CS 839 Systems Verification

Lecture 4: Abstraction

Learning outcomes

1. Separate implementation from abstraction
2. Extract an abstraction from an informal description

What is the abstraction of a queue?

Think about both *state* and *operations* (enqueue, dequeue, isEmpty).

Warmup exercise: how would you formalize what a queue data structure does?

Queue Abstract Data Type

state: `list V`

`enqueue(v)` op: changes state from `xs` to `xs ++ [v]`

`dequeue()` op: changes state from `x :: xs` to `xs` and returns `x` (errors if empty?)

`isEmpty()`: returns true iff state is `[]`

What should the behavior of dequeue be on an empty queue? This is a matter of choice: we could consider it an error and throw an exception, we could simply not specify it (allowing undefined or arbitrary behavior), or we could have dequeue return a boolean indicating success.

Benefits of abstraction

Ask whole class to list some things

Some ideas if not mentioned:

hide the implementation details; easier to understand how to use it
allow multiple implementations based on hardware, performance needs
allow changing the implementation in the future
teamwork: separate working on implementation vs client
better reuse: find a generic component and use it for multiple things
maybe more amenable to testing than alternatives (easily isolated)

Stack Abstract Data Type

state: `list V`

push(v) op: changes state from `xs` to `v :: xs`

pop() op: changes state from `x :: xs` to `xs` and returns `x` (errors if empty?)

Another ADT with different semantics is the stack. Similar to the queue, this interface permits a variety of implementations.

Example: the file-system abstraction

The file system provides an abstraction of the storage (typically a disk). The state is something like a tree of directories, containing files, with metadata associated with directories and files (timestamps, permissions), and files containing bytes.

It's a bit more complicated in that there are special files (symbolic links, devices). Hard links complicate the model, too; we need a notion of the physical identity of a file (almost requiring inodes) so that two links can point to the same underlying file.

The operations include things like `open`, which creates a file-descriptor in the calling process - so we need to also talk about processes to fully describe the state.

That's just state. The operations are a bit simpler; it's easy to develop a mental model of what `readdir`, `read`, and `write` do, as well as `open` flags and `unlink`.

This abstraction hides significant implementation complexity, especially indirect blocks to support large files, tracking free inodes and blocks, and journaling to make the implementation crash safe.

Example: The process abstraction

Processes are also an abstraction provided by the OS. They're a bit more complicated to describe as state + operations, in that the fundamental thing a process does is execute code, seemingly arbitrary CPU instructions. However, it also has special *system calls* that behave more like the abstractions above; these all are high-level abstractions on top of the lower-level primitives in the OS.

5-min break

Activity: formalization of Hanabi

Why are we doing this? It's a fun way to practice abstraction and formalization. We will take a written explanation - which in this case are game rules that are supposed to be complete since you need to follow the rules by hand - and turning it into something abstract and precise.


Hanabi

Antoine Bauza

2 to 5 players, Ages 8+

Contents:

- 60 Hanabi cards
  6 SUITS, each with
  Three **1's**, *Two* **2's**, *Two* **3's**, *Two* **4's**, *One* **5**
- 8 Blue Clock tokens
- 4 Black Fuse tokens

Summary

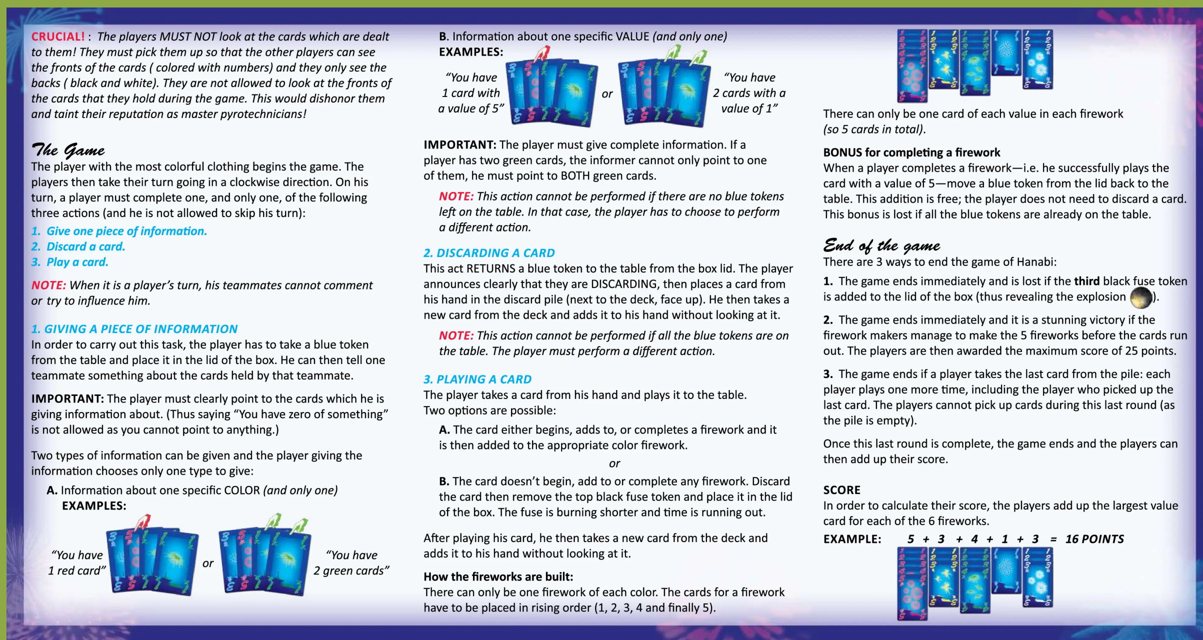
Hanabi is a cooperative game, i.e. a game where the players do not play against each other but work together towards a common goal. In this case they are absent minded firework manufacturers who accidentally mixed up powders, fuses and rockets from a firework display. The show is about to start and panic is setting in. They have to work together to stop the show from becoming a disaster! The pyrotechnicians have to put together 5 fireworks, (1 white, 1 red, 1 blue, 1 yellow, 1 green), by making a series rising in number (1, 2, 3, 4, 5) with the same colored cards.

Getting Ready

Place the 8 blue clock tokens face up on the table. Stack the 4 black Fuse tokens nearby with the longest fuse on top and the explosion on the bottom. Shuffle the 50 cards to make a deck and put them face down. Note, a 6th multicolor suit is included in the game, but it is for advanced play only, please see Variants section.

Deal a hand of 5 cards to each with 2 or 3 players.
Deal a hand of 4 cards if there are 4 or 5 players.

Hanabi is a cooperative game where players take turns making moves to complete five stacks of cards in order. The catch is that you cannot see your cards, only those of others, and there are rules for how you can give people hints about their cards.



CRUCIAL! : *The players MUST NOT look at the cards which are dealt to them! They must pick them up so that the other players can see the fronts of the cards (colored with numbers) and they only see the backs (black and white). They are not allowed to look at the fronts of the cards that they hold during the game. This would dishonor them and taint their reputation as master pyrotechnicians!*

The Game

The player with the most colorful clothing begins the game. The players then take their turn going in a clockwise direction. On his turn, a player must complete one, and only one, of the following three actions (and he is not allowed to skip his turn):

1. *Give one piece of information.*
2. *Discard a card.*
3. *Play a card.*

NOTE: *When it is a player's turn, his teammates cannot comment or try to influence him.*

1. GIVING A PIECE OF INFORMATION

In order to carry out this task, the player has to take a blue token from the table and place it in the lid of the box. He can then tell one teammate something about the cards held by that teammate.

IMPORTANT: The player must clearly point to the cards which he is giving information about. (Thus saying "You have zero of something" is not allowed as you cannot point to anything.)

Two types of information can be given and the player giving the information chooses only one type to give:

A. Information about one specific COLOR *(and only one)*

EXAMPLES:

"You have 1 red card" *or* *"You have 2 green cards"*

B. Information about one specific VALUE *(and only one)*

EXAMPLES:

"You have 1 card with a value of 5" *or* *"You have 2 cards with a value of 1"*

IMPORTANT: The player must give complete information. If a player has two green cards, the informer cannot only point to one of them, he must point to BOTH green cards.

NOTE: *This action cannot be performed if there are no blue tokens left on the table. In that case, the player has to choose to perform a different action.*

2. DISCARDING A CARD

This act RETURNS a blue token to the table from the box lid. The player announces clearly that they are DISCARDING, then places a card from his hand in the discard pile (next to the deck, face up). He then takes a new card from the deck and adds it to his hand without looking at it.

NOTE: *This action cannot be performed if all the blue tokens are on the table. The player must perform a different action.*

3. PLAYING A CARD

The player takes a card from his hand and plays it to the table.
Two options are possible:

A. The card either begins, adds to, or completes a firework and it is then added to the appropriate color firework.

or

B. The card doesn't begin, add to or complete any firework. Discard the card then remove the top black fuse token and place it in the lid of the box. The fuse is burning shorter and time is running out.

After playing his card, he then takes a new card from the deck and adds it to his hand without looking at it.

How the fireworks are built:

There can only be one firework of each color. The cards for a firework have to be placed in rising order (1, 2, 3, 4 and finally 5).

There can only be one card of each value in each firework *(so 5 cards in total).*

BONUS for completing a firework

When a player completes a firework—i.e. he successfully plays the card with a value of 5—move a blue token from the lid back to the table. This addition is free; the player does not need to discard a card. This bonus is lost if all the blue tokens are already on the table.

End of the game

There are 3 ways to end the game of Hanabi:

1. The game ends immediately and is lost if the **third** black fuse token is added to the lid of the box (thus revealing the explosion).

2. The game ends immediately and it is a stunning victory if the firework makers manage to make the 5 fireworks before the cards run out. The players are then awarded the maximum score of 25 points.

3. The game ends if a player takes the last card from the pile: each player plays one more time, including the player who picked up the last card. The players cannot pick up cards during this last round (as the pile is empty).

Once this last round is complete, the game ends and the players can then add up their score.

SCORE

In order to calculate their score, the players add up the largest value card for each of the 6 fireworks.

EXAMPLE: ***5 + 3 + 4 + 1 + 3 = 16 POINTS***

The goal is to play the numbers 1--5 of each color (white, blue, red, yellow, green), in order. When every card has been drawn, the game gives one more round and then ends; you score a point for every card you have played.

Players take turns. On your turn you do one of the following:

Give a hint to another player. A hint might be *"these cards are 1's"* or *"these cards are all yellow"*. You *must* point out all the cards matching the description. You *must* specify exactly one number or exactly one color in your hint. You cannot give 0-number hints, e.g., *"none of your cards are 3's"*. Hints cost tokens, of which there are a limited number.
Play a card. If the card is playable (that is, it is the next number in one of the color stacks), then put it in the right place. If it isn't, discard the card and lose a life. If you complete a stack (that is, play a five correctly), then you regain a hint token.
Discard a card. This regains a hint token.

Task: describe the abstract state machine of Hanabi

Too much to do in 30min

Focus on describing the **play** action

Example: modeling Tic-tac-toe

location: {

row: int // [0, 2]

col: int // [0, 2]

}

player := black | white

cell := empty | full(p: player)

current_player: state -> player

game_over: state -> Prop

full or someone won

full: forall l, s(l) = full(_)

move: state -> state -> Prop

not(game_over(s))

exists l, s(l) = empty /\ exists (p: player),

p = current_player(s) /
s' = s[l := full(p)]