Cooperating Languages - First Example

Hesam Samimi

This material is based upon work supported in part by the National Science Foundation under Grant No. 0639876. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

VPRI Memo M-2012-006

Viewpoints Research Institute, 1209 Grand Central Avenue, Glendale, CA 91201 t: (818) 332-3001 f: (818) 244-9761

Cooperating Languages
First Example

Hesam Samimi

August 31, 2012

The goal of Cooperating Languages framework is to integrate a host of languages and constraint solvers within one environment, so that different computing tasks can be performed in an elegant, domain specific manner, all on top of a single program model.

As a first demo, I adopted and extended the moving boxes demo that I think Aran worked out a while ago to demonstrate the use of Cassowary solver for laying out GUI components.

Moving Boxes Demo

As shown in Fig. 1, there are several GUI boxes on a canvas that can be moved and resized by the user. The example is run by evaluating the following.

```
CoopLangMovingBoxesDemo new openInWorld.
```

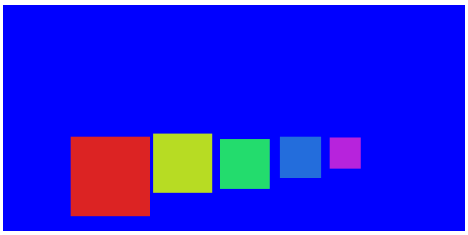

Fig. 1. Moving boxes

Program Model

We start by designing a model to represent the objects involved in the program, as seen in Fig. 2. Since boxes can be nested, other than position and size information, each box stores its set of children boxes. Variables `b1,...,b5` refer to the five boxes, while `p1,...,p5` keep their top-left corner points.

Class definitions implictly generate corresponding relations that store instances (e.g. `Box(a: Box)`) and attributes (e.g. `Point_x(p: Point, x: Real)`, `Box_contents(b: Box, a: Box)`).

```
class Point : Thing
    Attributes
        x, y : Real
class Box : Thing
    Attributes
        corner : Point
        width : Real
        contents : Set ( Box )
var b1, b2, b3, b4, b5 : Box
var p1, p2, p3, p4, p5 : Point
var padding : Real
```

Fig. 2. Moving boxes model

Task #1: Mouse Events

User mouse events are implemented using the listener pattern, so the appropriate language for this task is Squeak. I copied this code from the original moving boxes demo.

Task #2: Layout Constraints

The moving boxes have several constraints. They cannot be moved outside margins or overlap each other. More importantly, the x-coordinate layout ordering of the boxes needs to be preserved. There are soft *stay* constraints that express the inertia of these boxes. Thus when the user drags a box, other boxes may have to be pushed along with it, as illustrated in Fig. 3.

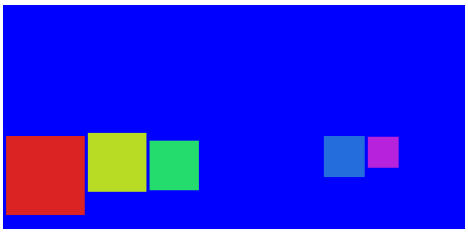

Fig. 3. Box layout constraints forcing boxes to be pushed along

Cassowary is ideal for handling these linear hard and soft constraints, which are stated in Fig. 4[1]. In fact, we use its objective version called *OCassowary* to directly express the constraints over the object attributes.

[1] The `padding` variable defines the margins that limit how close to the edges of the canvas and the neighboring box can these boxes be moved to.

```
env hostProgram:
    p1.x, p2.x, p3.x, p4.x, p5.x stay
    p1.x >= padding
    p5.x <= 600 - b5.width - padding
    p1.x + b1.width + padding <= p2.x
    p2.x + b2.width + padding <= p3.x
    p3.x + b3.width + padding <= p4.x
    p4.x + b4.width + padding <= p5.x
in:
    #0Cassowary
modifiableAttributes:
    #((Point x))
```

Fig. 4. Moving boxes layout constraints

Task #3: Which Boxes Move Along?

Fig. 5 demonstrates how the user can temporary disable the layout rules above to drag boxes inside each other in order to build composite GUI components.

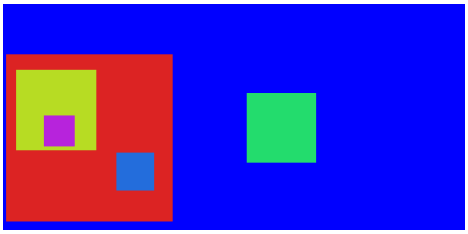

Fig. 5. Nested boxes

When a box is dragged by the user, we would like all the contained boxes to move along as one composite GUI. For example in Fig. 5 if the yellow box is moved, then the purple is moved along but the parent red box is not (Fig. 6). On the other hand, when we moved the red box, the yellow, purple, and blue boxes all were moved along (Fig. 7).

Since each box records only its child boxes, the task of computing what boxes are dragged is essentially a transitive closure operation. This computation is concisely expressed in Datalog, as seen in Fig. 8. The extensional `DragBox` relation indicates the box currently dragged by the user, while the intensional relation `DragAlong` computes the desired set.

Conclusion

The hope is that as we work out more examples, we can get more insights as to how to evolve the cooperating solvers project into the cooperating languages framework.

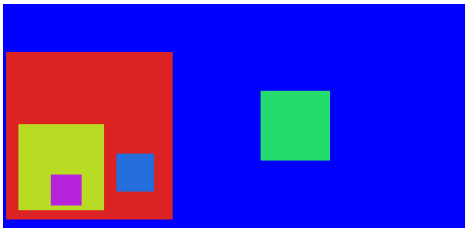

Fig. 6. Moving nested boxes. Parent box not moved.

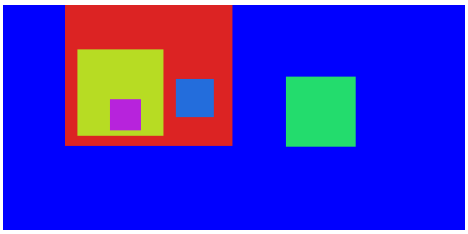

Fig. 7. Moving nested boxes. All contained boxes moved.

```
relation DragBox(a: Box)
relation DragAlong(a: Box) <--
    DragBox(a) ||
    exists b : Box | (Box_contents(b, a) && DragAlong(b))
```

Fig. 8. Moving Rules