Dicas Trabalho 1

Aula para disciplina de Métodos Formais

Gabriela Moreira

Departamento de Ciência da Computação - DCC
Universidade do Estado de Santa Catarina - UDESC

02 de outubro de 2024

Conteúdo

Outline

[Quint] q::debug

Se você está conseguindo executar (simular) sua especificação, mas não entende **porque** algo está se comportando de determinada maneira, pode ser útil usar o operador auxiliar `q::debug`. Por exemplo, considere a função:

```
1 pure def f(x) = (x + 1) * 2
```

Se tivermos dúvida se nossa soma está correta, podemos envolve-la em uma chamada de `q::debug`, onde o primeiro argumento é uma string qualquer e o segundo a expressão a ser avaliada. O resultado de `q::debug` é exatamente igual à expressão em si, mas ele tem um efeito colateral que resulta na impressão daquele valor precedido pela string fornecida:

```
1 >>> pure def f(x) = q::debug("resultado soma: ", x +
      1) * 2
2 >>> f(1)
3 > resultado soma:  2
4 4
```

Isso é o mais próximo que temos de `printf` para debugar :). Você

[TLA+] Print

De forma semelhante, temos o operator `Print` em TLA+, definido no módulo TLC

```
1 EXTENDS TLC
```

Usado da mesma forma:

```
1 otherTeam(team) == Print("otherTeam: ", IF team = "A"
      THEN "B" ELSE "A")
```

No VSCode, no painel do TLC teremos uma seção “Output” mostrando:

```
"otherTeam: " "A" (6)
"otherTeam: " "B" (2)
"otherTeam: " "A" (4)
"otherTeam: " "B" (4)
"otherTeam: " "A" (2)
...
```

Composição

E se, no mesmo turno, um pokemon puder dar dois ataques em vez de somente um?

Composição

E se, no mesmo turno, um pokemon puder dar dois ataques em vez de somente um?

Lembrem-se que não podemos atualizar uma variável duas vezes na mesma ação (transição).

Composição

E se, no mesmo turno, um pokemon puder dar dois ataques em vez de somente um?

Lembrem-se que não podemos atualizar uma variável duas vezes na mesma ação (transição).

Temos duas opções:

1. Usar duas ações e só atualizar `next_team` na segunda
2. Usar uma unica ação, onde só atualizamos `pokemons` com o resultado do segundo ataque.

Composição

E se, no mesmo turno, um pokemon puder dar dois ataques em vez de somente um?

Lembrem-se que não podemos atualizar uma variável duas vezes na mesma ação (transição).

Temos duas opções:

1. Usar duas ações e só atualizar `next_team` na segunda
2. Usar uma unica ação, onde só atualizamos `pokemons` com o resultado do segundo ataque.

A opção (2) é geralmente mais interessante, já que ela funciona melhor se, por exemplo, quisermos adicionar um terceiro ataque.

[Exemplo completo no moodle]

```
1 type AttackDescription = { attacker: Pokemon, receiver
      : Pokemon, attack: str, damage: int }
2 var last_attack: Option[List[AttackDescription]]
3
4 pure def tackle(attacker: Pokemon, receiver: Pokemon):
       (Pokemon, AttackDescription) = {
5   (receiver.damage(10), { attacker: attacker, receiver
```

[Quint] Tipos soma - salvando o último ataque

E se precisássemos registrar o modificador de dano de ataques elementais na nossa especificação de pokemons?

[Quint] Tipos soma - salvando o último ataque

E se precisássemos registrar o modificador de dano de ataques elementais na nossa especificação de pokemons?

- O problema aqui é que ataques do tipo `tackle` não tem modificador. Esse dado só existe para ataques elementais.

[Quint] Tipos soma - salvando o último ataque

E se precisássemos registrar o modificador de dano de ataques elementais na nossa especificação de pokemons?

- O problema aqui é que ataques do tipo `tackle` não tem modificador. Esse dado só existe para ataques elementais.
- Poderíamos colocar um valor qualquer no campo `modifier` quando o ataque é do tipo `tackle`, e lembrar que esse campo é irrelevante quando o tipo do ataque é `tackle`.

[Quint] Tipos soma - salvando o último ataque

E se precisássemos registrar o modificador de dano de ataques elementais na nossa especificação de pokemons?

- O problema aqui é que ataques do tipo `tackle` não tem modificador. Esse dado só existe para ataques elementais.
- Poderíamos colocar um valor qualquer no campo `modifier` quando o ataque é do tipo `tackle`, e lembrar que esse campo é irrelevante quando o tipo do ataque é `tackle`.

A alternativa mais elegante é usar um tipo soma para, no mesmo campo, termos tipos diferentes para um ataque elemental (aqui, um *record* com o modificador e o dano) e um `tackle` (aqui, um inteiro para o dano).

```
type Attack =
  | ElementalAttack({ modifier: DamageModifier, damage: int })
  | Tackle(int)
type AttackDescription = { attacker: Pokemon, receiver: Pokemon,
    attack: Attack }
```

[Quint] Tipos soma - representando mais ações

Se quisermos deixar nossa especificação de pokemons mais completa, podemos adicionar outras ações que um pokemon pode tomar no seu turno, além de atacar. Vamos pensar somente nos tipos aqui.

[Quint] Tipos soma - representando mais ações

Se quisermos deixar nossa especificação de pokemons mais completa, podemos adicionar outras ações que um pokemon pode tomar no seu turno, além de atacar. Vamos pensar somente nos tipos aqui.

```
type AttackDescription = ElementalAttack({ modifier: DamageModif
    , damage: int }) | Tackle(int)
type Status = Attack | Defense | SpecialDefense | SpecialAttack
    Speed | Accuracy | Evasiveness
type Condition = Poison | Burn | Freeze | Paralize | Sleep
type TurnDescription =
    | Attack({ attacker: Pokemon, receiver: Pokemon, attack:
    AttackDescription })
    | Heal({ pokemon: Pokemon, healed_amount: int })
    | Buff({ pokemon: Pokemon, status: Status })
    | Debuff({ caster: Pokemon, receiver: Pokemon, status: Status
    | ApplyCondition({ caster: Pokemon, receiver: Pokemon, condit
    : Condition })
    | RemoveCondition({ pokemon: Pokemon, condition: Condition })
```

[TLA+] Representando ações diferentes

Em TLA+ (usando TLC), não temos tipos, então é possível usar *records* com diferentes campos em cada estado.
Por exemplo:

```
1 lastAttack' =[ attacker |-> p1, receiver |-> p2,
    attack |-> "elemental", modifier |-> "super
    effective", damage |-> 20]
2 lastAttack' = [ action |-> "buff", pokemon |-> p,
    status |-> "speed" ]
```

Controle de turnos

O nosso controle de turnos na especificação dos pokemons e do jogo da velha funciona muito bem para 2 jogadores, mas não é muito útil com mais do que isso.

Controle de turnos

O nosso controle de turnos na especificação dos pokemons e do jogo da velha funciona muito bem para 2 jogadores, mas não é muito útil com mais do que isso.

```
1 action init = {
2   // ...
3   next_team' = if (team_A_pokemon.speed >=
      team_B_pokemon.speed) "A" else "B",
4 }
5
6 action step = {
7   // ...
8   next_team' = other_team(next_team),
9 }
```

Controle de turnos para N jogadores

Para não precisar salvar muita informação extra no estado (variáveis), uma alternativa legal é ter um contador de *rounds* e, a cada turno, computar quem é o jogador da vez naquele *round*.

Controle de turnos para N jogadores

Para não precisar salvar muita informação extra no estado (variáveis), uma alternativa legal é ter um contador de *rounds* e, a cada turno, computar quem é o jogador da vez naquele *round*.

```
1  var round: int
2
3  action init = all {
4    pokemons' = ...,
5    round' = 0,
6  }
7
8  action step = {
9    val attackers_by_initiative = pokemons.values().
       toList((p1, p2) => intCompare(p2.speed, p1.speed))
10   val attacker = attackers_by_initiative[round %
       pokemons.keys().size()]
11   val receiver = pokemons.values().filter(p =>
       attacker != p).oneOf()
12   all {
13     attack(attacker, receiver),
14     round' = round + 1,
```

Entendendo a ordenação

```
1 val attackers_by_initiative = pokemons.values().toList
    ((p1, p2) => intCompare(p2.speed, p1.speed))
```

Entendendo a ordenação

```
1 val attackers_by_initiative = pokemons.values().toList
    ((p1, p2) => intCompare(p2.speed, p1.speed))
```

- `toList`: Converte um conjunto (`set`) para uma lista. Precisa de um operador que indique como ordenar os elementos desse set, para que não tenhamos comportamento arbitrário.

Entendendo a ordenação

```
val attackers_by_initiative = pokemons.values().toList
    ((p1, p2) => intCompare(p2.speed, p1.speed))
```

- `toList`: Converte um conjunto (`set`) para uma lista. Precisa de um operador que indique como ordenar os elementos desse set, para que não tenhamos comportamento arbitrário.
- `intCompare`: Operador auxiliar para comparação de inteiros

```
  pure def intCompare(a: int, b:int): Ordering = {
    if (a < b)
      { LT }
    else if (a > b)
      { GT }
    else
      { EQ }
  }
```

Entendendo a ordenação

```
val attackers_by_initiative = pokemons.values().toList
    ((p1, p2) => intCompare(p2.speed, p1.speed))
```

- `toList`: Converte um conjunto (`set`) para uma lista. Precisa de um operador que indique como ordenar os elementos desse set, para que não tenhamos comportamento arbitrário.
- `intCompare`: Operador auxiliar para comparação de inteiros

```
  pure def intCompare(a: int, b:int): Ordering = {
    if (a < b)
      { LT }
    else if (a > b)
      { GT }
    else
      { EQ }
  }
```

- Para que a ordem seja descendente de acordo com a velocidade, basta inverter a ordem dos argumentos (`intCompare(p2.speed, p1.speed)`).

Equivalente em TLA+

Para converter um conjunto em uma sequência com uma dada ordenação em TLA+, podemos usar o operador `SetToSortSeq` do módulo `SequencesExt`:

```
1 EXTENDS SequencesExt
2
3 SetToSortSeq({ [ speed |-> 10 ], [ speed |-> 20 ] },
    LAMBDA r1, r2 : r1.speed > r2.speed)
```

Transformando todos os valores de um mapa

Supondo que queremos aplicar 1 de dano em todos os pokemons.

Transformando todos os valores de um mapa

Supondo que queremos aplicar 1 de dano em todos os pokemons.

Em Quint:

```
1 pokemons.transformValues(p => p.damage(1))
```

Transformando todos os valores de um mapa

Supondo que queremos aplicar 1 de dano em todos os pokemons.

Em Quint:

```
pokemons.transformValues(p => p.damage(1))
```

Em TLA+

```
[ team \in DOMAIN pokemons |-> damage(pokemons[team],
    1) ]
```

Transformando todos os valores de um mapa

Supondo que queremos aplicar 1 de dano em todos os pokemons.

Em Quint:

```
pokemons.transformValues(p => p.damage(1))
```

Em TLA+

```
[ team \in DOMAIN pokemons |-> damage(pokemons[team],
    1) ]
```

A versão em Quint equivalente a essa expressão em TLA+ seria:

```
pokemons.keys().mapBy(team => pokemons.get(team).
    damage(1))
```

Mapas indexam valores com chaves

Mapas mapeiam um conjunto de chaves para seus valores.

Mapas indexam valores com chaves

Mapas mapeiam um conjunto de chaves para seus valores.

- Na especificação dos pokemons, escolhemos indexar pokemons pelo seu time (`"A"` ou `"B"`).

Mapas indexam valores com chaves

Mapas mapeiam um conjunto de chaves para seus valores.

- Na especificação dos pokemons, escolhemos indexar pokemons pelo seu time (`"A"` ou `"B"`).
  - Ou seja, nossas chaves são times (*strings*), e nossos valores são pokemons.

Mapas indexam valores com chaves

Mapas mapeiam um conjunto de chaves para seus valores.

- Na especificação dos pokemons, escolhemos indexar pokemons pelo seu time (`"A"` ou `"B"`).
  - Ou seja, nossas chaves são times (*strings*), e nossos valores são pokemons.
  - Em Quint, o tipo do mapa é `str -> Pokemon`

Mapas indexam valores com chaves

Mapas mapeiam um conjunto de chaves para seus valores.

- Na especificação dos pokemons, escolhemos indexar pokemons pelo seu time (`"A"` ou `"B"`).
  - Ou seja, nossas chaves são times (*strings*), e nossos valores são pokemons.
  - Em Quint, o tipo do mapa é `str -> Pokemon`
- Isso foi possível porque temos apenas um pokemon por time. Se tivéssemos mais de um pokemon por time, teríamos que escolher outra chave. Por exemplo, o nome do pokemon.

Mapas indexam valores com chaves

Mapas mapeiam um conjunto de chaves para seus valores.

- Na especificação dos pokemons, escolhemos indexar pokemons pelo seu time (`"A"` ou `"B"`).
  - Ou seja, nossas chaves são times (*strings*), e nossos valores são pokemons.
  - Em Quint, o tipo do mapa é `str -> Pokemon`
- Isso foi possível porque temos apenas um pokemon por time. Se tivéssemos mais de um pokemon por time, teríamos que escolher outra chave. Por exemplo, o nome do pokemon.
  - Mapas são equivalentes a funções. Uma chave só pode mapear um único valor

Mapas indexam valores com chaves

Mapas mapeiam um conjunto de chaves para seus valores.

- Na especificação dos pokemons, escolhemos indexar pokemons pelo seu time (`"A"` ou `"B"`).
  - Ou seja, nossas chaves são times (*strings*), e nossos valores são pokemons.
  - Em Quint, o tipo do mapa é `str -> Pokemon`
- Isso foi possível porque temos apenas um pokemon por time. Se tivéssemos mais de um pokemon por time, teríamos que escolher outra chave. Por exemplo, o nome do pokemon.
  - Mapas são equivalentes a funções. Uma chave só pode mapear um único valor
  - Nesse caso, usamos time ao invés de nome por isso facilitar o acesso ao "pokemon adversário" (`pokemons.get(other_team(next_team))`)

[TLA+] Modo simulação do TLC

Como indicado no enunciado do trabalho, devemos usar simuladores.
Para usar o TLC no seu modo de simulação:

1. Instale a extensão do TLA+ no VSCode.
2. Aperte F1 e escolha: TLA+: Check model with TLC
   - Você deve ver um prompt para inserir opções adicionais para o TLC.

[TLA+] Modo simulação do TLC

Como indicado no enunciado do trabalho, devemos usar simuladores. Para usar o TLC no seu modo de simulação:

1. Instale a extensão do TLA+ no VSCode.
2. Aperte F1 e escolha: TLA+: Check model with TLC
   - Você deve ver um prompt para inserir opções adicionais para o TLC.

PS: Aconselho sempre usar a flag `-deadlock` para desativar checks de deadlock, já que essa não é uma preocupação para nossos sistemas de batalha.

[TLA+] Modo simulação do TLC

Como indicado no enunciado do trabalho, devemos usar simuladores. Para usar o TLC no seu modo de simulação:

1. Instale a extensão do TLA+ no VSCode.
2. Aperte F1 e escolha: TLA+: Check model with TLC
   - Você deve ver um prompt para inserir opções adicionais para o TLC.

PS: Aconselho sempre usar a flag `-deadlock` para desativar checks de deadlock, já que essa não é uma preocupação para nossos sistemas de batalha.

Opções para simular até encontrar um erro, com no máximo 10 mil tentativas (semelhante ao `quint run`):

```
1 -deadlock -simulate num=10000
```

[TLA+] Modo simulação do TLC

Como indicado no enunciado do trabalho, devemos usar simuladores. Para usar o TLC no seu modo de simulação:

1. Instale a extensão do TLA+ no VSCode.
2. Aperte F1 e escolha: TLA+: Check model with TLC
   - Você deve ver um prompt para inserir opções adicionais para o TLC.

PS: Aconselho sempre usar a flag `-deadlock` para desativar checks de deadlock, já que essa não é uma preocupação para nossos sistemas de batalha.

Opções para simular até encontrar um erro, com no máximo 10 mil tentativas (semelhante ao `quint run`):

```
-deadlock -simulate num=10000
```

Se a propriedade for satisfeita, nenhum trace será exibido. Então, se você quiser ver um trace qualquer, use:

```
-deadlock -simulate file=out,num=1
```

Assim, o TLC irá criar um arquivo `out_0_0` com um trace do modelo.

[Quint] nondet e oneOf

```
action x_recebe_algum_valor = {
  nondet valor = Set(1, 2, 3).oneOf()
  x' = valor
}
```

O operador `oneOf()` só pode ser usado nessa forma!

- Como parte de uma ação (`action`)
- Em uma definição do tipo `nondet`
- Sem mais operadores aplicados acima dele
  - (i.e. `nondet valor = Set(1, 2, 3).oneOf() + 1`)

[Quint] nondet e oneOf II

Por que isso é importante?

- Quint obedece a lógica temporal das ações e pode ser traduzido para TLA+
- Se usamos `oneOf()` fora desse padrão, temos potencialmente definições fora da lógica, que não podem ser traduzidas para TLA+
- Lembrando que a versão em TLA+ para isso é o existe, cujo corpo é sempre booleano.

[Quint] nondet e oneOf II

Por que isso é importante?

- Quint obedece a lógica temporal das ações e pode ser traduzido para TLA+
- Se usamos `oneOf()` fora desse padrão, temos potencialmente definições fora da lógica, que não podem ser traduzidas para TLA+
- Lembrando que a versão em TLA+ para isso é o existe, cujo corpo é sempre booleano.

Atenção: isso significa que não é possível definir "role um d20 para cada criatura". Vocês vão precisar definir um valor `nondet` para cada uma das criaturas:

```
action init = all {
  nondet d20_mago = // ...
  nondet d20_druida = // ...
  // ...
}
```