A Modal Language for the Safety of Mobile Values

Sungwoo Park

Pohang University of Science and Technology, Republic of Korea
gla@postech.ac.kr

Abstract. In the context of distributed computations, local resources give rise to an issue not found in stand-alone computations: the safety of mobile code. One approach to the safety of mobile code is to build a modal type system with the modality $\Box$ that corresponds to necessity of modal logic. We argue that the modality $\Box$ is not expressive enough for safe communications in distributed computations, in particular for the safety of mobile values. We present a modal language which focuses on the safety of mobile values rather than the safety of mobile code. The safety of mobile values is achieved with a new modality $\boxdot$ which expresses that given code evaluates to a mobile value. We demonstrate the use of the modality $\boxdot$ with a communication construct for remote procedure calls.

1 Introduction

A distributed computation is a cooperative process taking place in a network of nodes. Each node is capable of performing a stand-alone computation and also communicating with other nodes to distribute and collect code and data. Thus a distributed computation has the potential to make productive use of all the nodes in the network simultaneously.

Usually a distributed computation assumes a heterogeneous group of nodes with different *local resources*. A local resource can be either a permanent/physical object available at a particular node (*e.g.*, printer, database) or an ephemeral/semantic object created during a stand-alone computation (*e.g.*, heap cell, abstract data type). Local resources are accessed via their references (*e.g.*, handle for a database file, pointer to a heap cell).

Local resources, however, give rise to an issue not found in stand-alone computations: the safety of *mobile code*, or in our terminology, the safety of *mobile terms* where a term represents a piece of code. In essence, a node cannot access remote resources in the same way that it accesses its own local resources, but it may receive mobile terms in which references to remote resources are exposed. Therefore the safety of mobile terms is achieved by supporting direct access to remote resources (*e.g.*, remote file access, remote memory access), as in Obliq [1], by transmitting copies of local resources along with mobile terms, as in Facile [2], by preventing references to remote resources from being dereferenced, as in Mobile UNITY [3], or by allowing all of these methods, as in λdist [4]. Our paper focuses on the third case where we reject mobile terms containing references to remote resources.

One approach to the safety of mobile terms is to build a modal type system with the modality $\Box$ [5,6,7,8] which is based on a spatial interpretation of necessity of modal logic such as S4 and S5. The basic idea is that a value of modal type $\Box A$ contains a

N. Kobayashi (Ed.): APLAS 2006, LNCS 4279, pp. 217–233, 2006.

mobile term that can be evaluated at any node. By requiring that a mobile term be from a value of type $\Box A$, we ensure its safety without recourse to runtime checks.

A type system augmented with the modality $\Box$ is not, however, expressive enough for safe communications of *values*, *i.e.*, the safety of *mobile values*. In other words, we cannot rely solely on modal types $\Box A$ to verify that a value communicated from one node to another is mobile (*e.g.*, when a remote procedure call returns, or when a value is written to a channel). The reason is that in general, a value of type $\Box A$ contains *not a mobile value but a mobile term*. The evaluation of such a mobile term (with the intention of obtaining a mobile value) may result in a value that is not necessarily mobile because of references to local resources created during the evaluation.

As an example, consider a term of type `int -> int` in an ML-like language:

```
let
  val new_reference = ref 0
  val f = fn x => x + !new_reference
in
  f
end
```

The above term can be evaluated at any node and thus may be used in building a mobile term of type `□(int -> int)`. The resultant value `f`, however, is not mobile because it accesses a local resource `new_reference`. In contrast, the following term, also of type `int -> int`, cannot be used in building a mobile term of type `□(int -> int)`, but the resultant value is mobile because it does not access any local resource:

```
let
  val v = !some_existing_reference
  val f = fn x => x + v
in
  f
end
```

Hence the modality $\Box$ is irrelevant to the safety of mobile values, which should now be verified by programmers themselves.

This paper investigates a new modality $\boxdot$ which expresses that a given term evaluates to a mobile value. The basic idea is that a term contained in a value of modal type $\boxdot A$ evaluates to a value that is valid at any node. For example, the first term above cannot be used in building a term of type `⊡(int -> int)`, but the second term above may be used in building such a term. To obtain a value to be communicated to other nodes, we evaluate a term contained in a value of type $\boxdot A$. In this way, we achieve the safety of mobile values.

While the mobility of a term is independent of the mobility of the value to which it evaluates, the modality $\Box$ is weaker than the modality $\boxdot$ in that we can emulate $\Box$ with $\boxdot$. For example, we may define $\Box A$ as `⊡(unit -> A)`, in which case we check the mobility of a term M of type A by checking the mobility of a value `fn _ =>` M of type `unit -> A`. Thus $\boxdot$ is inherently more expressive than $\Box$, and the use of $\boxdot$ practically eliminates the need for $\Box$. The converse is not the case, however: we cannot