The Pip Language Proposal

Programming Languages and Translators (W4115), Fall 2008

Frank Wallingford (frw2106@columbia.edu)

Introduction

Pip will be a language that can be used to implement card games. It will contain types, expressions, and routines that will include features to facilitate the manipulation of cards, ranks, suits, decks, playing areas, and players.

Description

The Pip compiler will take as input the rules of a card game written in the Pip language and translate them into instructions for a card game engine to execute. As the card game engine executes the Pip program, one or more users interacting with the engine will be able to play the card game. The card game engine will deal cards, display hands and playing areas, advance through player turns, and allow only valid cards to be played at any given time, based on logic coded up in the Pip program. The card game engine will also be able to determine if the game has ended and who the game winner is.

A simple textual interface will be implemented to facilitate testing and to allow card games to be played in a rudimentary fashion. A graphical interface could be implemented which would allow for a more traditional interface, but that is outside the scope of this project.

There will be no mechanism for implementing artificial intelligence for a player. The Pip language and card game engine will only allow physical players to participate in the card games.

Types

Pip will contain several built-in types. Among them will probably be:

- Rank
- Suit
- Card
- String
- Number
- Boolean
- List (an ordered sequence of any type)
- Collection (a group of name-value properties)

Expressions

Expressions will consist of operators, identifiers, and constants. Common tasks such as list manipulation, testing for list membership, and querying and setting collection properties will done with the built-in operators.

```
card in deck // The "in" operator returns a Boolean
deck[0]      // List member access
deck + card  // List addition; results in new list
deck - card  // List deletion; results in new list
game.name    // Property access; See "Collections" below
```

Cards, including Rank and Suit, will have a direct representation in Pip.

```
A            // Ace, a Rank
S            // Spades, a Suit
A/C          // Ace of Clubs, a Card
2-9/H        // 2 through 9 of Hearts, a List of Cards
*/D          // All Diamonds
3,4,5/*      // All 3's, 4's, and 5's
*/*          // A 52-Card deck
```

The slash combines a Rank and Suit into a Card. If either side of a slash is the asterisk, then the slash results in a list of Cards that match the wildcard.

Predicates

Predicates will be like simple routines that take parameters and evaluate to a Boolean result. They are meant to be short pieces of logic that the card game engine will use at strategic points in the game to decide which cards are legal to play, etc.

```
can_play(Player p, Card c) = c in A/* or c.rank == 8;
```

Actions and Routines

Actions appear in Routines. Actions may move cards from one location to another, show messages to the user, or update properties of players or the game.

Routines will contain control flow, looping constructs, and actions. Routines will be used to advance the state of the game at appropriate times. The card game engine will provide standard actions like deal, display a message, ask for input, and others, and the routines defined in the Pip language will use those actions to control whose turn it is, what message is displayed, and what plays are legal as the game progresses. Routines will not return values.

```
take_turn(Player p) {
  if (p.hand.size < 5) {
    p.hand.add(draw_pile.take(5 - p.hand.size));
  }
  message "Play a card";
  p.get_move();
}
```

Collections

Collections will be used to group name-value pairs together to represent related groups of properties. Collections may describe playing areas, players, and the game itself.

Collections are created to be of a certain kind, and the kind of a collection designates which name-value pairs are valid for the collection. For example, a collection representing a Game will have a name for the game, a list of players, and certain routines and predicates. A collection for a player may also have a name, but it will have different predicates and other properties.

```
a = Area {
  name = "Center";
  show = true;
  stack = false;

  can_play(Player p, Card c) = ...; // Boolean predicate
}

g = Game {
  name = "Euchre";
  players = 4;
  deck = A-9/*;
  areas = [a];
  take_turn(Player p) { ... } // Action Routine
}
```

The name-value pairs inside collections will be checked by the compiler to ensure that they match the kind of collection they appear in.

Goal

The goal is to be able to write a few different styles of card games in the Pip language, compile them, and play them using a textual card game engine. The Pip language should be able to easily express a trick-taking game like Euchre and a shedding game like Crazy Eights to be minimally useful.