Notes - 4/6

Friday, April 06, 2012 10:56 AM

[In 41/4725 at 11:00 today.]

Agenda:

- Feedback from Mankala app building
  - Cross-declaration module privates
  - Options:
    - Textual include: no - havoc on IDE
    - Module merging at compile time
      - Issues with reordering
      - Only way to accomplish multiple files + encapsulation
      - **Possible best solution here.**
    - What we have now
    - Modules are non-extensible
    - "Private"
- Feedback from Backbone app building
  - Forwarding constructors
    - No for now, though we see the value
  - Base call ordering
  - Prototype properties
  - Property shadowing and global name access
  - Parens on classes?
  - Classes
    - Track potential limtiations of current class modle
      - Understand whether they can be absorbed into class model in future
    - Future proof
  - No magic global thing
- =>
  - Embrace ES6 => proposal
  - Kill self
  - Kill this parameters
  - Question: Can we spec void returning 'function' as getting a construct signature, la
    even if they return void

We agreed a few meetings back to loosen the typing rules for && and || given the useful variety of usage these operators see in realworld JavaScript.

Below is the proposal. It captures these rules:

1) A && B has the type of B
2) A || B has the type of A
3) But, if either A or B is any, the result is any

These are motivated by the notion that there are a lot more “truthy” values in JavaScript than “falsy” values. So the most common case will be two truthy inputs, and the result type should be “correct” in that case. This maps to the first arg for || and the second arg for &&. If the type of the result is truly guaranteed to be one of the input types (which is actually very common in practice), this rule will be “correct”. However, in usages where the operator is not well typed – like “3 && ‘hello’” – we will not report an error, and will state that the type of the result is the more common subset of the true (unrepresentable in Strada) union type.

For example – common cases like these – we will type correctly:

```
var obj = {prop: 3}
var x = obj && obj.prop
var isFoo = true, isBar = false, str1 = "a",
str2 = "b", str3 = "c"
var x = isFoo && str1 || isBar && str2 ||
str3
```

Rule 3 above is possibly up for debate – but is consistent with other operator typing.

mbdas do not get this

even if they return void

- "Return" + "return 5" should be legal
  - Loosen

- && and || typing
  - || should stay almost as is, but use best common type
  - && should be as in the new proposal, except 'any' is not infectious. AKA forget the
  - Question: What about valueOf and toString?
- Review all remaining “Open” issues on the [design items spreadsheet]
- Others?

Class

}

Class

}

Functi
Bar.
}

```
module A {

}

module B {
  console.log("hello");
}

module A {
  console.log("goodbye");
}
```

left hand type.

Bar(a,b,c) {

Foo extends Bar {
Var a

on Foo() {
apply(this, arguments);

In addition, we agreed to loosen the typing of the test-expressions of the following statements: if, while, do..while, for. Also, the ternary operator test expression has type any.

Does this sound right?

Luke

Current							***Pr ed***
||	any	bool	number	string	interfac e T		**|**
any	any	any	any	any	any		an
bool	any	bool	ERROR	ERROR	ERROR		bo
number	any	ERROR	number	ERROR	ERROR		nu
string	any	ERROR	ERROR	string	ERROR		str
interfac e T	any	ERROR	ERROR	ERROR	T (or ERROR)		int e T
Note: Kind of like (T,T)=> T, but with infectio us 'any'							A | ha typ A
&&	any	bool	number	string	interfac e T		**&**
any	any	any	any	any	any		an

opos					
l	any	bool	number	string	interface T
y	any	any	any	any	any
ol	any	bool	bool	bool	bool
mber	any	number	number	number	number
ing	any	string	string	string	string
terfac T	any	T	T	T	T
|| B s pe of					
&	any	bool	number	string	interface T
y	any	any	any	any	any

bool	any	bool	ERROR	ERROR	ERROR		bo
number	any	ERROR	number	ERROR	ERROR		nu
string	any	ERROR	ERROR	string	ERROR		str
interfac e T	any	ERROR	ERROR	ERROR	T (or ERROR)		int e T
							A & ha typ B

ol	any	bool	number	string	T
mber	any	bool	number	string	T
ing	any	bool	number	string	T
terfac -	any	bool	number	string	T
&& B s pe of					