HCI Design Methodology Example: From informal Behavioural Descriptions to Implementation of Function. Prof. Roy Eagleson

The following document outlines a case study in which “informal English requirements” are transformed incrementally to more formal representations, resulting in a representation that can be implemented as computer source code.

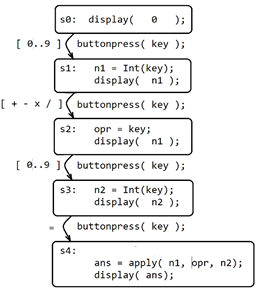
|  |  |
| --- | --- |
| Input | Output |
|  | 0 |
| 5 | 5 |
| + | 5 |
| 3 | 3 |
| = | 8 |

|  |  |
| --- | --- |
| Input | Output |
|  | 0 |
| 7 | 7 |
| - | 7 |
| 1 | 1 |
| = | 6 |

These are all examples of “Usage Scenarios”… Notice, how they are also “Test Cases”.

|  |  |
| --- | --- |
| Event | Action |
| [start] | Display “0” |
| Key=0..9 | Display key |
| Key = + | - | [ display key ] |
| Key = 0..9 | Display key |
| Key is “=” | *What goes here?* |

|  |  |
| --- | --- |
| Event | Action |
| [start] | Display “0” |
| Key=0..9 | Display (num1=key) |
| Key: + - x / | opr = key, [ display num1 ] |
| Key = 0..9 | Display (num2 = key) |
| Key is “=” | Display “apply(num1,opr,num2)” |



Note that this statechart can easily be transformed into source code:

display( 0 );

buttonpress( key );

n1 = Int(key);

display( n1 );

buttonpress( key );

opr = key;

display( n1 );

buttonpress( key );

n2 = Int(key);

display( n2 );

buttonpress( key );

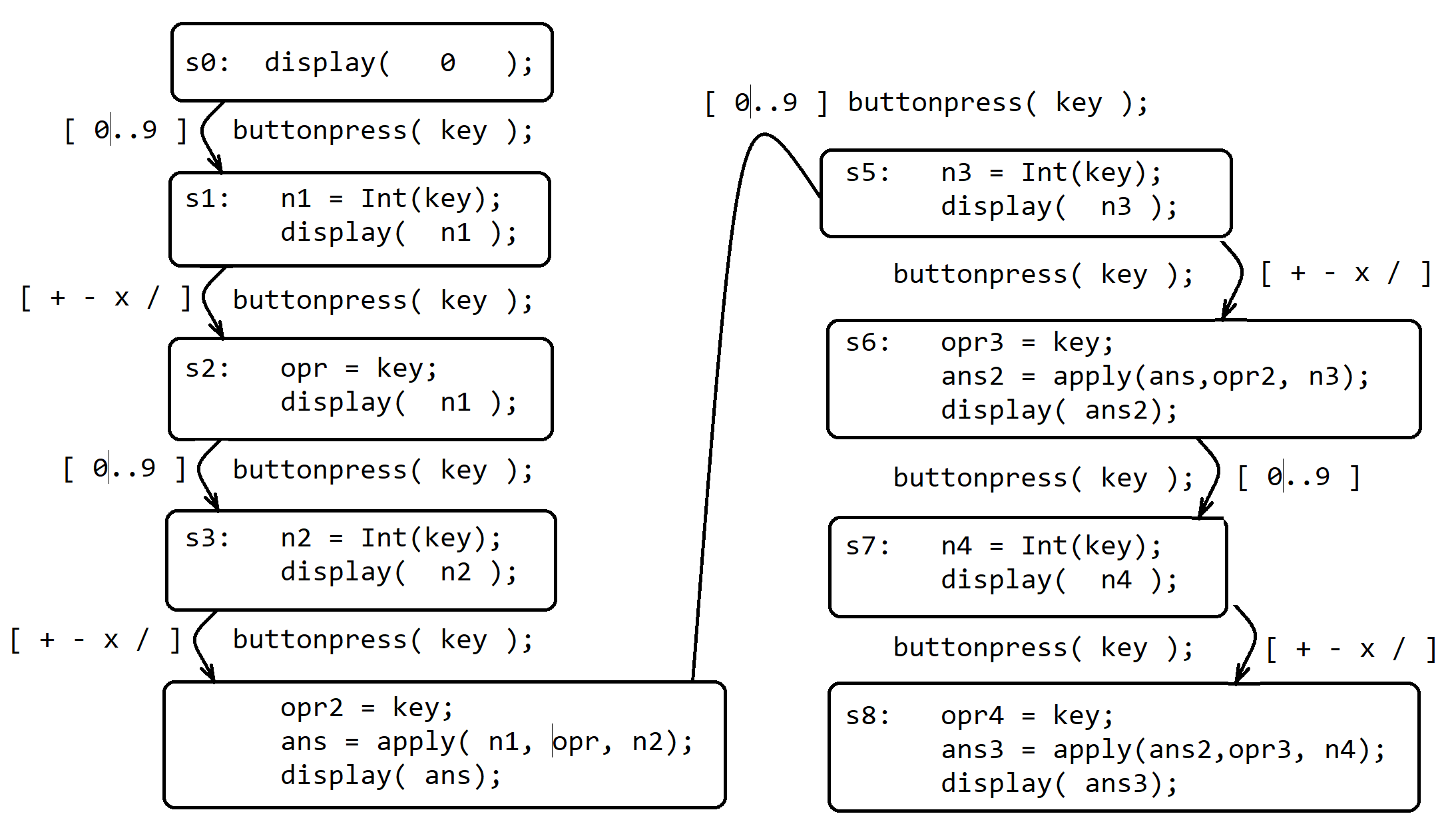
ans = apply( n1, opr, n2);

display( ans);

We can consider a slightly different scenario: This will be different from the previous usage scenario

|  |  |
| --- | --- |
| Input | Output |
|  | 0 |
| 5 | 5 |
| + | 5 |
| 3 | 3 |
| - | 8 |
| 2 | 2 |
| x | 6 |
| 7 | 7 |
| / | 42 |

and so in this case, the ‘=’ is not used, but each new operator reveals the result from the previous operator:



As designers, we notice a pattern emerges and continues for arbitrary sequences of ‘opr’ ‘num’ ‘opr’ ‘num’…. etc. In order to capitalize on this pattern, we strive to convert these specific instances into a generic set of states. We also observe that each time a number is stored in n1, n2, n3, etc, it is read once and then never used again; so we will seek a solution which does not require a sequence of these variables (just one of them might suffice). Accordingly, consider the next state (S9) to involve a number “nk” (instead of n5, which would follow the pattern seen in the diagram). Also, note that there is a “paired” pattern that has emerged, where each pair can be indexed by a digit – a generic ‘k’ variable. In other words, if the states in each pair are indexed by i=2k and j = 2k +1, then the subsequent two states would be “Si”and “Sj” with k=5 (instead of “S10 and S11”). Similarly, if k=4, then “Si and Sj” would correspond to S8 and S9.

So, with these rules for renaming the specific variable names, we now have the following which are generic over i,j, and k: (and if we want to refer to the “previous” pair, then let p=k-1 )

si: oprk = key;

ansp = apply( ans(p-1), oprp, nk );

display( ansp );

sj: nk+1 = Int(key);

display( nk+1 );

We can now examine either of these two states (si and sj), to see if they are functionally equivalent to the other states previously derived. In other words, are any of these two states “Formally Equivalent” to any of the states that we have examined so far? If the answer is yes, then we can collapse those equivalent states, thus minimizing the structure of our implementation.

Of course, computer source code is generally written using ASCII text without subscripts, so we can adapt the names of these variables so that subscripts are replaced by variable names:

si: oprk = key;

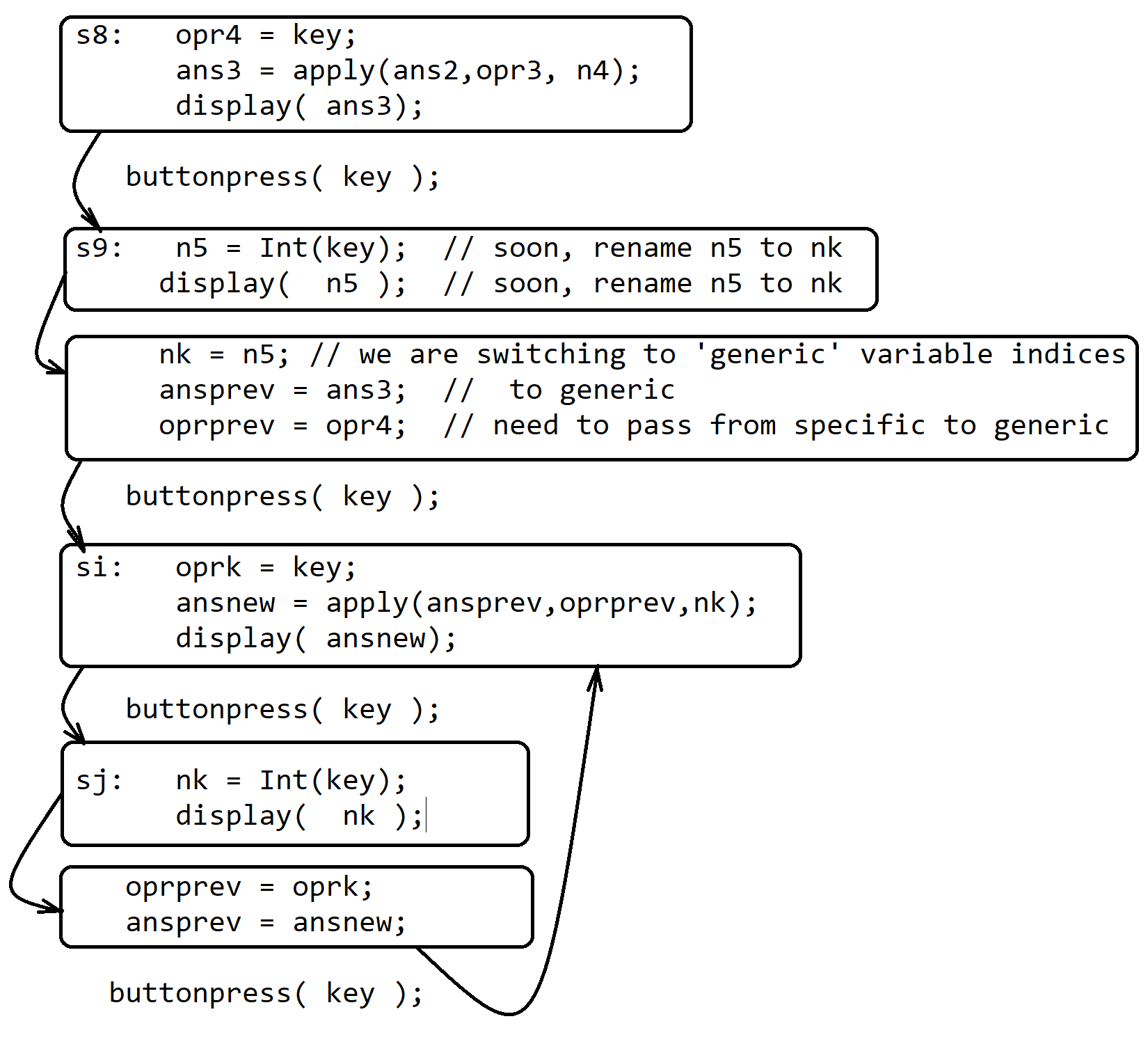
ansnew = apply(ansprev,oprprev,nk);

display( ansnew);

sj: nk = Int(key);

display( nk );

In order to weave these generic states into the existing variables, assignments would need to be made, as show in the following diagram, which makes use of a loop involving them:



An important simplification can be applied here, by observing that “ansprev” appears on the RHS of an assignment, and “ansnew” appears on the LHS of that same assignment. So only one variable is needed. Therefore, we can call that variable “ans”, removing the need for its copy at each iteration of the loop.

Also, the assignment of ‘oprk’ can be done after the other statement(s) in si, since they are not dependent on its value. Remarkably, this means that “oprk” and “oprprev” can make use of one single variable ‘opr’ rather than make use of two copies:

si: ans = apply(ans,opr,nk);

display( ans );

opr = key;

Now, a number of Equivalent States can be recognized as redundant. Note that si is equivalent to S8 and sj is equivalent to S9 when k=4. In fact, sj is equivalent to S1, S3, S5 and S7. And under the renaming of ‘ans’, then si is equivalent to S6 when k=3.

Some care will need to be taken, however, in considering whether si is equivalent to S4, S2, or S0:

s0: display( 0 );

s2: opr = key;

display( n1 );

s4: opr2 = key;

ans = apply( n1, opr, n2);

display( ans);

At first glance, these states do not seem equivalent. In S4, the variables (n1, opr, n2) seem quite different from (ans, opr, nk), however, if we can arrange that n1 is stored in ans when it is read, then all we need to check is whether ans is used before S4. Similarly, the operator can be stored in ‘opr’ rather than opr2. So, without changing its functionality, S4 can be re-implemented as follows:

s4: ans = apply( ans, opr, num );

display( ans );

opr = key;

And for S2, although all that is needed in that state is to display the entered number, there is no reason why it could not be stored in ‘ans’, since at S2 there was an operator entered, but there had been no previous operator.

s2: ans = nk; // only S2 when there is no previous operator pending

display( ans );

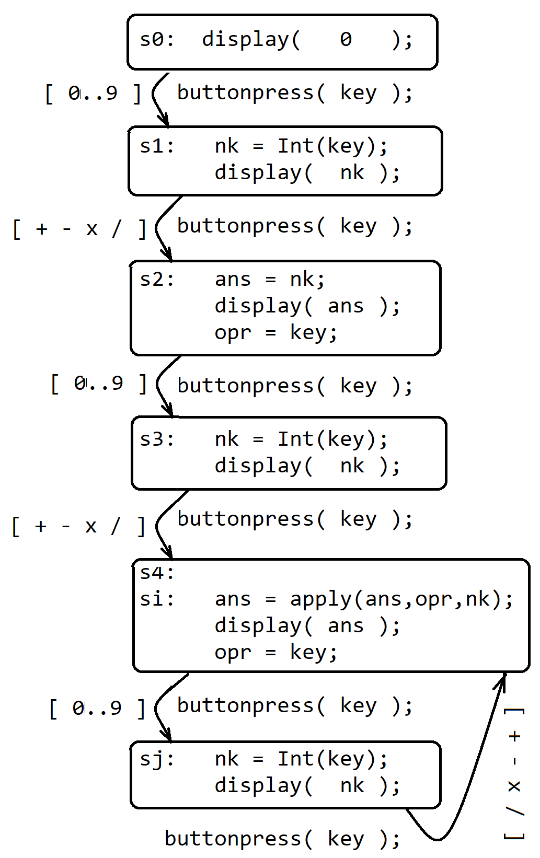
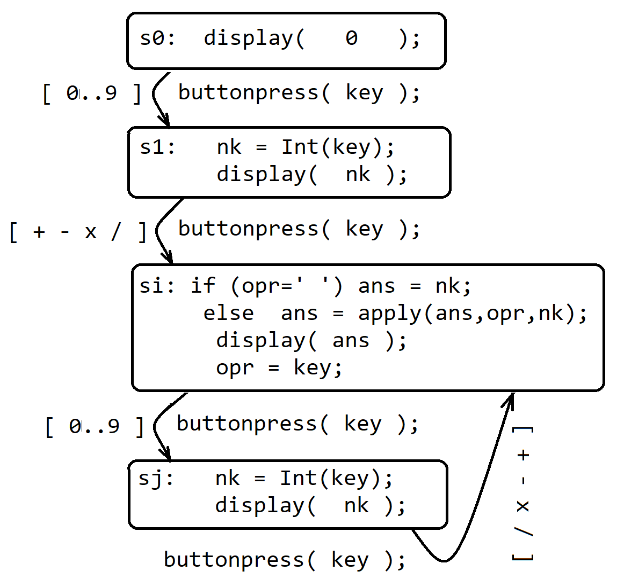
opr = key;

And finally, if we initialize these variables in S0, the following is equivalent to S0:

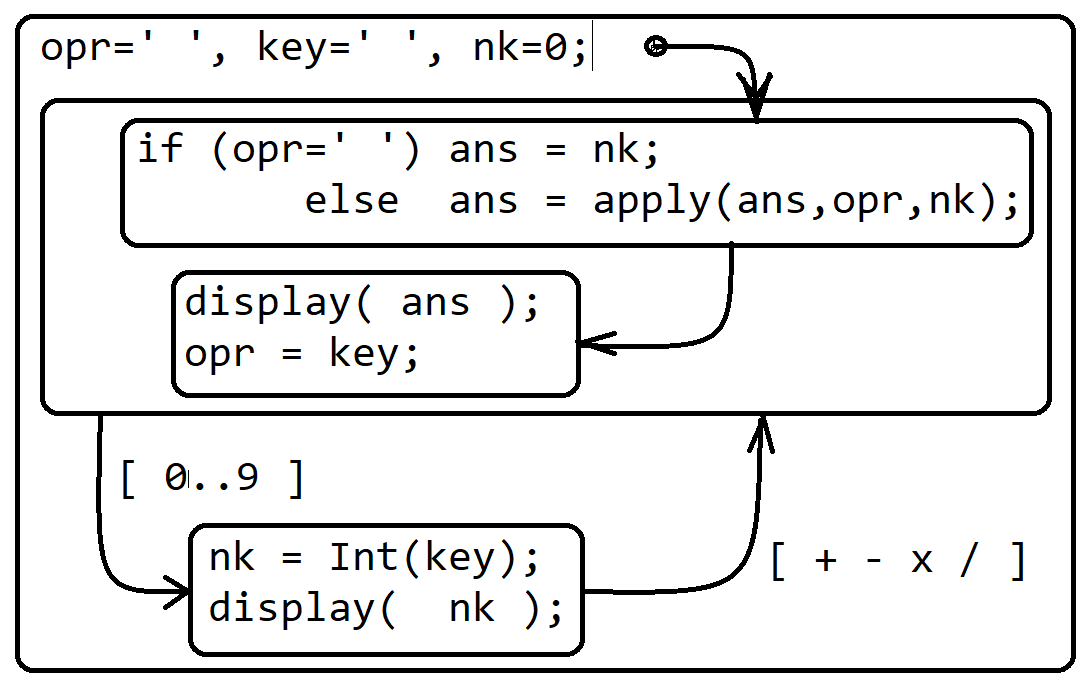
s0: ans = 0;

display( ans );

opr = ‘ ‘;

Three stages of simplification through state minimization, with final representation here as a hierarchical Harel Statechart:



We can now transform this Harel Statechart into a ‘source code’ representation, in a computer language. (note, there was no need to commit to a computer language until this point in the design process.) We implement this state machine in two different languages on the following pages:

// One possible ANSI C implementation of the previous statechart

// can be run to test the system without even needing a GUI

// All that is needed here is the stdin keypresses getchar and printf

#include <stdio.h>

#include <stdlib.h>

#define Int(a) a-'0';

#define display(a) printf("\r %d ", a );

#define buttonpress( a ) a = getchar();

int apply(int a, char b, int c) {

if(b == '+') return a+c;

if(b == '-') return a-c;

if(b == '\*') return a\*c;

if(b == '/') return a/c; else return(c);

}

char key,opr;

int nk, ans;

main() { system("stty raw -echo");

opr=' ', key=' ', nk=0;

si: if (opr==' ') ans = nk;

else ans=apply(ans,opr,nk);

display( ans );

opr = key;

buttonpress( key ); if(key=='q') goto finish;

nk = Int(key);

display( nk );

buttonpress( key ); if(key=='q') goto finish;

goto si;

finish: system("stty -raw echo");

exit(0);

}

// on the following page, we can use Java and the AWT to creat a GUI version

import java.awt.\*; import java.awt.event.\*;

public class calc00 extends Frame {

TextField calcTextField;

Button[] buttons = new Button[16];

char[] buttonText={'9','8','7','X','4','5',

'6','/','1','2','3','-','0',' ',' ','+'};

int ans=0, nk=0;

char opr=' ', key=' ';

public calc00() {

setLayout(new FlowLayout());

calcTextField = new TextField(5);

add(calcTextField);

for (int i=0; i<16; i++) {

buttons[i] = new Button (""+buttonText[i]);

add(buttons[i]);

}

}

public int apply (int ans, char opr, int nk) {

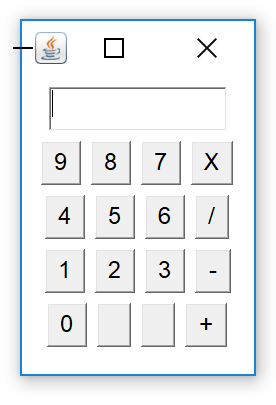
switch (opr) {

case '+': return ans += nk;

case '-': return ans -= nk;

case 'X': return ans \*= nk;

case '/': return ans /= nk;

 } return -1;

}

public boolean action(Event ce, Object co) {

key = ((String)co).charAt(0);

if((key>='0')&&(key<='9')) {

nk=(key-'0');

calcTextField.setText(""+nk);

} else {

if (opr==' ') ans = nk;

else ans = apply (ans,opr,nk);

calcTextField.setText(""+ans);

opr = key;

}

return true;

}

public static void main(String[] s){ new calc00().setVisible(true); }

}

When compiled, executed, and the window resized, we see this display:

On the following page, we can implement the same functionality using HTML and Javascript:

<FORM name="mycalc">

<input name="mydisp" type="Text" size=10 value="0" > <br>

<input type="Button" value="7" onclick="digitKey(7)">

<input type="Button" value="8" onclick="digitKey(8)">

<input type="Button" value="9" onclick="digitKey(9)">

<input type="Button" value="X" onclick="oprKey('X')"> <br>

<input type="Button" value="4" onclick="digitKey(4)">

<input type="Button" value="5" onclick="digitKey(5)">

<input type="Button" value="6" onclick="digitKey(6)">

<input type="Button" value="/" onclick="oprKey('/')"> <br>

<input type="Button" value="1" onclick="digitKey(1)">

<input type="Button" value="2" onclick="digitKey(2)">

<input type="Button" value="3" onclick="digitKey(3)">

<input type="Button" value="-" onclick="oprKey('-')"> <br>

<input type="Button" value="0" onclick="digitKey(0)">

<input type="Button" value=" " >

<input type="Button" value=" " >

<input type="Button" value="+" onclick="oprKey('+')">

</FORM>

<script>

function display(a) { document.forms['mycalc']['mydisp'].value=a; }

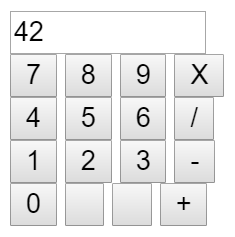
var opr = ' ';

function apply (a,op,n) {

switch (op) {

case "+": return a += n;

case "-": return a -= n;

 case "X": return a \*= n;

case "/": return a /= n;

}

}

function digitKey (key) {

num = key;

display(num);

}

function oprKey (key) {

if(opr==' ') ans = num;

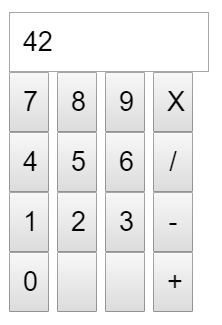
else ans = apply(ans,opr,num);

display(ans);

opr = key;

}

</script>

And if we add the following CSS declarations to the HTML file, we can change the visual layout:

<style>

input {

height: 30px;

width: 20px;

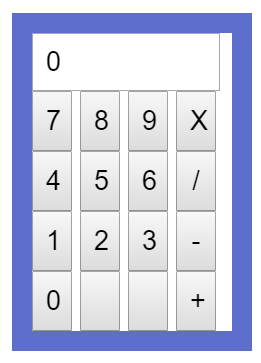
padding: 5px 5px 5px 5px;

box-sizing: border-box;

}

input[type=text] { width: 100px; }

</style>

<style>

input {

height: 30px;

width: 20px;

padding: 5px 5px 5px 5px;

}

input[type=text] {

height: 15px; width: 80px; }

form {

width: 100px;

border: solid #5B6DCD 10px; }

</style>

// This can also be implemented for Android:

package com.royeagleson;

import android.app.Activity;

import android.os.Bundle;

import android.widget.\*;

import android.view.\*;

public class demo extends Activity {

char key,opr;

float ans,num;

char[][] btext={{'7','8','9','\*'},{'4','5','6','/'},

{'1','2','3','-'},{'0',' ',' ','+'}};

Button[][] buttons = new Button[btext.length][btext[0].length];

static TextView calcText;

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

LinearLayout myPanel = new LinearLayout(this);

myPanel.setOrientation(LinearLayout.VERTICAL);

setContentView(myPanel);

TextView iniText = new TextView(this);

myPanel.addView(iniText);

iniText.setText("Hello, World!");

calcText = iniText;

LinearLayout[] rowPanel = new LinearLayout[btext.length];

for(int j=0; j<btext.length; j++) {

rowPanel[j] = new LinearLayout(this);

rowPanel[j].setOrientation(LinearLayout.HORIZONTAL);

for (int i=0; i<btext[j].length; i++) {

buttons[j][i] = new Button(this);

buttons[j][i].setText(""+btext[j][i]);

rowPanel[j].addView(buttons[j][i]);

if( (btext[j][i]-'0') >= 0 && (btext[j][i]-'0') <= 9 )

buttons[j][i].setOnClickListener( new View.OnClickListener() {

public void onClick(View view) { myNumAction(view); }} );

else

buttons[j][i].setOnClickListener( new View.OnClickListener() {

public void onClick(View view) { myOprAction(view); }} );

}

myPanel.addView(rowPanel[j]);

}}

public float apply(float num1, char opr, float num2) {

switch (opr) {

case '+': return(ans+num);

case '-': return(ans-num);

case '\*': return(ans\*num);

case '/': return(ans/num);

default: return(num);

}}

public void myOprAction(View view) {

ans = apply( ans, opr, num);

calcText.setText(""+ans);

opr = ((Button)view).getText().charAt(0);

}

public void myNumAction(View view) {

num=(float)(((Button)view).getText().charAt(0)-'0');

calcText.setText(""+num);

}}

