Linear typing for asset-aware programming: the case of Sui Move

Oral Communication
DLT 2024: 6th Distributed Ledger Technology Workshop

Michele Bugliesi[1] Silvia Crafa[2] Giacomo Dal Sasso[2] Sabina Rossi[1]
Alvise Spanó[1]
[1]Università Ca' Forscari [2]Università di Padova

Smart contracts running on blockchain platforms are critical components of decentralized applications that manipulate valuable assets; their errors can lead to huge capital losses. The Move language, born with the Diem blockchain and later inherited by the Aptos and Sui blockchains, seems to be a step forward in terms of safe asset management in smart contracts. As the main novelty, Move introduces the use of linear types to represent resources (like financial assets) in the language, in a way that allows compile-time detection of common errors in the manipulation of resources. More precisely, assets like cryptocurrencies and NFTs are represented in Move as C-like structs, and the type system guarantees that instances of these structs are *not duplicated or lost* during the execution of the smart contract.

As an example, let `Nft` be the type of an asset, and consider an application for trading Nfts. Clearly, when a user purchases a `Nft`, this asset should be guaranteed to be unique and it should be impossible to accidentally lose it. In Solidity, given a value of type `Nft` it is possible to create any number of copies of the same value, for instance, the following code mistakenly assigns two copies of the same asset to the same user:

```
function buyNft() public payable { //...
  Nft memory nft = Nft(counter, msg.value);
  ownedNfts[msg.sender].push(nft);
  ownedNfts[msg.sender].push(nft);
  //...
}
```

While the error is obvious here, the duplicated erroneous assignment could be hidden within more complex code, e.g. within a nested function call, and the point is that no compile-time error is raised. Instead, this kind of error can't happen in Move because the type system prevents the copy of a value of an asset type like `Nft`, provided that it has been defined as a *linear type* (i.e. a structure which has not the *drop* and *copy* abilities).

Similarly, in Solidity, it is possible to silently and accidentally lose a value (an asset) of type `Nft`, thus causing a loss for its owner. Assume, for example, in the `buyNft` function the programmer wants to increment the counter by 1 when the counter is less than 10 and by 2 otherwise. But by doing this, he forgets to store the newly created `Nft` resource somewhere, when the counter is less than 10:

```
function buyNft() public payable { //...
  Nft memory nft = Nft(counter, msg.value);
  if (counter < 10) {    counter ++;  }
     else {     counter += 2;  ownedNfts[msg.sender].push(nft);  }
}
```

The asset in the `nft` variable is lost when the counter is less than 10 and `nft` goes out of scope. As before the error is obvious here, but depending on the complexity of the function's logic it may

not be so. In Move the corresponding code would not compile since the compiler would detect that at the end of the if-branch the value of the `nft` variable is dropped without being consumed.

As a consequence in Move a value of an asset type can't be accidentally lost; it can only be explicitly unpacked, published on the blockchain's global store, or passed to another function, that in turn must do one of these three things with the received value.

Finally, notice that in Solidity an asset can also be lost when a variable pointing to the asset is overwritten like in the following example:

```
Nft memory n1 = Nft(1, 0);
Nft memory n2 = Nft(4, 0);
n1 = n2; // <- n1 is overwritten and the asset Nft(1, 0) is lost
```

Even if this is a simple bug, it often goes unnoticed. Then having a compiler that statically spots it is valuable support for safe smart contract programming.

In this oral communication we report the ongoing work [1] that formally studies the Sui Move language [2], its linear type system and its resource safety properties. More precisely, the contributions of this work can be summarized as follows:

- We formalize the operational semantics and the type system of the functional core of the Sui Move language, and we prove a Resource Preservation theorem stating that in a well-typed program resources (assets in particular) can't be duplicated or accidentally lost at runtime. This theorem lifts to the source code an equivalent theorem proved by Blackshear et al. [3] for (the formalization of) the Move bytecode.

- While proving safety for the bytecode is more general, providing static tools for the safety of the source code better supports the reasoning of programmers, and promotes an asset-aware programming style. In particular, the formalization of Move high-level language gives a better understanding of what it means to use a variable and a value (in particular a linear value), diving into the *move semantics* of Move (inspired by Rust) and clarifying the mechanism underlying the linear types in the language.

- The formalization also helps to clarify what are the kinds of errors prevented in Move by means of the linear discipline, disentangling them from those prevented by the mechanism of ownership/access control, which in Move is based on the controlled use of mutable references. This consents to understanding the potential of linear types in the context of other smart contracts programming languages.

- Finally, the entire formalization has been mechanized within the Agda proof assistant. In particular, typed and non-typed programs can be checked by Agda, and the proofs of the two main theorems -type preservation and resource preservation- have been fully mechanized. The code in Agda is not only a formal proof of the correctness of the type system and the operational semantics of the Move core but is also a framework for future extensions of the language and for the investigation of new language properties.

Acknowledgements.

This work has been partially supported by the Project PRIN 2020 "Nirvana - Noninterference and Reversibility Analysis in Private Blockchains", and by the project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU.

[**1**] Giacomo Dal Sasso. Linear typing for resource-aware programming. Master Thesis. Università di Padova, Feb. 2024. https://hdl.handle.net/20.500.12608/62007

[**2**] Adam Welc, Sam Blackshear. Sui Move: Modern Blockchain Programming with Objects. SPLASH 2023: Companion Proceedings of the 2023 ACM SIGPLAN International Conference on Systems, Programming, Languages, and Applications: Software for Humanity. October 2023. Pages 53–55. https://doi.org/10.1145/3618305.3623605

[**3**] Sam Blackshear, David L Dill, Shaz Qadeer, Clark W Barrett, John C Mitchell, Oded Padon, and Yoni Zohar. Resources: A safe language abstraction for money. arXiv preprint arXiv:2004.05106, 2020.