Symbolic Distinction

by Sven Nilsen, 2021

In this paper I introduce a special operator for symbolic distinction.

A symbolic distinction is a special partial[1] operator of type `bool`:

```
sd(a, b) : bool
```

This special operator is interpreted as a proof of symbolic distinction,
regardless of how the values are otherwise treated as equal or inequal.

The reason symbolic distinction is partial is that it does not terminate for symbolic equal inputs:

```
sd(a, a) : !                    `!` means the program does not terminate
```

In many theories this means symbolic distinction does not terminate for syntatically equal[2] inputs.

Hence, symbolic distinction can only prove two symbols to be distinct.
It can not prove two symbols to be symbolic equal.

The motivation for this can be seen in the table below:

	Symbolic indistinct	**Symbolic unknown**	**Symbolic distinct**
Equal	ok	ok	ok
Unknown	ok	ok	ok
Unequal	err	ok	ok

When two objects are symbolic indistinct but unequal, it leads to soundness[3] problems.

By restricting ourselves to only talking about the symbolic unknown and symbolic distinct,
one avoids the case leading to soundness problems, at the cost of eliminating two cases.

It is tempting to define symbolic distinction as as syntactically inequality:

```
sd <=> not . syn_eq
```

The problem is that syntactic equality is treated differently in various theories.
If syntactic equality is too strong in one theory, then its opposite becomes too weak.
Therefore, I will not define precisely what it means, but instead provide axioms of its behaviour.

Since symbolic distinction is partial, this leads to more complex theorem proving.
To avoid more complexity than necessary, symbolic distinction should be used with moderation.
The idea is to design languages where users can ignore the complexity behind the language.

Termination Assumption

A simplifying assumption of termination is often used to split proofs into cases.
Under this assumption, it is possible to reason partially and locally.
In lamba expressions and for-all loops `∀`, it suffices that the parameters are different:

\(a, b) = …	sd(a, b)	`a` and `b` are different parameters
∀ a, b { … }	sd(a, b)	`a` and `b` are different paremeters

For example, under the termination assumption, the following lambda expressions are equivalent:

\(a, b) = ¬sd(a, b)	<=>	\(a, b) = false

Substitution Under Equality

Even without the simplifying assumption of termination for lamba expressions,
one can prove some things using the simplifying assumption of termination on a higher level.

Substitution[4] under equality is sound using the following assumption:

∀ a, b, c, f { ((b == c) ∧ sd(a, c)) => (f(a, b) == f(a, c)) }

Since `f` might depend on `sd(a, b)`, one must prove that `c` is distinct from `a` to substitute for `b`.
This means `sd(a, c)` returns `true` whenver `sd(a, b)` returns `true`.

However, one can not prove the following:

∀ a, b, f { ((b == a) ∧ sd(a, a)) => (f(a, b) == f(a, a)) }

This is because the proof does not terminate due to `sd(a, a) : !`.
It might seem confusing because one usually expects any proof to hold by substitution.

When `f` does not depend on symbolic distinction, one can prove the following:

∵ ∀ a, b, f { (b == a) => (f(a, b) == f(a, a)) }
∴ true

Substitution under equality holds in the two cases above and is unknown otherwise.

Notice the complexity that arises from needing multiple proofs.
Each proof says something about the dependence/independence of symbolic distinction.

References:

[1] “Partial function”
Wikipedia
https://en.wikipedia.org/wiki/Partial_function

[2] “Equality, specifications and implementations”
The Xena Project
https://xenaproject.wordpress.com/2020/07/03/equality-specifications-and-implementations/

[3] “Soundness”
Wikipedia
https://en.wikipedia.org/wiki/Soundness

[4] “Substitution (logic)”
Wikipedia
https://en.wikipedia.org/wiki/Substitution_%28logic%29