \in lvl_n$, add $\neg C$
- For every combination $A, B \in lvl_n, A \neq B$, add $(A \sqcup B)$ and $(A \sqcap B)$
- For every combination $A \in lvl_n, r \in R$, add $(\exists r. A)$ and $(\forall r. A)$

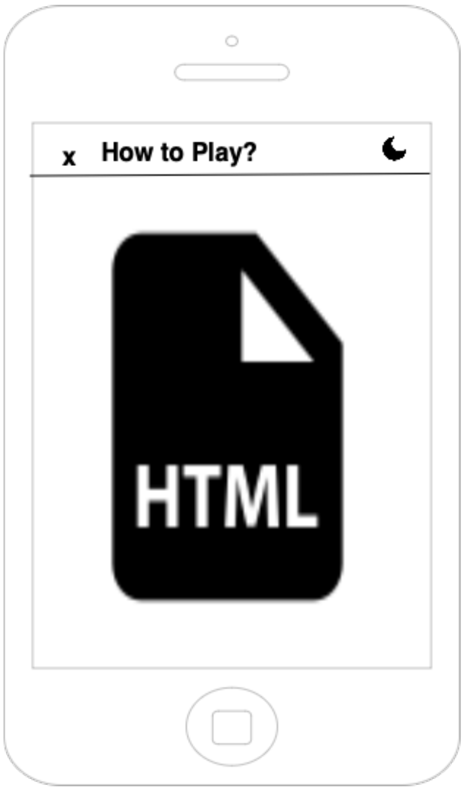
3. Flatten the list $lvls$ to one big set C^* and for every entry, compute 1. $complexity$ (operator count) and 2. $satisfiability$
4. For every $C \in C^*$, normalize the variable names and then remove duplicates
5. Randomly pick a $C \in C^*$ that meets the following conditions
    a. $complexity = taskComplexity$
    b. $satisfiability = \neg requestClash$

To answer satisfiability, an ALC ABox consistency algorithm needs to be implemented. This could be done using tableau algorithms. 1. Convert to Negation Normal Form (NNF) 2. construct a complete and clash-free ABox using the $\sqcap, \sqcup, \forall, \exists$-Rules to expand.

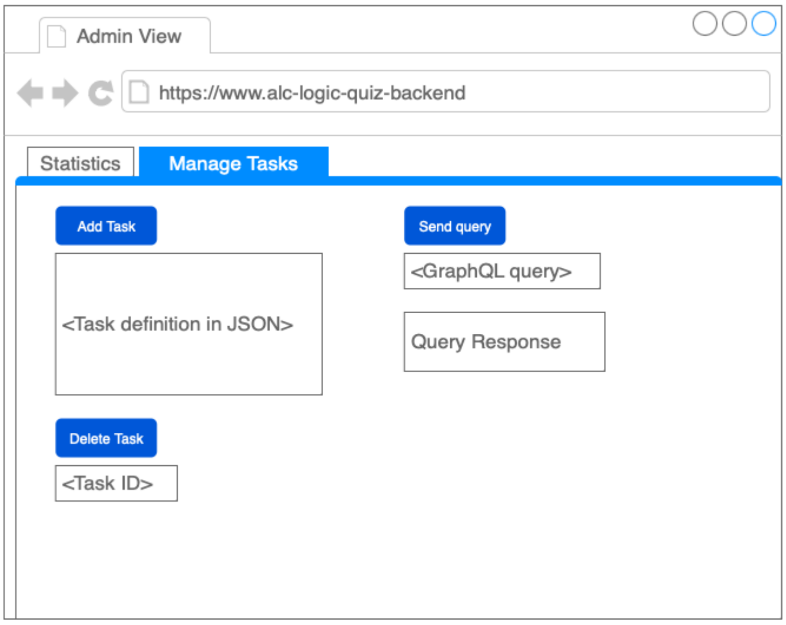
Frontend User Interface

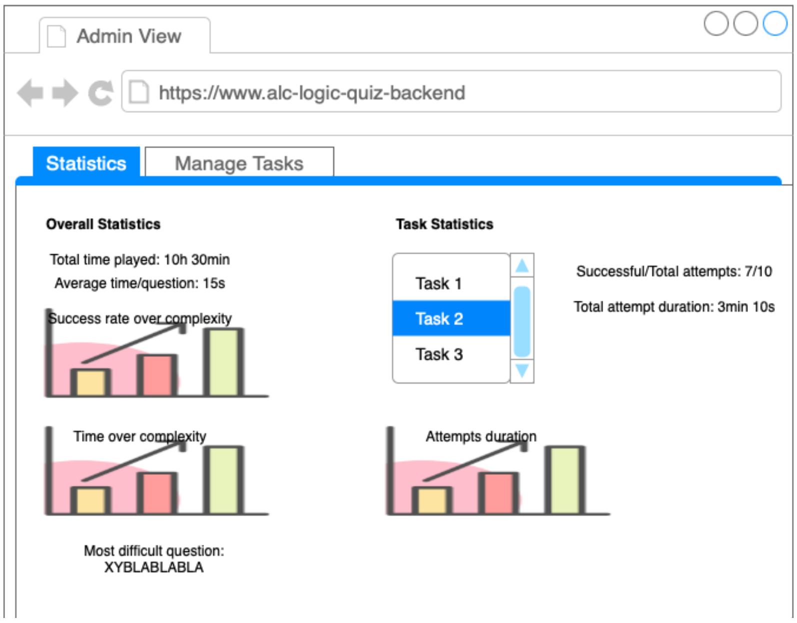



x Statistics
Total time played: 10h 30min
Average time/question: 15s
Success rate over complexity
Time over complexity
Most difficult question:
XYBLABLABLA



Backend Admin Interface





Overall Statistics

Total time played: 10h 30min

Average time/question: 15s

Most difficult question: XYBLABLABLA

Task Statistics

Successful/Total attempts: 7/10

Total attempt duration: 3min 10s