Minutes of the: ***Ecma TC39-TG1***

held in: ***Phone conference***

on: ***13th September 2006***

Attendees

- Francis Cheng, Adobe Systems
- Dave Herman, Northeastern University
- Graydon Hoare, Mozilla Foundation
- Edwin Smith, Adobe Systems
- Lars Hansen, Opera Software
- Dan Smith, Adobe Systems
- Jeff Dyer, Adobe Systems
- Brendan Eich, Mozilla Foundation
- Pratap Lakshman, Microsoft
- Cormac Flanagan, UC Santa Cruz

Agenda

Grammar issues

Dave's type system questions:

- conversions to `Boolean` only for `if`, `_?_:_`, and boolean operators vs. all types implicitly convertible to `Boolean` (see minutes_aug_23_2006) – what about

  ```
  var x : Boolean = "hello, world!"
  ```

  ?

- implicit vs. explicit conversions; always implicit?
  - from `*` to `T`: implicit
  - from `T` to `Boolean`: implicit
  - from a nullable type to a non-nullable type: implicit (see nullability)
  - from `int` to `double`: ?
  - from `function(int):int` to `function(double):double`: ?
- slightly tricky `is` cases:
  - `(new Object()) is type {}`
  - `(new Function(''print('hello')'')) is type function(...:*):*`
  - `(new Array(1,2,3)) is type []`
- resolved that
  - `Object` is an identical type to `{}`
  - `Function` is an identical type to `function(...:*):*`
  - `Array` is an identical type to `[]`
- clarifying `cast` vs. `to`: `cast` is *only* downcasts, `to` is all conversions?

- subtyping of rest-args with non-rest-args - do we agree that `function(...:int):*` should be a subtype of `function(x:int,y:int):*`?
- runtime conversion ambiguity for unions

Discussion

Grammar Issues

- We have to add some new contextually reserved identifiers because of the use pragma.
  - Dave: What about intrinsic, public and private, why are these reserved?
  - Jeff: Because those tokens appear in the grammar.
  - Lars: Another reason is that there won't be a binding somewhere that changes their meaning.
  - Dave: But is it a problem that you can't use instrinsic on the rhs?
  - Jeff: In AS3 public and private can't be used on the rhs of an assignment statement.
- Brendan: We'll have a hard time reserving new words. For example, if you add new keywords like "let", "yield", "to" etc., you could get legal ES3 programs that produce different results. For Example, the SJ Mercury News uses "yield" as a formal param name. Current idea is to require explicit versioning.
  - Lars: Is that going to be required for all browsers?
  - Brendan: We'll have to think more about compatibility. Doug C wanted us to not reserve anything, but that could cause problems as well. Another suggestion was to use syntax that is illegal.
  - Brendan will prepare something for the face-to-face.
- Dave: "as" and "enum" are still reserved. Is that because some lang, (AS3) use them?
  - Brendan/Jeff: "enum" and "as" are there because of Jscript.net. AS3 has "as" as well.
- Lars: ContextuallyReservedIdentifier appears several places in the grammar but is redundant. Jeff says he'll take a look.
- Brendan: Added notes about "yield" to the wiki. Jeff will take a look at those.

Conversion to Boolean

- Dave: Earlier we decided it should only happen in certain places. We didn't in general want just any type to convert to Boolean. But a few weeks ago, just for simplicity sake we decided to have all types implicitly convert to boolean. This means that the following would get through the compiler:

```
var x : Boolean = "hello, world!"
```

- Dave: One way around this is to make only `if`, `_?_:_`, and boolean operators implicitly convert.
- Jeff: This is a known issue and is a concession to backward compatibility.
- Dave: Okay, I just wanted to make sure everyone is okay with this behavior.
- Jeff: I wouldn't say that it feels fine, but it's the best we could come up with.

Implicit vs. explicit conversions; always implicit?

- Resolved: To conversions are always implicit. You're never required to explicitly say `to` for slots or passing to a function. Implicit `to` conversions happen any time the context expects a certain type but is sent a different type. There are some exceptions for primitive types, for example string to int.
- Brendan: This is written up in the `is as to` proposal.
- Dave: Are the special cases listed out?
- Jeff: Don't know, but we should list them. The ones that come to mind are: String to Number, and Number to String, but there may be others.

Slightly tricky `is` cases

- The following three examples show that there are two different ways to express the same type:
  - `(new Object()) is type {}`
  - `(new Function(''print('hello')'')) is type function(...:*):*`
  - `(new Array(1,2,3)) is type []`
- An important issue that stems from this discussion is the practical implications of this identity relationship for implementors of dictionaries and hash maps. Implementors don't want a change to Object.prototype to affect (i.e. pollute) their dictionary/hashmap.
- Dave: one option is to distinguish structural types from old-school ES3 objects so that "new Object()" is not equal to {}, thus severing the tie between objects created with {} and Object.prototype.
- Brendan: I was thinking of going the other direction and creating a new, concise, syntax for the dictionary/hashmap case. This way we preserve the existing symmetry. I'll write up a proposal that we can discuss at the face-to-face meeting next week.

Clarifying `cast` vs. `to`: `cast` is *only* downcasts?

- Resolved: Yes, `cast` is only used for downcasts.
- Jeff: AS3 behaves this way. There is no user-exposed "cast" operator, but the "as" operator returns null if you try to convert rather than downcast with it. For example:

```
var x = 1.23 as int
```

results in null.

Subtyping of rest-args with non-rest-args

- Dave: Do we agree that `function(...:int):*` should be a subtype of `function(x:int,y:int):*`?
- Resolved: Yes.

Runtime conversion ambiguity for unions

- Jeff: If you have a union type and you're assigning a slot with a union type, which type do you pick if there is ambiguity?
- Lars: You're converting a non-union type to a union type? I don't think we should allow this. This should be a compile time error.
- Dave: One proposal is that you can't have more than one function type in a union.
- Cormac: That everything is convertible to boolean makes this harder.
- Jeff: so this has to be a runtime error.
- Jeff: whats the value of union types?
- Dave: Some kind of union is necessary, but is often done in OOP through subclassing. In functional languages, you usually use a union type (if it's an `a`, do this, if it's a `b` do that, etc.). Our problem is that we don't guarantee that the unions are disjoint.
- Jeff: And with conversion it's difficult to make that guarantee.
- Dave: With the switch type, we resolve by using order of appearance, but this doesn't help with conversions. We could try to use first type declared, but that presents issues.
- Lars: I still don't see the point of converting to union types.
- Cormac: If you have a slot of type Boolean and you want to generalize so that it accepts other types, wouldn't that break old code?
- Dave: A more radical proposal is to revisit union type and make it disjoint. A disjoint union means I'll stick an extra tag to indicate what other types are accepted. For example:

disjoint union: a:int, b:string, c:double

I always know what type of each is because they are labeled a, b, or c. When converting to a disjoint union, you may still have ambiguity.

- Cormac: Another proposition is to drop unions and do nullability another way.
- Brendan: But we have other motivations for union types.
- Cormac: Then as ugly as it sounds, ordered unions may be the best way to get around this problem.
- Jeff: I think that's the least offensive proposal.