CmpE 260 - Principles of Programming Languages
Spring 2019
Project 1

Due Date: 21.04.2019 - 23:59

1 Pokémon: Let's Go, Prolog!

Pokémon World needs your help to set things to rights. You will design an interface for Pokémon Trainers, so that they can create different Pokémon teams and simulate battles against other Trainers. Pokémon is identical in the singular and plural. (Motivation)

2 Knowledge Base

In your knowledge base (**pokemon_data.pl**), you have a list of predicates, *pokemon_stats*, *pokemon_evolution*, *pokemon_types*, *type_chart_attack* and *pokemon_trainer*. They are defned as follows:

```
pokemon_stats(Pokemon, Types, HealthPoint, Attack, Defense).
pokemon_evolution(Pokemon, EvolvedPokemon, MinRequiredLevel).
pokemon_types(PokemonTypes).
type_chart_attack(AttackingType, TypeMultipliers).
pokemon_trainer(PokemonTrainer, PokemonTeam, PokemonLevels).
```

- `pokemon_stats` defines a Pokémon, and for each Pokémon, its types, base health points, attack and defense values are listed. Health points, attack and defense values are Pokémon's base statistics or in other words, when the Pokémon is at level 0.

  ```
  pokemon_stats(bulbasaur, [grass, poison], 45, 49, 49).
  ```

- `pokemon_evolution` defines the evolution process for Pokémon that can evolve.

  ```
  pokemon_evolution(bulbasaur, ivysaur, 16).
  ```

- `pokemon_types` defines all Pokémon types. This is just a single fact to show the order of types to be used in `type_chart_attack`.

  ```
  pokemon_types([normal, fire, water, electric, grass, ice, fighting, poison, ground,
  flying, psychic, bug, rock, ghost, dragon, dark, steel, fairy]).
  ```

- `type_chart_attack` defines a type, and for each type, attack multipliers against all types are listed. The ordering of the multiplier lists are the same with `pokemon_types`.

  ```
  type_chart_attack(grass, [1.0, 0.5, 2.0, 1.0, 0.5, 1.0, 1.0, 0.5, 2.0, 0.5, 1.0,
  0.5, 2.0, 1.0, 0.5, 1.0, 0.5, 1.0]).
  ```

- `pokemon_trainer` defines a Pokémon trainer, and for each Pokémon trainer, his/her Pokémon team and their levels are listed.

  ```
  pokemon_trainer(ash, [pikachu, bulbasaur, charmeleon, squirtle], [45, 15, 28, 42]).
  ```

3 Game Mechanics

In this section we will explain important mechanics that you should consider while coding writing required predicates. It will also help you to understand the concept better. Do not make any assumptions or add new concepts that are not present here.

3.1 Leveling Up

Each Pokémon can be at any level. Level of a Pokémon has an effect on its statistics and evolution process. With every level a Pokémon gains 2 health points, 1 attack point and 1 defense point. For example; base values of health point, attack and defense for Balbasaur are 45, 49, 49 respectively. If we have a level 30 Bulbasaur then its values of health point, attack and defense will be $45 + 30 \times 2 = 105$, $49 + 30 \times 1 = 79$ and $49 + 30 \times 1 = 79$ respectively.

3.2 Evolution

When a Pokémon reaches its minimum required level for evolution it can evolve. However, it is not a compulsory action. So there can be Pokémon of any kind at any level (level 10 Charizard or level 50 Charmander are both possible situations). But, Pokémon trainers always make their Pokémon evolve before tournaments if possible (see `pokemon_tournament` predicate).

3.3 Pokémon Types and Type Multipliers

All Pokémon creatures are assigned certain types. Each type has several strengths and weaknesses in both attack and defense. In battle, you should use Pokémon that have a type advantage over your opponent; doing so will cause much more damage than normal. Each Pokémon are assigned to at least one and at most two types.

A single-type advantage (for instance a Water-type Pokémon against a Ground-type Pokémon) will net you double normal damage. The advantages also "stack up", so a double-type advantage (for instance a Water-type Pokémon against a Ground/Rock-type Pokémon) will net you quadruple damage. Conversely, a single- and double-type disadvantage will afflict half and a quarter normal damage respectively.

For more information and visual version of the type chart: Pokémon Types. (We are not using attack types in this project, so don't confuse yourself with attack types or STAB from the link.)

4 Gotta Catch 'Em All

In this section, we will explain the predicates that you need to write for this project.

4.1 `find_pokemon_evolution(+PokemonLevel, +Pokemon, -EvolvedPokemon)`

This predicate is to find the evolved version of Pokémon given its level. If there is no evolution, then *EvolvedPokemon* = *Pokemon*. Pokémon can evolve two times if it has enough levels.

Example:

```
find_pokemon_evolution(40, charmeleon, EvolvedPokemon).
EvolvedPokemon = charizard.

find_pokemon_evolution(50, charmander, EvolvedPokemon).
EvolvedPokemon = charizard.
```

Note: charmander $\rightarrow^{lvl16}$ charmeleon $\rightarrow^{lvl36}$ charizard.

```
find_pokemon_evolution(10, charizard, EvolvedPokemon).
EvolvedPokemon = charizard.
```

4.2 pokemon_level_stats(+PokemonLevel, ?Pokemon, -PokemonHp, -PokemonAttack, -PokemonDefense)

This predicate evaluates the statistics of a Pokémon for the given level. With every level a Pokémon gains 2 health points, 1 attack point and 1 defense point. You can get the base statistics from *pokemon_stats*.

Example:

```
pokemon_level_stats(30, squirtle, PokemonHp, PokemonAttack, PokemonDefense).
PokemonHp = 104,
PokemonAttack = 78,
PokemonDefense = 95.
```

4.3 single_type_multiplier(?AttackerType, ?DefenderType, ?Multiplier)

This predicate will be used to find single-type advantage/disadvantage multiplier. It can also be used to find types that achieves a given multiplier.

Example:

```
single_type_multiplier(fire, grass, Multiplier).
Multiplier = 2.0;

single_type_multiplier(AttackerType, poison, 2.0).
AttackerType = ground;
AttackerType = psychic;
```

4.4 type_multiplier(?AttackerType, +DefenderTypeList, ?Multiplier)

This predicate will be used to find double-type advantage/disadvantage multiplier. It can also be used to find types that achieves a given multiplier.

Example:

```
type_multiplier(ice, [grass, ground], Multiplier).
Multiplier = 4.0;

type_multiplier(AttackerType, [grass, ground], 4.0).
AttackerType = ice;
```

4.5 pokemon_type_multiplier(?AttackerPokemon, ?DefenderPokemon, ?Multiplier)

This predicate will be used to find type multiplier between two Pokémon. It can also be used to find different attacker/defender Pokémon that achieves a given multiplier. If an attacker Pokémon has two types, then the Pokémon uses the type that gives the higher multiplier against the defender Pokémon.

Example:

```
pokemon_type_multiplier(bulbasaur, geodude, Multiplier).
Multiplier = 4.0;
```

Note: bulbasaur → (grass, poison), geodude → (rock, ground); grass vs (rock, ground) → Multiplier = 4.0, poison vs (rock, ground) → Multiplier = 0.25 ; $4.0 > 0.25 \rightarrow 4.0$

```
pokemon_type_multiplier(AttackerPokemon, charizard, 4.0).
AttackerPokemon = geodude;
AttackerPokemon = graveler;
AttackerPokemon = golem;
AttackerPokemon = onix;
```

4.6 pokemon_attack(+AttackerPokemon, +AttackerPokemonLevel, +DefenderPokemon, +DefenderPokemonLevel, -Damage)

This predicate finds the damage dealt from the attack of the AttackerPokemon to the DefenderPokemon.

Damage = (0.5 × AttackerPokemonLevel × (AttackerPokemonAttack / DefenderPokemonDefense) × TypeMultiplier) + 1

Example:

```
pokemon_attack(pikachu, 15, ekans, 12, Damage).
Damage = 10.375.
```

4.7 pokemon_fight(+Pokemon1, +Pokemon1Level, +Pokemon2, +Pokemon2Level, -Pokemon1Hp, -Pokemon2Hp, -Rounds)

This predicate simulates a fight between two Pokémon then finds health points of each Pokémon at the end of the fight and the number of rounds. Each Pokémon attacks at the same time and each attack sequence count as one round. After each attack, health points of each Pokémon reduced by the amount of calculated damage points. When a Pokémon's health points drop below zero, the fight ends.

Example:

```
pokemon_fight(pikachu, 15, ekans, 12, Pokemon1Hp, Pokemon2Hp, Rounds).
Pokemon1Hp = 11.872727272727257,
Pokemon2Hp = -3.25,
Rounds = 6.
```

4.8 pokemon_tournament(+PokemonTrainer1, +PokemonTrainer2, -WinnerTrainerList)

This predicate simulates a tournament between two Pokémon trainers then finds the winner Pokémon trainer of each fight. Pokémon trainers must have the same number of Pokémon. Pokémon fights in order. First Pokémon of the first Pokémon trainer fights with the first Pokémon of the second Pokémon trainer, second Pokémon of the first Pokémon trainer fights with the second Pokémon of the second Pokémon trainer.. A fight ends when a Pokémon's health points drop below zero. At the end of the fight, Pokémon with more health points win the fight, so does the Pokémon trainer that owns the winner Pokémon. **Important Note:** Pokémon trainers force their Pokémon to evolve (if possible) before tournament fights to gain maximum efficiency. So you should check evolution status of each Pokémon.

Example:

```
pokemon_tournament(ash, team_rocket, WinnerTrainerList).
WinnerTrainerList = [ash, team_rocket, team_rocket, ash].
```

4.9 best_pokemon(+EnemyPokemon, +LevelCap, -RemainingHP, -BestPokemon)

This predicate finds the best Pokémon against the given *EnemyPokemon* where the both Pokémon's levels are *LevelCap*. We define the best Pokémon as the Pokémon with the most remaining health points after the fight.

Example:

```
best_pokemon(charizard, 42, RemainingHP, BestPokemon).
RemainingHP = 144.17441860465118,
BestPokemon = golem.
```

4.10 best_pokemon_team(+OpponentTrainer, -PokemonTeam)

This predicate finds the best Pokémon Team against the given *OpponentTrainer* where the levels of each Pokémon of our best Pokémon Team are the same with the corresponding Opponent's Pokémon levels (e.g. Level of the first Pokémon of the best Pokémon Team is same with the level of the first Pokémon of the Opponent Trainer). We define the best Pokémon as the Pokémon with the most remaining health points after the fight.

Example:

```
best_pokemon_team(team_rocket, PokemonTeam)
PokemonTeam = [wigglytuff, mew, zapdos, golem].
```

4.11 pokemon_types(+TypeList, ?InitialPokemonList, -PokemonList)

This predicate will be used to find every Pokémon from *InitialPokemonList* that are at least one of the types from *TypeList*.

Example:

```
pokemon_types([grass, flying, ground], [bulbasaur, charmander, charizard, gyarados,
pikachu] , PokemonList).
PokemonList = [bulbasaur, charizard, gyarados].
```

4.12 generate_pokemon_team(+LikedTypes, +DislikedTypes, +Criterion, +Count, -PokemonTeam)

This predicate generates a Pokémon team based on liked and disliked types and some criteria. This team can only have Pokémon from *LikedTypes* and can't have Pokémon from *DislikedTypes*. The predicate sorts Pokémon according to one of the three criterion in descending order: health points (**h**), attack (**a**), defense (**d**). Then selects *Count* number of Pokémon that have highest values in the selected criterion. If two or more Pokémon has the same value, the order is not important between these Pokémon.

Example:

```
generate_team([dragon, fire, ghost, ice], [flying, ground, rock], a, 4, PokemonTeam).
PokemonTeam = [[cloyster, 50, 95, 180], [magmar, 65, 95, 57], [lapras, 130, 85, 80],
[dragonair, 61, 84, 65]].
```

5 Efficiency

Your code should run under thirty seconds for all the test cases collectively.

6 Documentation

You will be graded for the readability of your code, so don't skip steps or make your code complex, write as clear as possible. You will also be graded for the documentation of your code, so explain what each predicate is for in comments in your code. No extra documentation.

7 Submission

You are going to submit just one file named *pokemon.pl* to Moodle. First four lines of your pokemon.pl file must have exactly the lines below, since it will be used for compiling and testing your code automatically:

```
% name surname
% student id
% compiling:  yes
% complete:  yes
```

The third line denotes whether your code compiles correctly, and the fourth line denotes whether you completed all of the project, which must be "no" (without quotes) if you're doing a partial submission. This whole part must be lowercase and include only English alphabet.

8 Tips for the Project

- Try to formalize the problem, then try to convert the logic formulate to Prolog.
- You can use findall/3, bagof/3 or setof/3. Be careful when using bagof/3 and setof/3, and remember to set which free variables to ignore.
- Pay attention to the prefixes of the arguments in predicates which determines if the argument is a free variable or not (+, -, ?).
- You can use extra predicates for listing all the Pokémon or removing lists etc., but the ones given above are compulsory.
- Try to build complex predicates over the simpler ones, the project is designed to encourage that.
- If a predicate becomes too complex, either divide it into some predicates, or take another approach. Use debugging (through trace/1 and spy/1), approach your program systematically.

Çocukluk

Affan Dede'ye para saydım,
Sattı bana çocukluğumu.
Artık ne yaşım var, ne adım;
Bilmiyorum kim olduğumu.
Hiçbir şey sorulmasın benden;
Haberim yok olan bitenden.

Bu bahar havası, bu bahçe;
Havuzda su şırıl şırıldır.
Uçurtmam bulutlardan yüce,
Zıpzıplarım pırıl pırıldır.
Ne güzel dönüyor çemberim;
Hiç bitmese horoz şekerim!

Cahit Sıtkı Tarancı