`maybe` We Should Enable More Uncertain Mobile App Programming

Yihong Chen, Sriram Shantharam, Guru Prasad Srinivasa, Jinghao Shi, Jerry Ajay, Ali Ben Ali, Lukasz Ziarek, Oliver Kennedy, Geoffrey Challen

blue.cse.buffalo.edu

Problem: Forced Certainty in the Face of Uncertainty

```
if (batteryLevel < threshold) {
  /* Save energy */
} else {
  /* Don't save energy */
}
```

Uncertainty is inherent to mobile systems programming. But today's languages require programmers to be certain at development time about uncertain runtime conditions, forcing them to attempt brittle ad-hoc adaptations.

Consider the simple attempt to adapt to available energy shown above. Even in this example, several things can go wrong, caused by differences between users, devices, networks, and other exogenous conditions.

Solution: Structured Uncertainty Using maybe

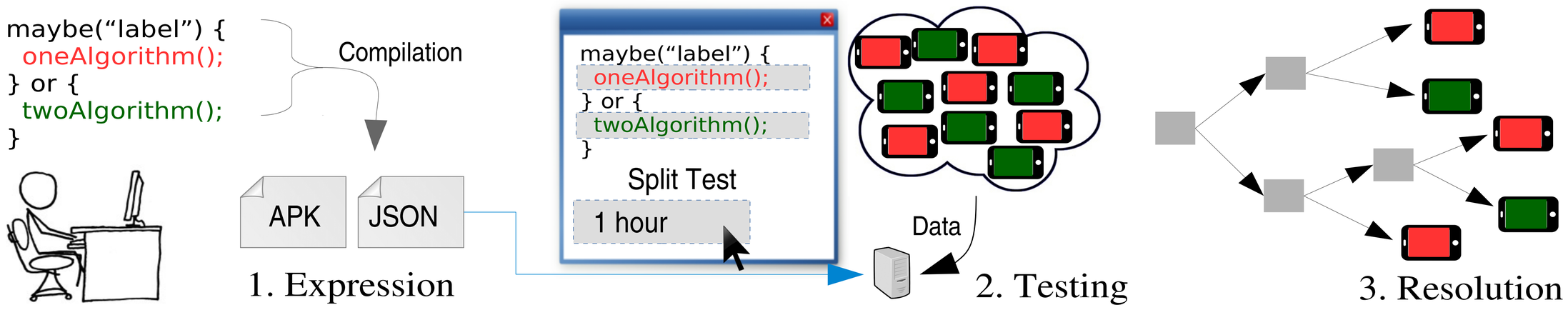
```
int interval = maybe ("interval") 10, 60, 120;

String option = maybe ("level") "low", "high";

maybe ("battery_level") {
  /* Save energy one way */
} or {
  /* Don't save energy */
}
```

We propose a simple solution: allow programmers to express *structured uncertainty* using a new programming construct, the **`maybe`** statement. Structured uncertainty allows programmers to provide multiple options with different behaviors and let the system choose between them later using a variety of different techniques.

`maybe` System Overview

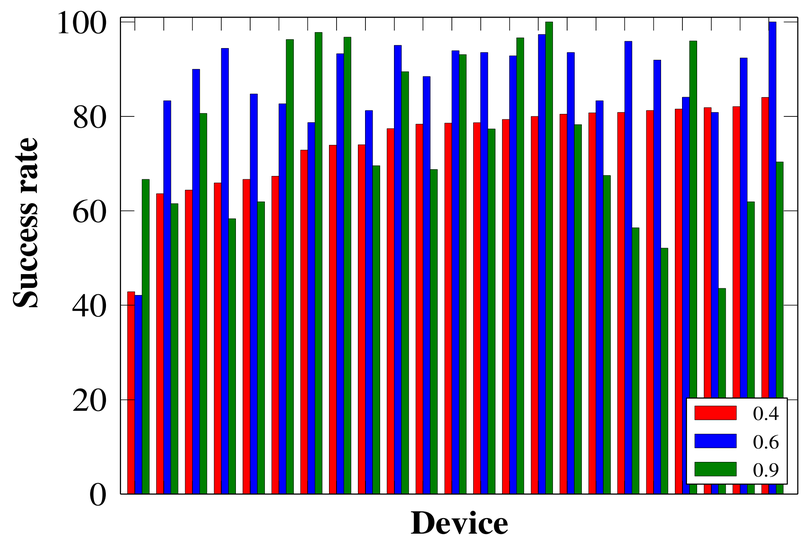


Post-deployment, large scale split testing is used to gather data. Machine learning techniques are used to decide which alternative gets chosen.

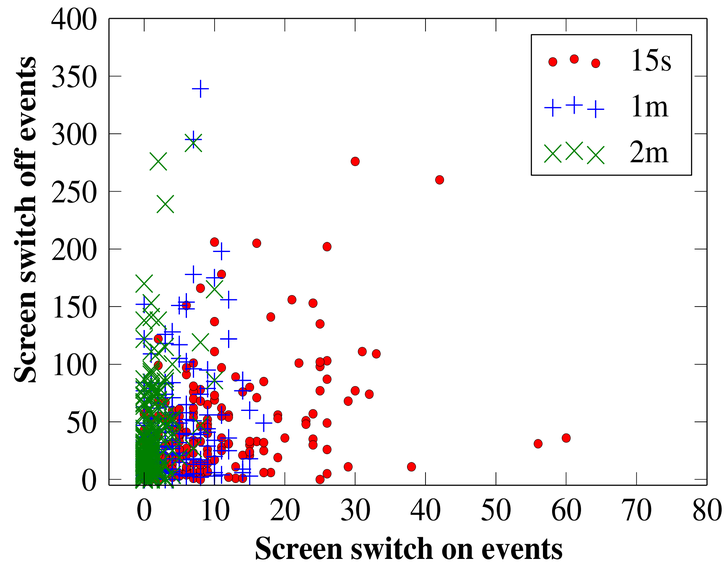
Case Studies

In the first case, we used **`maybe`** to experiment with three different keyguard pattern sensitivities. To score these alternatives we recorded the rate of incorrect unlock attempts. Our adaptation goal was to choose the smallest target that produced a reasonable success rate to avoid frustrating users. The target value of 0.9 corresponds to the most sensitive setting which might be problematic for users with larger fingers. The target size that achieved a 80% unlock success rate was small for 2.2% of users, medium for 41.3% of users, and large for 54.3% of users. 2.2% of users did not achieve 80% success rate at any threshold.

In the second experiment, we used three different timeouts, 15 s, 1 m and 2 m. To score these alternatives, we used score to record both the number of spurious screen locks by identifying unlock events that followed rapidly (possibly indicating a threshold that is too small), and the number of unnecessary manual locks (possibly indicating a threshold that is too large). We weighted both of these negative events equally, and the ideal score for a particular timeout is zero: indicating that it is as close a fit as possible to perfectly predicting the moment when the user is done using their device. 1 m performed best for 34% of users, 2 m performed best for 52% of users while 15 s performed best for 14% of users



Per-user bars are shown comparing success rates under different keyguard pattern sensitivities for 23 participants. The value of 0.9 corresponds to the most sensitive setting which might be problematic for users with larger fingers.



A scatter plot of spurious lock events v. unnecessary lock events is shown. Ideal values are close to the origin.

Future work

- Compiler support
- Integration with existing pre-deployment testing frameworks
- *A Priori* Decisions - Use the past to predict the future
- Automatic Settings Annotation - Let the users decide

This Is A Message of Hope

`maybe` gives programmers more freedom and leads to less defensive development and accelerated adaptation process.

```
maybe { /* You liked our poster */
  email("maybe@blue.cse.buffalo.edu");
} or {int i = *NULL;}
```