Implementing UNO using Maude

Luca Fuligni
Michele Russo
Marco Zamponi

https://github.com/ZamponiMarco/Uno

What is UNO?

- UNO is a *shedding-type* card game;
- Developed in 1971, owned by Mattel since 1992;
- Deck of 108 cards divided in *DRAW Pile* and *DISCARD Pile*
- The number of players varies from 2 to 10;
- Each player starts with 7 cards.

Code structure

We have divided the main concepts of the game as:

- game = < A, B, C, drawPile, discardPile, played, drawn, counter >
- player = < label, hand >
- hand = { $c_0^{m(c0)}$, ... , $c_n^{m(cn)}$ }
- deck = (c_0, c_1, ... , c_n)

Deck composition

The deck is composed of 108 cards of different colors (red, green, blue, yellow).
For each color we have:

- Numbers from 0 to 9
- Skip
- Reverse
- Draw Two (+2)

All numbers and types occur two times, except the 0 which is unique for every color.

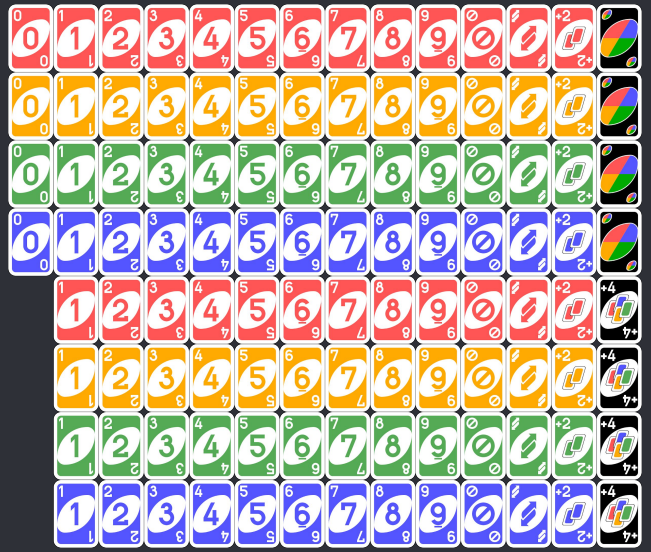
Then we have also four other color-indipendent cards, two for each type:

- Wild
- Wild Draw Four (+4)

Code structure

The code structure for a card is:

- card = < type, color >
- type = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, reverse, stop, plus2, plus4, change }
- color = { red, green, yellow, blue, all }



Rewriting rules - Throw

The throw rules all behave in the same way: they throw a card into the *DISCARD Pile* but do not actually change the turn, they are then followed by the action rules.

```
crl [throw-color] :
      game(A' = Ca' ~ H, Bp, Cp, D, Ca | G, false, P, 0) => game(A' = H, Bp, Cp, D, Ca' | Ca | G, true,
P, 0)
            if getColor(Ca') == getColor(Ca) and not isFinished(A' = Ca' ~ H, Bp, Cp) .

crl [throw-type] :
      game(A' = Ca' ~ H, Bp, Cp, D, Ca | G, false, P, 0) => game(A' = H, Bp, Cp, D, Ca' | Ca | G, true,
P, 0)
            if getType(Ca') == getType(Ca) and not getColor(Ca) == all and not isFinished(A' = Ca' ~
H, Bp, Cp) .
```

Rewriting rules - Throw

The throw rules all behave in the same way: they throw a card into the *DISCARD Pile* but do not actually change the turn, they are then followed by the action rules.

```
crl [throw-changecolor] :
      game(A' = Ca' ~ H, Bp, Cp, D, G, false, P, 0) => game(A' = H, Bp, Cp, D, (card(change,
getMostFrequentColor(H)) | G), true, P, 0)
            if getType(Ca') == change and not isFinished(A' = Ca' ~ H, Bp, Cp) .
```

Rewriting rules - Throw

The throw rules all behave in the same way: they throw a card into the *DISCARD Pile* but do not actually change the turn, they are then followed by the action rules.

```
crl [throw-plusfour] :
      game(A' = Ca' ~ H, Bp, Cp, D, Ca | G, false, P, 0) => game(A' = H, Bp, Cp, D, card(plus4,
getMostFrequentColor(H)) | Ca | G, true, P, 0)
            if getType(Ca') == plus4 and not containsColor(H, getColor(Ca)) and not isFinished(A' = Ca'
~ H, Bp, Cp)
```

Rewriting rules - Action

All the action rules do is look at the first card of the *DISCARD Pile* and act differently according to it. Consequently there is one of them for each type of card.

```
crl [turn-action-reverse] :
      game(Ap,Bp,Cp,D,Ca | G, true, P, 0) => game(Cp,Bp,Ap,D,Ca | G, false, false, 0)
            if getType(Ca) == reverse and not isFinished(Ap, Bp, Cp) .

crl [turn-action-stop] :
      game(Ap, Bp, Cp, D, Ca | G, true, P, 0) => game(Cp, Ap, Bp, D, Ca | G,false, false, 0)
            if getType(Ca) == stop and not isFinished(Ap, Bp, Cp) .
```

Rewriting rules - Action

All the action rules do is look at the first card of the *DISCARD Pile* and act differently according to it. Consequently there is one of them for each type of card.

```
crl [turn-action-plustwo] :
      game(Ap, Bp, Cp, D, Ca | G, true, P, 0) => game(Bp, Cp, Ap, D, Ca | G, false, false, 2)
            if getType(Ca) == plus2 and not isFinished(Ap, Bp, Cp) .

crl [turn-action-plusfour] :
      game(Ap, Bp, Cp, D, Ca | G, true, P, 0) => game(Bp, Cp, Ap, D, Ca | G, false, false, 4)
            if getType(Ca) == plus4 and not isFinished(Ap, Bp, Cp) .
```

Rewriting rules - Action

The turn rules, given a player, simply manage the passage of the turn by changing the arrangement of players within the structure representing the game.

```
crl [turn-number] :
      game(Ap, Bp, Cp, D, Ca | G, true, P, 0) => game(Bp, Cp, Ap, D, Ca | G, false, false, 0)
            if (isNumber(Ca) or getType(Ca) == change) and not isFinished(Ap, Bp, Cp) .

crl [turn-nomove] :
      game(Ap, Bp, Cp, D, Ca' | G, false, true, 0) => game(Bp, Cp, Ap, D, (Ca' | G), false, false,
0)
            if not possibleMove(getHand(Ap), Ca') and not isFinished(Ap, Bp, Cp) .
```

Rewriting rules - Draw

There are two types of draws, the simple one in case there are no other cards to play and a forced one that follows the +2 or +4.

- Simple draw:

```
crl [draw] :
      game(A' = A, Bp, Cp, (Ca | D), (Ca' | G), false, false, 0) => game(A' = A ~ Ca, Bp, Cp,
D, (Ca' | G), false, true, 0)
            if not possibleMove(A, Ca') and not isFinished(A' = A, Bp, Cp) .
```

Rewriting rules - Draw

- Forced draw:

```
crl [draw-forced] :
      game(A' = A, Bp, Cp, Ca | D, G, false, false, Cou) => game(A' = Ca ~ A, Bp, Cp, D, G,
false, false, Cou - 1)
            if Cou > 1 and not isFinished(A' = A, Bp, Cp) .


crl [stop-draw-forced] :
      game(A' = A, Bp, Cp, Ca | D, G, false, false, Cou) => game(Bp, Cp, A' = Ca ~ A, D, G,
false, false, Cou - 1)
            if Cou == 1 and not isFinished(A' = A, Bp, Cp) .
```

Rewriting rules - Shuffle

It is particularly complicated to perform a random shuffle: the level of complexity increases exponentially.
We have solved the problem by building a fixed deck from which the cards still in play are removed.

```
crl [shuffle] :
      game(Ap, Bp, Cp, D, (Ca | G), P, P', Cou) => game(Ap, Bp, Cp,
removeCard(removeHand(removeHand(removeHand(shuffleDeck, getHand(Cp)), getHand(Bp)), getHand(Ap)),
Ca), Ca | >, P, P', Cou)
      if D == > and not isFinished(Ap, Bp, Cp) .
```

Used rules

For the development of this project the official Uno rules have been used.

At the beginning of the project we built the game using a popular variant of the game rules in which the cards “Draw Two” and “Stop” could have a cumulative effect, which isn’t allowed in the official rules.

A copy of the official rules and the previous version developed are available in the GitHub repository: https://github.com/ZamponiMarco/Uno

Given an initial configuration, is there a game strategy that makes a certain player to win? What is it?

Definition of an example game

With the help of a python script we generated two random shuffled decks in the format already defined in Maude. Such items will be stored in operations named:

- `shuffleDeck`
- `drawDeck`

We then created three hands for each one of the three players by draining seven cards per time from the drawDeck, these hands have been defines in the operations named:

- `handA`
- `handB`
- `handC`

Then we drained another card and put it inside the operation:

- `discardPile`

The specification of our example game, put inside the operation exampleGame, is:

```
exampleGame = game('A = handA, 'B = handB, 'C = handC, drawDeck, discardPile, false, false, 0) .
```

Winner of a match

To find one game strategy that makes the player labelled as *'A* win we use the command:

```
search [1] exampleGame =>* G:Game such that hasWon(G:Game, 'A) .
```

The output should result in something like:

```
Solution 1 (state 124450)
states: 124451  rewrites: 39680306 in 0ms cpu (6002ms real) (~ rewrites/second)
G:Game --> game('A = empty, 'B = … , 'C = … , card(8, yellow) | … | >, card(stop, red) | … | >,
true, false, 0)
```

To find the path of rewriting rules used from the solution 1 we use:

```
show path 124450 .
```