

Caja Transformation Rules

Default set of transformations used by Caja

	Rule	Synopsis	Reason	Matches	Substitutes
0	module	Disallow top-level "this". Import free variables.	In Caja, "this" may only be bound to an object when within the object's encapsulation boundary. At top-level level, "this" would be bound to the provided imports object, but the module is outside that object's encapsulation boundary.	@ss*;	@startStmts* ; @ss*;
1	synthetic	Pass through synthetic nodes.	Allow a relied-upon (trusted) translator to supply JavaScript code to be included in the output with no further translation.	<@synthetic>	<@synthetic>
2	block	Initialize named functions at the beginning of their enclosing block.	Nested named function declarations are illegal in ES3 but are universally supported by all JavaScript implementations,	{@ss* ; }	@startStmts* ; @ss* ;

Rule	Synopsis	Reason	Matches	Substitutes
		<p>though in different ways. The compromise semantics currently supported by Caja is to hoist the declaration of a variable with the function's name to the beginning of the enclosing function body or module top level, and to initialize this variable to a new anonymous function every time control re-enters the enclosing block. Note that ES3.1 and ES4 specify a better and safer semantics -- block level lexical scoping -- that we'd like to adopt into Caja eventually. However, it so challenging to implement this semantics by translation to currently-implemented JavaScript that we provide something</p>		

	Rule	Synopsis	Reason	Matches	Substitutes
			<p>quicker and dirtier for now.</p>		
3	with	<p>Statically reject if a `with` block is found.</p>	<p>`with` violates the assumptions made by Scope, and makes it very hard to write a Scope that works. http://yuiblog.com/blog/2006/04/11/with-statement-considered-harmful/ briefly touches on why `with` is bad for programmers. For reviewers -- matching of references with declarations can only be done at runtime. All other secure JS subsets that we know of (ADSafe, Jacaranda, & FBJS) also disallow `with`.</p>	<p>with (@scope) @body;</p>	<p><reject></p>
4	foreach	<p>Only enumerate Caja-visible and enumerable property names. A for-in on "this" will see public and protected property names. Otherwise, only</p>	<p>To enumerate any other property names would be to violate the object's encapsulation, leak internals of the Caja implementation, or violate taming decisions of what should be visible.</p>	<p>for (@k in @o) @ss;</p>	<p><approx > for (@k in @o) { if (__. @canEnum(@o,@k) @ss</p>

Rule	Synopsis	Reason	Matches	Substitutes
	public property names.	When manually reviewing code for vulnerability, experience shows that reviewers cannot pay adequate attention to the pervasive possibility of thrown exceptions. These lead to four dangers: 1) leaking an authority-bearing object, endangering integrity, 2) leaking a secret, endangering secrecy, and 3) aborting a partially completed state update, leaving the state malformed, endangering integrity, and 4) preventing an operation that was needed, endangering availability. Caja only seeks to make strong claims about integrity. By ensuring that only immutable (transitively frozen)	<pre>try { @s0*; } catch (@x) { @s1*; }</pre>	<pre>try { @s0*;} catch (ex___) { try { throw ____.tameException(ex___); } catch (@x) { @s1*; } }</pre>
5	tryCatch	Ensure that only immutable data is thrown, and repair scope confusion in existing JavaScript implementations of try/catch.		

Rule	Synopsis	Reason	Matches	Substitutes
		<p>data is thrown, we prevent problem #1. For the others, programmer vigilance is still needed. Current JavaScript implementations fail, in different ways, to implement the scoping of the catch variable specified in ES3. We translate Caja to JavaScript so as to implement the ES3 specified scoping on current JavaScript implementations.</p>		
6	tryCatchFinally	<p>Finally adds no special issues beyond those explained in try/catch.</p>	<pre>try { @s0*; } catch (@x) { @s1*; } finally { @s2*; }</pre>	<pre>try { @s0*;} catch (ex __) { try { throw __.tameException(ex __);} catch (@x) { @s1*; }} finally { @s2*;} }</pre>
7	tryFinally	<p>See bug 383. Otherwise, it's just the trivial translation.</p>	<pre>try { @s0*; } finally { @s1*; }</pre>	<pre>try { @s0*; } finally { @s1*; }</pre>
8	varArgs	<p>Make all references to the magic "arguments" "arguments"</p>	arguments	a__

	Rule	Synopsis	Reason	Matches	Substitutes
		<p>variable into references to a frozen array containing a snapshot of the actual arguments taken when the function was first entered.</p>	<p>array-like reflection of the values of the parameter variables. However, te typical usage is to pass it to provide access to one's original arguments -- without the intention of providing the ability to mutate the caller's parameter variables. By making a frozen array snapshot with no "callee" property, we provide the least authority assumed by this typical use. The snapshot is made with a "var a___ = ___.args(arguments);" generated at the beginning of the function body.</p>		
9	varThis	<p>Translates all occurrences of "this" to "t___".</p>	<p>The translation is able to worry less about the complex scoping rules of "this". In a function mentioning "this", a "var t___ = this;" is generated at the beginning of the function body.</p>	this	t___

Rule	Synopsis	Reason	Matches	Substitutes
10 varBadSuffix	Statically reject if a variable with <code>`_`</code> suffix is found.	Caja reserves the <code>`_`</code> suffix for internal use.	<code>@v_</code>	<reject>
11 varBadSuffixDeclaration	Statically reject if a variable with <code>`_`</code> suffix is found.	Caja reserves the <code>`_`</code> suffix for internal use.	<approx> (var function) <code>@v_...</code>	<reject>
12 varBadImportSuffix	Statically reject if an imported variable with <code>`_`</code> suffix is found	A module is outside the encapsulation boundary of its imports object, and so cannot address any of that object's protected properties.	<code>@import_</code>	<reject>
13 varFuncFreeze	An escaping occurrence of a function name freezes the function.	By adopting this static rule, we only need to generate freezes for names that are statically known to be function names, rather than freezing at every potential point of use.	<code>@fname</code>	<code>____.primFreeze(@fname)</code>
14 varDefault	Any remaining uses of a variable name are preserved.		<code>@v</code>	<code>@v</code>
15 readBadSuffix	Statically reject if a property has <code>`_`</code> suffix is found.	Caja reserves the <code>`_`</code> suffix for internal use.	<code>@x.@p_</code>	<reject>

Rule	Synopsis	Reason	Matches	Substitutes
16 readInternal	Read a public or protected property.	Since it is addressed from "this.", Caja assumes we are inside the encapsulation boundary of the object bound to "this", and so its protected properties should be accessible.	this.@p	<approx> ____.readProp(t____, @'p')
17 readBadInternal	Statically reject public reading of a property ending with `_,`.	Caja defines variable with a `_,` suffix as protected.	@x.@p_	<reject>
18 readPublic			@o.@p	<approx> ____. readPub(@o, @'p')
19 readIndexInternal			this[@s]	____. readProp(t____, @s)
20 readIndexPublic			@o[@s]	____. readPub(@o, @s)
21 setBadAssignToFunctionName	Statically reject if an assignment expression assigns to a function name.		<approx> @fname @op?= @x	<reject>
22 setBadThis	Statically reject if an expression assigns to `this`.	Invalid JavaScript.	this = @z	<reject>
23 setBadFreeVariable	Statically reject if an expression assigns to a free variable.	This is still controversial (see bug 375). However, the rationale is to prevent code that's	@import = @y	<reject>

	Rule	Synopsis	Reason	Matches	Substitutes
			<p>nested lexically within a module to from introducing mutable state outside its local function-body scope. Without this rule, two nested blocks within the same module could communicate via a pseudo-imported variable that is not declared or used at the outer scope of the module body.</p>		
24	setBadSuffix	<p>Statically reject if a property with `__` suffix is found.</p>	<p>Caja reserves the `__` suffix for internal use.</p>	<pre>@x.@p__ = @z</pre>	<pre><reject></pre>
25	setInternal	<p>Set or create a public or protected property.</p>	<p>We allow methods and constructors within a constructed object to create new properties on itself directly by assignment.</p>	<pre>this.@p = @r</pre>	<pre><approx> __.setProp(t __, @'p', @r)</pre>
26	setMember	<p>Initialize a member of the prototypical object associated with a constructor or</p>	<p>The right hand side of this rule is a "method context" -- a position in which Caja methods can appear. This allows</p>	<pre>@df.prototype.@p = @m</pre>	<pre>___.setMember(@df, @'p', @m)</pre>

Rule	Synopsis	Reason	Matches	Substitutes
	named function, to be inherited by the instances of that function.	unattached methods to be stored in the prototypical object, which is necessary for allowing instances to share these. However, any attempt to obtain access to a method as a value will obtain at most an attached method.		
27 setBadInternal	Cannot publicly access a property ending with ','.	Caja defines variable with a `` suffix as protected.	@x.@y_ = @z	<reject>
28 setStatic	Initialize the direct properties (static members) of a potentially-mutable constructor or named function.		@fname.@p = @r	____.setStatic(@fname, @'p', @r)
29 setPublic	Set a public property.	If the object is an unfrozen JSONContainer (a record or array), then this will create the own property if needed. If it is an unfrozen constructed object, then clients	@o.@p = @r	<approx> ____.setPub(@o, @'p', @r);

Rule	Synopsis	Reason	Matches	Substitutes
		can assign to existing public own properties, but cannot directly create such properties.		
30 setIndexInternal			this[@s] = @r	___.setProp(t___, @s, @r)
31 setIndexPublic			@ol[@s] = @r	___.setPub(@o, @s, @r)
32 setBadInitialize	Statically reject if a variable with `__` suffix is found.	Caja reserves the `__` suffix for internal use.	var @v__ = @r	<reject>
33 setInitialize	Ensure v is not a function name. Expand the right side.		var @v = @r	var @v = @r
34 setBadDeclare	Statically reject if a variable with `__` suffix is found.	Caja reserves the `__` suffix for internal use.	var @v__	<reject>
35 setDeclare	Ensure that v isn't a function name.		var @v	var @v
36 setBadVar	Statically reject if a variable with `__` suffix is found.	Caja reserves the `__` suffix for internal use.	@v__ = @r	<reject>
37 setVar	Only if v isn't a function name.		@v = @r	@v = @r
38 setReadModifyWriteLocalVar			@x @op = @y	<approx> @x = @x @op @y

	Rule	Synopsis	Reason	Matches	Substitutes
39	setIncrDecr	Handle pre and post ++ and --.		UNKNOWN	UNKNOWN
40	newCalllessCtor	Add missing empty argument list.	JavaScript syntax allows constructor calls without "()".	new @ctor	<expand> new @ctor()
41	newCtor			new @ctor(@as*)	new (____.asCtor(@ctor)) (@as*)
42	deleteBadSuffix			delete @o.@p__	<reject>
43	deleteInternal			delete this.@p	____.deleteProp(t____, @'p')
44	deleteBadInternal			delete @o.@p_	<reject>
45	deletePublic			delete @o.@p	____.deletePub(@o, @'p')
46	deleteIndexInternal			delete this[@s]	____.deleteProp(t____, @s)
47	deleteIndexPublic			delete @o[@s]	____.deletePub(@o, @s)
48	deleteNonProperty			delete @v	<reject>
49	callBadSuffix	Statically reject if a selector with `__` suffix is found.	Caja reserves the `__` suffix for internal use.	@o.@p__(@as*)	<reject>
50	callInternal			this.@p(@as*)	<approx> ____callProp(t____, @'p', [@as*])
51	callBadInternal	Statically reject if a public selector with `__` suffix is found.	Caja defines selectors with a `__` as private.	@o.@s__(@as*)	<reject>
52	callCajaDef2	Declares that the first argument acts as a derived constructor	Sets up a well formed prototype inheritance chain between these two functions. The first argument must	caja.def(@fname, @base)	caja.def(@fname, @base)

	Rule	Synopsis	Reason	Matches	Substitutes
		inheriting from the second.	be a declared function name. Calling <code>caja.def()</code> on it does not freeze it.		
53	callCajaDef2BadFunction	Reject calls to <code>caja.def()</code> on names of functions statically known to be frozen.	Within a function <code>foo()</code> , <code>foo</code> must already be frozen, so it is too late to initialize it.	<code>caja.def(@fname, @base)</code>	<reject>
54	callCajaDef2Bad	Reject other calls to <code>caja.def()</code> .	If the first argument is not a declared function name, then it cannot be an unfrozen function.	<code>caja.def(@x, @base)</code>	<reject>
55	callCajaDef3Plus	Declare an inheritance relationship, and initialize methods and statics.	The enumerable own properties of the third and fourth arguments, if present are used to initialize <code>@fname.prototype</code> and <code>@fname</code> , respectively. The third argument must statically be an object-literal expression. The value positions of this expression is a method context -- a position in which methods are allowed.	<code>caja.def(@fname, @base, @mm, @ss?)</code>	<code>caja.def(@fname, @base, @mm, @ss?)</code>

Rule	Synopsis	Reason	Matches	Substitutes
56	callCajaDef3PlusBadFunction	Reject initialization of a name of a function statically known to be frozen.	caja.def(@fname, @base, @mm, @ss?)	<reject>
57	callCajaDef3PlusBad	Reject other calls to caja.def().	caja.def(@x, @base, @mm, @ss?)	<reject>
58	callFuncInLineMethodCall		(function (@formals*) { @body*; }) .call(this, @args*);	(function (@formals*) { @fh*, @stmts*; @body*; }) .call(this, @args*);
59	callFuncInLineMethodApply		(function (@formals*) { @body*; }) .apply(this, @arg);	(function (@formals*) { @fh*, @stmts*; @body*; }) .apply(this, @arg);
60	callFuncInLineMethodBind		(function (@formals*) { @body*; }) .bind(this, @args*);	(function (@formals*) { @fh*, @stmts*; @body*; }) .bind(t___, @args*);
61	callPublic		@o. @p(@as*)	<approx > ____.callPub(@o, @'p', [@as*])
62	callIndexInternal		this[@s][@as*)	____.callProp(t___, @s, [@as*])
63	callIndexPublic		@o[@s][@as*)	____.callPub(@o, @s, [@as*])
64	callFunc		@f(@as*)	____.asSimpleFunc(@f @as*)

Rule	Synopsis	Reason	Matches	Substitutes
65 funcAnonSimple			function (@ps*) { @bs*; }	____.primFreeze(____.simpleFunc(function (@ps*) { @fh*, @stmts*, @bs*; })))
66 funcNamedSimpleDecl			function @fname(@ps*) { @bs*; }	@fr = ____ .simpleFunc(function @fname(@ps*) { @fh*, @stmts*, @bs*; }));
67 funcNamedSimpleValue			function @fname(@ps*) { @bs*; }	____.primFreeze(____.simpleFunc(function @fname(@ps*) { @fh*, @stmts*, @bs*; }));
68 funcX04a	Rewrites an 1) anonymous function 2) mentioning this 3) whose earliest function scope ancestor is NOT a constructor or method into an exophoric function.	A moderately risky stepping stone to ease the conversion of old code.	(function (@formals*) { @body*; })	<approx> ____.x04a(function (@formals*) { @fh*, @stmts*, @body*; })
69 funcCtor			function @fname(@ps*) { @b; @bs*; }	<approx> @fname = (function () { ____.splitCtor(@fRef, @f_init __Ref); function @fname(var_args) { return new @fRef.make__(arguments); } function @f_init(@ps*) { @fh*, @stmts*; @b; @bs*; }

	Rule	Synopsis	Reason	Matches	Substitutes
					return @fRef;})();
70	mapEmpty			{}	{}
71	mapBadKeySuffix	Statically reject if a key with ` _ ` suffix is found		<approx> { @keys_*: @vals* }	<reject>
72	mapNonEmpty			{ @keys* : @vals* }	{ @keys* : @vals* }
73	multiDeclaration			UNKNOWN	UNKNOWN
74	otherInstanceof			@o instanceof @f	@o instanceof @f
75	otherTypeof			typeof @f	<approx > typeof ____ .readPub(IMPORTS __, @f')
76	otherSpecialOp			UNKNOWN	UNKNOWN
77	labeledStatement	Statically reject if a label with ` _ ` suffix is found	Caja reserves the ` _ ` suffix for internal use	UNKNOWN	UNKNOWN
78	regexLiteral	Use the regular expression constructor	So that every use of a regex literal creates a new instance to prevent state from leaking via interned literals. This is consistent with the way ES4 treats regex literals.	UNKNOWN	new ____ .RegExp(@pattern, @modifiers?)
79	recurse	Automatically recurse into some structures		UNKNOWN	UNKNOWN