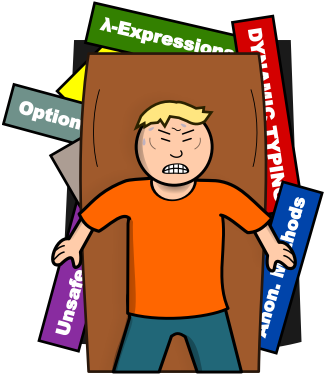
Adding Dynamic Types to $C^{\#}$



Michael Hansen

Indiana University

November 3, 2010

What is $C^\#$?

- If C++ and Java had a baby...and it was adopted by Microsoft
- Modern C-style language for the .NET runtime
  - Imperative, object-oriented
  - JIT compiled - CIL (bytecode) on the CLR (run-time)
  - Garbage collected
  - Unsafe code
  - Parametric polymorphism with constraints (generics)
  - First-class functions (λ-expressions, delegates)
- Version 4.0 now has
  - Covariance and contravariance for generics and delegates
  - Keyword and optional parameters
  - Late-binding via dynamic types (you are here)
- Mono Project = .NET on Linux and Mac!
  - Tell your friends

What is $C^{\#}$?

- If C++ and Java had a baby...and it was adopted by Microsoft
- Modern C-style language for the .NET runtime
  - Imperative, object-oriented
  - JIT compiled - CIL (bytecode) on the CLR (run-time)
  - Garbage collected
  - Unsafe code
  - Parametric polymorphism with constraints (generics)
  - First-class functions (λ-expressions, delegates)
- Version 4.0 now has
  - Covariance and contravariance for generics and delegates
  - Keyword and optional parameters
  - Late-binding via dynamic types (you are here)
- Mono Project = .NET on Linux and Mac!
  - Tell your friends

What is C$^{\#}$?

- If C++ and Java had a baby...and it was adopted by Microsoft
- Modern C-style language for the .NET runtime
  - Imperative, object-oriented
  - JIT compiled - CIL (bytecode) on the CLR (run-time)
  - Garbage collected
  - Unsafe code
  - Parametric polymorphism with constraints (generics)
  - First-class functions (λ-expressions, delegates)
- Version 4.0 now has
  - Covariance and contravariance for generics and delegates
  - Keyword and optional parameters
  - Late-binding via dynamic types (you are here)
- Mono Project = .NET on Linux and Mac!
  - Tell your friends

What is $C^{\#}$?

- If C++ and Java had a baby…and it was adopted by Microsoft
- Modern C-style language for the .NET runtime
  - Imperative, object-oriented
  - JIT compiled - CIL (bytecode) on the CLR (run-time)
  - Garbage collected
  - Unsafe code
  - Parametric polymorphism with constraints (generics)
  - First-class functions (λ-expressions, delegates)
- Version 4.0 now has
  - Covariance and contravariance for generics and delegates
  - Keyword and optional parameters
  - Late-binding via dynamic types (you are here)
- Mono Project = .NET on Linux and Mac!
  - Tell your friends

Contributions

- Define a core fragment of $\mathrm{C}^{\#}$ 4.0 (Featherweight $\mathrm{C}^{\#}$)
- Translation $\mathrm{FC}_4^{\#} \rightarrow \mathrm{C}_{\mathrm{CLR}}^{\#}$
- Addition of dynamic type
  - Subtyping transitivity is maintained (*in your face, Siek and Taha*)
- Operational semantics for $\mathrm{C}_{\mathrm{CLR}}^{\#}$
  - Prove type soundness
  - **Note**: Unsafe code is not included in $\mathrm{C}_{\mathrm{CLR}}^{\#}$

Terminology

Term	Meaning
Delegate	Function pointer
Value Type	Stack-based, passed by value
Reference Type	Heap-based, passed by reference
Boxing	Value Type → Reference Type
Unboxing	Reference Type → Value Type
DLR	Dynamic Language Runtime (efficient runtime dispatch)

C$^{\#}$ in 20 Seconds!

```
class Foomatic {
  public int Bazar { get; set; } // Read/write property
  public int Fooz(bool bar) {
    return bar ? Bazar : Bazar * 2;
  }
}

class Program {
  static void Main(string[] args) {
    var myFoo = new Foomatic(); // Construct a Foomatic
    myFoo.Bazar = 10;

    Console.WriteLine(myFoo.Bazar); // 10
    Console.WriteLine(myFoo.Fooz(false)); // 20

    myFoo.GetType().GetProperty("Bazar")
      .SetValue(myFoo, 50); // myFoo.Bazar = 50

    Console.WriteLine(myFoo.Bazar); // 50
    Console.WriteLine(typeof(Foomatic).GetMethod("Fooz")
      .Invoke(myFoo, new object[] { false })); // 100
  }
}
```

Motivation

- Interact cleanly with COM components

```
var word = new Word.Application();
word.Visible = true;
word.Documents.Add();
// ...
word.Selection.PasteSpecial(Link: true, DisplayAsIcon: true);
```

- Access DLR objects

```
dynamic random = Python.CreateRuntime.UseFile("random.py");
random.shuffle(Enumerable.Range(0, 100).ToArray());
```

- Make $C^{\#}$ a better language for web scripting (i.e. Silverlight)

```
dynamic doc = HtmlPage.Document;
doc.Title = "Hello World";
```

Motivation

- Interact cleanly with COM components

```
var word = new Word.Application();
word.Visible = true;
word.Documents.Add();
// ...
word.Selection.PasteSpecial(Link: true, DisplayAsIcon: true);
```

- Access DLR objects

```
dynamic random = Python.CreateRuntime.UseFile("random.py");
random.shuffle(Enumerable.Range(0, 100).ToArray());
```

- Make C# a better language for web scripting (i.e. Silverlight)

```
dynamic doc = HtmlPage.Document;
doc.Title = "Hello World";
```

Motivation

- Interact cleanly with COM components

```
var word = new Word.Application();
word.Visible = true;
word.Documents.Add();
// ...
word.Selection.PasteSpecial(Link: true, DisplayAsIcon: true);
```

- Access DLR objects

```
dynamic random = Python.CreateRuntime.UseFile("random.py");
random.shuffle(Enumerable.Range(0, 100).ToArray());
```

- Make $\mathrm{C}^{\#}$ a better language for web scripting (i.e. Silverlight)

```
dynamic doc = HtmlPage.Document;
doc.Title = "Hello World";
```

The dynamic Type

- Implicitly convertible to any type

```
dynamic d = "Hello World"; // Succeeds, string <: dynamic
int i = d; // Fails at runtime, dynamic :> int
```

- Method calls and property accesses are resolved at runtime

```
dynamic dynObject = someObject;
dynObject.Field = 5; // [...].GetProperty("Field").SetValue(someObject, 5);
dynObject.Method(); // [...].GetMethod("Method").Invoke(someObject);
```

- Runtime type of dynamic subexpressions for method resolution

```
void M(byte b, int i) { ... }
void M(short s, int i) { ... }

short s = 42; dynamic d = 7; int i = 42; // Numeric literals are ints
M(s, 7);        //(1) short, int
M(42, 7);       //(2) byte, int
M(s, d);        //(3) short, int
M(42, d);       //(4) byte, int
M(i, 7);        //(5) FAIL at compile-time - no (int, int) overload
M(i, d);        //(6) FAIL at compile-time - no overload permits i
M(d, i);        //(7) FAIL at runtime - no (int,int) overload
```

The dynamic Type

- Implicitly convertible to any type

```
dynamic d = "Hello World"; // Succeeds, string <: dynamic
int i = d; // Fails at runtime, dynamic :> int
```

- Method calls and property accesses are resolved at runtime

```
dynamic dynObject = someObject;
dynObject.Field = 5; // [...].GetProperty("Field").SetValue(someObject, 5);
dynObject.Method(); // [...].GetMethod("Method").Invoke(someObject);
```

- Runtime type of dynamic subexpressions for method resolution

```
void M(byte b, int i) { ... }
void M(short s, int i) { ... }

short s = 42; dynamic d = 7; int i = 42; // Numeric literals are ints
M(s, 7);        //(1) short, int
M(42, 7);       //(2) byte, int
M(s, d);        //(3) short, int
M(42, d);       //(4) byte, int
M(i, 7);        //(5) FAIL at compile-time - no (int, int) overload
M(i, d);        //(6) FAIL at compile-time - no overload permits i
M(d, i);        //(7) FAIL at runtime - no (int,int) overload
```

The dynamic Type

- Implicitly convertible to any type

```
dynamic d = "Hello World"; // Succeeds, string <: dynamic
int i = d; // Fails at runtime, dynamic :> int
```

- Method calls and property accesses are resolved at runtime

```
dynamic dynObject = someObject;
dynObject.Field = 5; // [...].GetProperty("Field").SetValue(someObject, 5);
dynObject.Method(); // [...].GetMethod("Method").Invoke(someObject);
```

- Runtime type of dynamic subexpressions for method resolution

```
void M(byte b, int i) { ... }
void M(short s, int i) { ... }

short s = 42; dynamic d = 7; int i = 42; // Numeric literals are ints
M(s, 7);        //(1) short, int
M(42, 7);       //(2) byte, int
M(s, d);        //(3) short, int
M(42, d);       //(4) byte, int
M(i, 7);        //(5) FAIL at compile-time - no (int, int) overload
M(i, d);        //(6) FAIL at compile-time - no overload permits i
M(d, i);        //(7) FAIL at runtime - no (int,int) overload
```

Summary of Changes

- $C^{\#}$ now has type dynamic
- Converted to object at runtime
  - Treated specially by the compiler
- A dynamic expression can be converted to any type
  - Runtime type test inserted
- Method calls with dynamic subexpressions deferred to runtime
  - Compile-time types of non-dynamic subexpressions for resolution

Bidirectional Type System

- Type checking

```
T x = e; // Ensure that e can be converted to type T
```

- Type synthesis

```
var y = e; // Determine a type for e, and consequently y
```

- Subtle differences...

```
Button x = null; // null can be converted to reference type Button
var y = null; // null does NOT synthesize a type!
```

- Coercive subtyping
  - If T is a subtype of S, generate a coercion C s.t. $C(T) = S$
- Most formalizations use declarative typing and subtyping judgements

Bidirectional Type System

- Type checking

```
T x = e; // Ensure that e can be converted to type T
```

- Type synthesis

```
var y = e; // Determine a type for e, and consequently y
```

- Subtle differences...

```
Button x = null; // null can be converted to reference type Button
var y = null; // null does NOT synthesize a type!
```

- Coercive subtyping
  - If T is a subtype of S, generate a coercion C s.t. $C(T) = S$
- Most formalizations use declarative typing and subtyping judgements

Bidirectional Type System

- Type checking

```
T x = e; // Ensure that e can be converted to type T
```

- Type synthesis

```
var y = e; // Determine a type for e, and consequently y
```

- Subtle differences...

```
Button x = null; // null can be converted to reference type Button
var y = null; // null does NOT synthesize a type!
```

- Coercive subtyping
  - If T is a subtype of S, generate a coercion C s.t. $C(T) = S$
- Most formalizations use declarative typing and subtyping judgements

Bidirectional Type System

- Type checking

```
T x = e; // Ensure that e can be converted to type T
```

- Type synthesis

```
var y = e; // Determine a type for e, and consequently y
```

- Subtle differences...

```
Button x = null; // null can be converted to reference type Button
var y = null; // null does NOT synthesize a type!
```

- Coercive subtyping
  - If T is a subtype of S, generate a coercion C s.t. $C(T) = S$
- Most formalizations use declarative typing and subtyping judgements

Featherweight C#

- Completely valid subset of C# 4.0
- Classes, generics, overloading, inheritance, side-effects
- Constructors treated as normal methods (`.ctor`)
- Assume a unique entry point `main`

Featherweight $C^{\#}$ Grammar

Programs

$p ::= \overline{cd}$	Program
$cd ::=$	Class declaration
$\quad$ public class $C<\overline{X}>:C<\overline{\sigma}>\ \{\overline{fd}\ \overline{md}\ \overline{cmd}\}$	
$fd ::=$ public $\sigma\ f;$	Field declaration
$md ::=$	Method declaration
$\quad$ public virtual $\sigma\ m<\overline{X}>(\overline{\sigma}\ \overline{x})\{\overline{s}\}$	
$\quad$ public override $\sigma\ m<\overline{X}>(\overline{\sigma}\ \overline{x})\{\overline{s}\}$	
$cmd ::=$	Constructor method declaration
$\quad$ public $C<\overline{X}>(\overline{\sigma}\ \overline{x})$:this$(\overline{e})\{\overline{s}\}$	
$\quad$ public $C<\overline{X}>(\overline{\sigma}\ \overline{x})$:base$(\overline{e})\{\overline{s}\}$	

Types

$\sigma ::=$	Type
$\quad \gamma$	$\quad$ Value type
$\quad \rho$	$\quad$ Reference type
$\quad X$	$\quad$ Type parameter
$\gamma ::=$	Value Type
$\quad$ bool	$\quad$ Boolean
$\quad$ int	$\quad$ Integer
$\quad$ byte	$\quad$ Byte
$\rho ::=$	Reference Type
$\quad C<\overline{\sigma}>$	$\quad$ Class type (including object and dynamic)
$\quad D<\overline{\sigma}>$	$\quad$ Delegate type

Statements

$s ::=$	Statement
$\quad ;$	$\quad$ Skip
$\quad se;$	$\quad$ Expression statement
$\quad$ if $(e)\ s$ else s	$\quad$ Conditional statement
$\quad \sigma\ x = e;$	$\quad$ Variable declaration
$\quad e.f = e;$	$\quad$ Field assignment
$\quad$ return $e;$	$\quad$ Return statement
$\quad \{\overline{s}\}$	$\quad$ Block

Expressions

$e ::=$	Expression
$\quad b$	$\quad$ Boolean
$\quad i$	$\quad$ Integer
$\quad e \oplus e$	$\quad$ Built-in operator
$\quad x$	$\quad$ Variable
$\quad$ null	$\quad$ Null
$\quad (\sigma)e$	$\quad$ Cast
$\quad e.f$	$\quad$ Field access
$\quad$ delegate $(\overline{\sigma}\ \overline{x})\{\overline{s}\}$	$\quad$ Anonymous method expression
$\quad se$	$\quad$ Statement expression
$se ::=$	Statement expression
$\quad e(\overline{e})$	$\quad$ Delegate invocation
$\quad e.m<\overline{\sigma}>(\overline{e})$	$\quad$ Method invocation
$\quad$ new $C<\overline{\sigma}>(\overline{e})$	$\quad$ Object creation
$\quad x = e$	$\quad$ Variable assignment

$C^{\#}{}_{\text{CLR}}$

- Non-trivial conversions are explicit (subtyping = subclassing)
- Method invocations are fully resolved
- Explicit dynamic operations available
- $\text{dynamic} \rightarrow \text{object}$ ($\text{dynamic} \notin \theta$)

$C^{\#}{}_{\mathrm{CLR}}$ Grammar

Expressions

E ::=	Target expressions
b	Boolean
i	Integer
$E \oplus E$	Built-in operator
x	Variable
`null`	Null
$E.f$	Field access
$\texttt{delegate}\ (\overline{\tau\ x})\{\overline{S}\}$	Anonymous method expression
CE	Conversion Expression
DE	Dynamic Expression
SE	Statement expression
CE ::=	Conversion Expression
$\mathsf{ByteToInt}(E)$	Byte to Integer conversion
$\mathsf{IntToByte}(E)$	Integer to Byte conversion
$\mathsf{Box}[\gamma](E)$	Boxing conversion
$\mathsf{Unbox}[\gamma](E)$	Unboxing conversion
$\mathsf{Downcast}[\rho](E)$	Downcast
DE ::=	Dynamic Expression
$\mathsf{Convert}[\sigma](E)$	Dynamic type test
$\mathsf{MemberAccess}[f](E\colon\sigma)$	Dynamic field selection
$\mathsf{DInvoke}(E\colon\sigma, \overline{E\colon\sigma})$	Dynamic delegate invocation
$\mathsf{ObjectCreate}[\rho](\overline{E\colon\sigma})$	Dynamic object creation
$\mathsf{MInvoke}[m](E\colon\sigma, \overline{E\colon\sigma})$	Dynamic method invocation

Expressions (cont)

MD ::=	Target method descriptor
$\texttt{C<}\overline{X_C}\texttt{><}\overline{\tau_C}\texttt{>::}m\texttt{<}\overline{X_m}\texttt{><}\overline{\tau_m}\texttt{>:}(\overline{\tau_p}) \rightarrow \tau_r$	
SE ::=	Statement expression
$E(\overline{E})$	Delegate invocation
$E.MD(\overline{E})$	Method invocation
$\texttt{new}\ MD(\overline{E})$	Object creation
$x = E$	Variable assignment

Statements

S ::=	Statement
;	Skip
SE;	Expression statement
$\texttt{if}\ (E)\ S\ \texttt{else}\ S$	Conditional statement
$\tau\ x = E$;	Variable declaration
$E.f = E$;	Field assignment
$\texttt{return}\ E$;	Return statement
$\{\overline{s}\}$	Block
$\mathsf{Assign}[f](E\colon\sigma, E\colon\sigma)$;	Dynamic field assignment

Notation

- $|\mathrm{C} < \mathrm{int}, \mathrm{dynamic} > |^* = \mathrm{C} < \mathrm{int}, \mathrm{object} >$
- $\Gamma \triangleright E \leq \tau$
  - In Γ, E can be **converted** to type τ
- $\Gamma \triangleright E \uparrow \tau$
  - In Γ, E **synthesizes** type τ
- $\Gamma \vdash E \uparrow^+ \tau$
  - Same as $\uparrow$, except $\mathrm{null} \uparrow^+ \mathrm{object}$ and int literals are recorded
- $\mathit{ftype}(\sigma, f)$
  - Get type of field f in type σ
- $\mathit{dtype}(\mathrm{D})(\overline{\sigma}) = \overline{\sigma_2} \to \sigma_3$
  - Get type of delegate D, substitute in types σ, end up with type $\overline{\sigma_2} \to \sigma_3$
- $\mathit{mtype}(\sigma, m)$
  - Get method signatures named m reachable from type σ

$\mathrm{C}^{\#}{}_{\mathrm{CLR}}$ Type Conversion

$$\text{[CLR-Byte]}\frac{0 \leq \underline{i} \leq 255}{\Gamma \rhd \underline{i} \leq \texttt{byte}} \quad \text{[CLR-Null]}\frac{}{\Gamma \rhd \texttt{null} \leq \theta}$$

$$\text{[CLR-AME]}\frac{|dtype(\texttt{D})(\overline{\tau})|^{\star} = \overline{\tau_0} \to \tau_1 \qquad \Gamma, \overline{x}{:}\,\overline{\tau_0} \rhd \overline{S_1} \leq \tau_1}{\Gamma \rhd \texttt{delegate(}\overline{\tau_0}\ \overline{x}\texttt{)\{}\overline{S_1}\texttt{\}} \leq \texttt{D<}\overline{\tau}\texttt{>}}$$

$$\text{[CLR-Synth]}\frac{\Gamma \rhd e_1 \uparrow \tau_0 \qquad \tau_0 \leq \tau_1}{\Gamma \rhd e_1 \leq \tau_1}$$

$$\text{[CLR-Skip]}\frac{}{\Gamma \rhd \texttt{;} \leq \tau} \quad \text{[CLR-ExpStatement]}\frac{\Gamma \rhd SE_1 \uparrow \tau_1}{\Gamma \rhd SE_1\texttt{;} \leq \tau}$$

$$\text{[CLR-Cond]}\frac{\Gamma \rhd E_1 \leq \texttt{bool} \qquad \Gamma \rhd S_1 \leq \tau \qquad \Gamma \rhd S_2 \leq \tau}{\Gamma \rhd \texttt{if (}E_1\texttt{)}\ S_1\ \texttt{else}\ S_2 \leq \tau}$$

$$\text{[CLR-FAss]}\frac{\Gamma \rhd E_1 \uparrow \tau_1 \qquad |ftype(\tau_1, f)|^{\star} = \tau_2 \qquad \Gamma \rhd E_2 \leq \tau_2}{\Gamma \rhd E_1\texttt{.}f\texttt{=}E_2\texttt{;} \leq \tau}$$

$$\text{[CLR-FAssDyn]}\frac{\Gamma \rhd E_1 \leq |\sigma_1|^{\star} \qquad \Gamma \rhd E_2 \leq |\sigma_2|^{\star}}{\Gamma \rhd \mathsf{Assign}[f](E_1{:}\,\sigma_1, E_2{:}\,\sigma_2)\texttt{;} \leq \tau}$$

$$\text{[CLR-ReturnExp]}\frac{\Gamma \rhd E_1 \leq \tau_1}{\Gamma \rhd \texttt{return}\ E_1\texttt{;} \leq \tau_1}$$

$$\text{[CLR-Seq]}\frac{\Gamma \rhd E_1 \leq \tau_1 \qquad x \notin dom(\Gamma) \qquad \Gamma, x{:}\,\tau_1 \rhd \overline{S_1} \leq \tau}{\Gamma \rhd \tau_1\ x\ \texttt{=}\ E_1\texttt{;}\ \overline{S_1} \leq \tau}$$

$\mathrm{C^{\#}}_{\mathrm{CLR}}$ Type Synthesis (1/2)

$$[\text{CLR-S-Int}]\ \frac{}{\Gamma \rhd \underline{i} \uparrow \texttt{int}} \qquad [\text{CLR-S-Bool}]\ \frac{}{\Gamma \rhd \underline{b} \uparrow \texttt{bool}}$$

$$[\text{CLR-S-Var}]\ \frac{}{\Gamma, x{:}\,\tau \rhd x \uparrow \tau} \qquad [\text{CLR-S-Field}]\ \frac{\Gamma \rhd E_1 \uparrow \tau_1 \qquad |\mathit{ftype}(\tau_1, f)|^\star = \tau_2}{\Gamma \rhd E_1\,.\,f \uparrow \tau_2}$$

$$[\text{CLR-S-DelInv}]\ \frac{\Gamma \rhd E_1 \uparrow \texttt{D<}\overline{\tau}\texttt{>} \qquad |\mathit{dtype}(\texttt{D})(\overline{\tau})|^\star = \overline{\tau_1} \rightarrow \tau_2 \qquad \Gamma \rhd \overline{E_2} \leq \overline{\tau_1}}{\Gamma \rhd E_1(\overline{E_2}) \uparrow \tau_2}$$

$$[\text{CLR-S-New}]\ \frac{MD = \texttt{C<}\overline{X_C}\texttt{><}\overline{\tau_C}\texttt{>::.ctor:}\,(\overline{\tau_p}) \qquad \Gamma \rhd \overline{e_1} \leq \overline{\tau_p}[\overline{X_C} := \overline{\tau_C}]}{\Gamma \rhd \texttt{new}\ MD(\overline{E_1}) \uparrow \texttt{C<}\overline{\tau_C}\texttt{>}}$$

$$[\text{CLR-S-VarAssign}]\ \frac{\Gamma, x_1{:}\,\tau_1 \rhd E_1 \leq \tau_1}{\Gamma, x_1{:}\,\tau_1 \rhd x_1 \texttt{ = } E_1 \uparrow \tau_1}$$

$$[\text{CLR-S-MethInv}]\ \frac{\begin{array}{c} MD = \texttt{C<}\overline{X_C}\texttt{><}\overline{\tau_C}\texttt{>::}m\texttt{<}\overline{X_m}\texttt{><}\overline{\tau_1}\texttt{>:}\,(\overline{\tau_p}) \rightarrow \tau_r \\ \Gamma \rhd E_1 \leq \texttt{C<}\overline{\tau_C}\texttt{>} \qquad \Gamma \rhd \overline{E_2} \leq \overline{\tau_p}[\overline{X_C}, \overline{X_m} := \overline{\tau_C}, \overline{\tau_1}] \end{array}}{\Gamma \rhd E_1\,.\,MD(\overline{E_2}) \uparrow \tau_r}$$

$\mathrm{C^{\#}}_{\mathrm{CLR}}$ Type Synthesis (2/2)

$$[\text{CLR-S-B2I}]\frac{\Gamma \rhd E \leq \texttt{byte}}{\Gamma \rhd \text{ByteToInt}(E) \uparrow \texttt{int}} \quad [\text{CLR-S-I2B}]\frac{\Gamma \rhd E \leq \texttt{int}}{\Gamma \rhd \text{IntToByte}(E) \uparrow \texttt{byte}}$$

$$[\text{CLR-S-Box}]\frac{\Gamma \rhd E \leq \gamma}{\Gamma \rhd \text{Box}[\gamma](E) \uparrow \texttt{object}} \quad [\text{CLR-S-Unbox}]\frac{\Gamma \rhd E \leq \texttt{object}}{\Gamma \rhd \text{Unbox}[\gamma](E) \uparrow \gamma}$$

$$[\text{CLR-S-Downcast}]\frac{\Gamma \rhd E \uparrow \tau \qquad \tau \leq |\rho|^{\star}}{\Gamma \rhd \text{Downcast}[\rho](E) \uparrow |\rho|^{\star}} \quad [\text{CLR-S-DConv}]\frac{\Gamma \rhd E \leq \texttt{object}}{\Gamma \rhd \text{Convert}[\sigma](E) \uparrow |\sigma|^{\star}}$$

$$[\text{CLR-S-DMemAcc}]\frac{\Gamma \rhd E \leq \texttt{object}}{\Gamma \rhd \text{MemberAccess}[f](E) \uparrow \texttt{object}}$$

$$[\text{CLR-S-DDelInv}]\frac{\Gamma \rhd E_0 \leq |\sigma_0|^{\star} \qquad \sigma_0 = \texttt{dynamic} \text{ or } D\texttt{<}\overline{\sigma}\texttt{>} \qquad \Gamma \rhd \overline{E_1} \leq |\overline{\sigma_1}|^{\star}}{\Gamma \rhd \text{DInvoke}(E_0\colon \sigma_0, \overline{E_1\colon \sigma_1}) \uparrow \texttt{object}}$$

$$[\text{CLR-S-DNew}]\frac{\Gamma \rhd \overline{E} \leq |\overline{\sigma}|^{\star}}{\Gamma \rhd \text{ObjectCreate}[\rho](\overline{E\colon \sigma}) \uparrow |\rho|^{\star}}$$

$$[\text{CLR-S-DMethInv}]\frac{\Gamma \rhd E_0 \leq |\sigma_0|^{\star} \qquad \sigma_0 = \texttt{dynamic} \text{ or } C\texttt{<}\overline{\sigma}\texttt{>} \qquad \Gamma \rhd \overline{E_1} \leq |\overline{\sigma_1}|^{\star}}{\Gamma \rhd \text{MInvoke}[m](E_0\colon \sigma_0, \overline{E_1\colon \sigma_1}) \uparrow \texttt{object}}$$

Translation to $\mathrm{C^{\#}}_{\mathrm{CLR}}$ (Type Conversion)

- Subclassing relation $C_1 < \overline{\sigma_1} > \, : \, C_2 < \overline{\sigma_2} >$

$$\frac{}{\rho : \rho} \qquad \frac{\texttt{class } \texttt{C}_1\texttt{<}\overline{X}\texttt{>:C}_2\texttt{<}\overline{\sigma_2}\texttt{>} \qquad \texttt{C}_2\texttt{<}\overline{\sigma_2}\texttt{>}[\overline{X} := \overline{\sigma_1}] : \texttt{C}_3\texttt{<}\overline{\sigma_3}\texttt{>}}{\texttt{C}_1\texttt{<}\overline{\sigma_1}\texttt{>} : \texttt{C}_3\texttt{<}\overline{\sigma_3}\texttt{>}}$$

- Implicit conversion

$$[\text{IC-Refl}]\frac{}{\sigma_1 <:_{\mathsf{i}} \sigma_1 \rightsquigarrow \bullet} \qquad [\text{IC-ByteToInt}]\frac{}{\texttt{byte} <:_{\mathsf{i}} \texttt{int} \rightsquigarrow \mathsf{ByteToInt}(\bullet)}$$

$$[\text{IC-Val-Obj}]\frac{}{\gamma <:_{\mathsf{i}} \texttt{object} \rightsquigarrow \mathsf{Box}[\gamma](\bullet)} \qquad [\text{IC-Ref-Obj}]\frac{}{\rho <:_{\mathsf{i}} \texttt{object} \rightsquigarrow \bullet}$$

$$[\text{IC-Sub}]\frac{\texttt{C}_1\texttt{<}\overline{\sigma_1}\texttt{>} : \texttt{C}_2\texttt{<}\overline{\sigma_2}\texttt{>}}{\texttt{C}_1\texttt{<}\overline{\sigma_1}\texttt{>} <:_{\mathsf{i}} \texttt{C}_2\texttt{<}\overline{\sigma_2}\texttt{>} \rightsquigarrow \bullet} \qquad [\text{IC-Dynamic}]\frac{\sigma <:_{\mathsf{i}} \texttt{object} \rightsquigarrow C}{\sigma <:_{\mathsf{i}} \texttt{dynamic} \rightsquigarrow C}$$

- Explicit conversion (cast required)

$$[\text{XC-Refl}]\frac{}{\sigma_1 <:_{\mathsf{x}} \sigma_1 \rightsquigarrow \bullet} \qquad [\text{XC-IntToByte}]\frac{}{\texttt{int} <:_{\mathsf{x}} \texttt{byte} \rightsquigarrow \mathsf{IntToByte}(\bullet)}$$

$$[\text{XC-ObjVal}]\frac{}{\texttt{object} <:_{\mathsf{x}} \gamma \rightsquigarrow \mathsf{Unbox}[\gamma](\bullet)} \qquad [\text{XC-ObjRef}]\frac{\rho \neq \texttt{dynamic}}{\texttt{object} <:_{\mathsf{x}} \rho \rightsquigarrow \mathsf{Downcast}[\rho](\bullet)}$$

$$[\text{XC-Down}]\frac{\texttt{C}_1\texttt{<}\overline{\sigma_1}\texttt{>} : \texttt{C}_2\texttt{<}\overline{\sigma_2}\texttt{>}}{\texttt{C}_2\texttt{<}\overline{\sigma_2}\texttt{>} <:_{\mathsf{x}} \texttt{C}_1\texttt{<}\overline{\sigma_1}\texttt{>} \rightsquigarrow \mathsf{Downcast}[\texttt{C}_1\texttt{<}\overline{\sigma_1}\texttt{>}](\bullet)} \qquad [\text{XC-IC}]\frac{\sigma_1 <:_{\mathsf{i}} \sigma_2 \rightsquigarrow C}{\sigma_1 <:_{\mathsf{x}} \sigma_2 \rightsquigarrow C}$$

Translation to $\mathrm{C}^{\#}{}_{\mathrm{CLR}}$ (Term Conversion 1/2)

Implicit Conversion (Expressions)

$$\text{[IC-Byte]}\ \frac{0 \leq \underline{i} \leq 255}{\Gamma \vdash \underline{i} <:_{\mathsf{i}} \texttt{byte} \leadsto \underline{i}} \qquad \text{[IC-Null]}\ \frac{}{\Gamma \vdash \texttt{null} <:_{\mathsf{i}} \rho \leadsto \texttt{null}}$$

$$\text{[IC-AME]}\ \frac{dtype(\texttt{D})(\overline{\sigma}) = \overline{\sigma_1} \to \sigma_2 \qquad \Gamma, \overline{x}{:}\,\overline{\sigma_1} \vdash \overline{s_1} <:_{\mathsf{i}} \sigma_2 \leadsto \overline{S_1}}{\Gamma \vdash \texttt{delegate(}\overline{\sigma_0}\ \overline{x}\texttt{)\{}\overline{s_1}\texttt{\}} <:_{\mathsf{i}} \texttt{D<}\overline{\sigma}\texttt{>} \leadsto \texttt{delegate(}\overline{\sigma_0}\ \overline{x}\texttt{)\{}\overline{S_1}\texttt{\}}}$$

$$\text{[IC-Synth]}\ \frac{\Gamma \vdash e_1 \uparrow \sigma_0 \leadsto E_1 \qquad \sigma_0 \neq \texttt{dynamic} \qquad \sigma_0 <:_{\mathsf{i}} \sigma_1 \leadsto C}{\Gamma \vdash e_1 <:_{\mathsf{i}} \sigma_1 \leadsto C[E_1]}$$

$$\text{[IC-Dynamic]}\ \frac{\Gamma \vdash e_1 \uparrow \texttt{dynamic} \leadsto E_1}{\Gamma \vdash e_1 <:_{\mathsf{i}} \sigma_1 \leadsto \mathsf{Convert}[\sigma_1](E_1{:}\,\texttt{dynamic})}$$

Explicit Conversion (Expressions)

$$\text{[XC-Synth]}\ \frac{\Gamma \vdash e_1 \uparrow \sigma_0 \leadsto E_1 \qquad \sigma_0 \neq \texttt{dynamic} \qquad \sigma_0 <:_{\mathsf{x}} \sigma_1 \leadsto C}{\Gamma \vdash e_1 <:_{\mathsf{x}} \sigma_1 \leadsto C[E_1]}$$

$$\text{[XC-Dynamic]}\ \frac{\Gamma \vdash e_1 \uparrow \texttt{dynamic} \leadsto E_1}{\Gamma \vdash e_1 <:_{\mathsf{x}} \sigma_1 \leadsto \mathsf{Convert}[\sigma_1](E_1{:}\,\texttt{dynamic})}$$

Translation to $\mathrm{C}^{\#}{}_{\mathrm{CLR}}$ (Term Conversion 2/2)

Implicit Conversion (Statements)

$$\text{[C-Skip]}\ \frac{}{\Gamma \vdash \texttt{;} <:_{\mathsf{i}} \sigma \rightsquigarrow \texttt{;}} \qquad \text{[C-ExpStatement]}\ \frac{\Gamma \vdash se_1 \uparrow \sigma_1 \rightsquigarrow SE_1}{\Gamma \vdash se_1\texttt{;} <:_{\mathsf{i}} \sigma \rightsquigarrow SE_1\texttt{;}}$$

$$\text{[C-Cond]}\ \frac{\Gamma \vdash e_1 <:_{\mathsf{i}} \texttt{bool} \rightsquigarrow E_1 \qquad \Gamma \vdash s_1 <:_{\mathsf{i}} \sigma \rightsquigarrow S_1 \qquad \Gamma \vdash s_2 <:_{\mathsf{i}} \sigma \rightsquigarrow S_2}{\Gamma \vdash \texttt{if}\ (e_1)\ s_1\ \texttt{else}\ s_2 <:_{\mathsf{i}} \sigma \rightsquigarrow \texttt{if}\ (E_1)\ S_1\ \texttt{else}\ S_2}$$

$$\text{[C-FAss]}\ \frac{\Gamma \vdash e_1 \uparrow \sigma_1 \rightsquigarrow E_1 \qquad \sigma_1 \neq \texttt{dynamic} \qquad \mathit{ftype}(\sigma_1, f) = \sigma_2 \qquad \Gamma \vdash e_2 <:_{\mathsf{i}} \sigma_2 \rightsquigarrow E_2}{\Gamma \vdash e_1\texttt{.}f\texttt{=}e_2\texttt{;} <:_{\mathsf{i}} \sigma \rightsquigarrow E_1\texttt{.}f\texttt{=}E_2\texttt{;}}$$

$$\text{[C-FAssDyn]}\ \frac{\Gamma \vdash e_1 \uparrow \texttt{dynamic} \rightsquigarrow E_1 \qquad \Gamma \vdash e_2 \uparrow^{+} \sigma_2 \rightsquigarrow E_2}{\Gamma \vdash e_1\texttt{.}f\texttt{=}e_2\texttt{;} <:_{\mathsf{i}} \sigma \rightsquigarrow \mathsf{Assign}[f](E_1 \colon \texttt{dynamic}, E_2 \colon \sigma_2)\texttt{;}}$$

$$\text{[C-ReturnExp]}\ \frac{\Gamma \vdash e_1 <:_{\mathsf{i}} \sigma \rightsquigarrow E_1}{\Gamma \vdash \texttt{return}\ e_1\texttt{;} <:_{\mathsf{i}} \sigma \rightsquigarrow \texttt{return}\ E_1\texttt{;}}$$

$$\text{[C-Seq]}\ \frac{\Gamma \vdash e_1 <:_{\mathsf{i}} \sigma_1 \rightsquigarrow E_1 \qquad x \notin \mathit{dom}(\Gamma) \qquad \Gamma, x \colon \sigma_1 \vdash \overline{s_1} <:_{\mathsf{i}} \sigma \rightsquigarrow \overline{S_1}}{\Gamma \vdash \sigma_1\ x\ \texttt{=}\ e_1\texttt{;}\ \overline{s_1} <:_{\mathsf{i}} \sigma \rightsquigarrow |\sigma_1|^{\star}\ x\ \texttt{=}\ E_1\texttt{;}\ \overline{S_1}}$$

Translation to $\mathrm{C}^{\#}{}_{\mathrm{CLR}}$ (Type Synthesis 1/2)

$$\text{[S-Bool]}\ \frac{}{\Gamma \vdash b \uparrow \texttt{bool} \leadsto b} \qquad \text{[S-Int]}\ \frac{}{\Gamma \vdash i \uparrow \texttt{int} \leadsto i} \qquad \text{[S-Var]}\ \frac{}{\Gamma, x{:}\,\sigma \vdash x \uparrow \sigma \leadsto x}$$

$$\text{[S-Cast]}\ \frac{\Gamma \vdash e_1 <:_{\mathsf{x}} \sigma_1 \leadsto E_1}{\Gamma \vdash (\sigma_1)e_1 \uparrow \sigma_1 \leadsto E_1}$$

$$\text{[S-Field]}\ \frac{\Gamma \vdash e_1 \uparrow \sigma_1 \leadsto E_1 \qquad \sigma_1 \neq \texttt{dynamic} \qquad \mathit{ftype}(\sigma_1, f) = \sigma_2}{\Gamma \vdash e_1\,.\,f \uparrow \sigma_2 \leadsto E_1\,.\,f}$$

$$\text{[S-DelInv]}\ \frac{\Gamma \vdash e_1 \uparrow \texttt{D<}\overline{\sigma}\texttt{>} \leadsto E_1 \qquad \mathit{dtype}(\texttt{D})(\overline{\sigma}) = \overline{\sigma_1} \to \sigma_2 \qquad \Gamma \vdash \overline{e_2} <:_{\mathsf{i}} \overline{\sigma_1} \leadsto \overline{E_2}}{\Gamma \vdash e_1(\overline{e_2}) \uparrow \sigma_2 \leadsto E_1(\overline{E_2})}$$

$$\text{[S-New]}\ \frac{\begin{array}{c} \mathrm{CMG} \stackrel{\text{def}}{=} \mathit{mtype}(C\texttt{<}\overline{\sigma}\texttt{>}, \texttt{.ctor}) \\ \mathrm{AMG} \stackrel{\text{def}}{=} \{\texttt{C<}\overline{X_C}\texttt{><}\overline{\sigma_C}\texttt{>::.ctor}{:}\,(\overline{\sigma_p}) \mid \\ \texttt{C<}\overline{X_C}\texttt{><}\overline{\sigma_C}\texttt{>::.ctor}{:}\,(\overline{\sigma_p}) \in \mathrm{CMG}, \\ |\overline{\sigma_p}| = |\overline{e_1}|, \Gamma \vdash \overline{e_1} <:_{\mathsf{i}} \overline{\sigma_p}[\overline{X_C} := \overline{\sigma_c}]\} \\ \Gamma \vdash \mathit{best}(\mathrm{AMG}, \overline{e_1}) \leadsto \mathit{md} = \texttt{C<}\overline{X_C}\texttt{><}\overline{\sigma_C}\texttt{>::.ctor}{:}\,(\overline{\sigma_p}) \\ \Gamma \vdash \overline{e_1} <:_{\mathsf{i}} \overline{\sigma_p}[\overline{X_C} := \overline{\sigma_C}] \leadsto \overline{E_1} \end{array}}{\Gamma \vdash \texttt{new C<}\overline{\sigma}\texttt{>(}\overline{e_1}\texttt{)} \uparrow \texttt{C<}\overline{\sigma}\texttt{>} \leadsto \texttt{new}\ |\mathit{md}|^{\star}(\overline{E_1})}$$

$$\text{[S-VarAssign]}\ \frac{\Gamma, x_1{:}\,\sigma_1 \vdash e_1 <:_{\mathsf{i}} \sigma_1 \leadsto E_1}{\Gamma, x_1{:}\,\sigma_1 \vdash x_1 \texttt{ = } e_1 \uparrow \sigma_1 \leadsto x_1 \texttt{ = } E_1}$$

$$\text{[S-MInv]}\ \frac{\begin{array}{c} \Gamma \vdash e_1 \uparrow \sigma \leadsto E_1 \qquad \sigma \neq \texttt{dynamic} \qquad \mathrm{CMG} \stackrel{\text{def}}{=} \mathit{mtype}(\sigma, m) \\ \mathrm{AMG} \stackrel{\text{def}}{=} \{\texttt{C<}\overline{X_C}\texttt{><}\overline{\sigma_C}\texttt{>::}m\texttt{<}\overline{X_m}\texttt{><}\overline{\sigma_1}\texttt{>}{:}\,(\overline{\sigma_p}) \to \sigma_r \mid \\ \texttt{C<}\overline{X_C}\texttt{><}\overline{\sigma_C}\texttt{>::}m\texttt{<}\overline{X_m}\texttt{>}{:}\,(\overline{\sigma_p}) \to \sigma_r \in \mathrm{CMG}, \\ |\overline{X_m}| = |\overline{\sigma_1}|,\ \Gamma \vdash \overline{e_2} <:_{\mathsf{i}} \overline{\sigma_p}[\overline{X_C} := \overline{\sigma_C}, \overline{X_m} := \overline{\sigma_1}]\} \\ \Gamma \vdash \mathit{best}(\mathrm{AMG}, \overline{e_2}) \leadsto \mathit{md} = \texttt{C<}\overline{X_C}\texttt{><}\overline{\sigma_C}\texttt{>::}m\texttt{<}\overline{X_m}\texttt{><}\overline{\sigma_1}\texttt{>}{:}\,(\overline{\sigma_p}) \to \sigma_r \\ \Gamma \vdash \overline{e_2} <:_{\mathsf{i}} \overline{\sigma_p}[\overline{X_C} := \overline{\sigma_C}, \overline{X_m} := \overline{\sigma_1}] \leadsto \overline{E_2} \end{array}}{\Gamma \vdash e_1\,.\,m\texttt{<}\overline{\sigma_1}\texttt{>(}\overline{e_2}\texttt{)} \uparrow \sigma_r[\overline{X_C} := \overline{\sigma_C}, \overline{X_m} := \overline{\sigma_1}] \leadsto E_1\,.\,|\mathit{md}|^{\star}(\overline{E_2})}$$

Translation to $\mathrm{C^{\#}}_{\mathrm{CLR}}$ (Type Synthesis 2/2)

$$[\text{S-FieldDyn}]\;\frac{\Gamma \vdash e_1 \uparrow \texttt{dynamic} \rightsquigarrow E_1}{\Gamma \vdash e_1\texttt{.}f \uparrow \texttt{dynamic} \rightsquigarrow \mathsf{MemberAccess}[f](E_1\colon \texttt{dynamic})}$$

$$[\text{S-DInvDyn1}]\;\frac{\begin{array}{cc}\Gamma \vdash e \uparrow \texttt{D<}\overline{\sigma}\texttt{>} \rightsquigarrow E & \mathit{dtype}(\texttt{D})(\overline{\sigma}) = \overline{\sigma_2} \to \sigma_3 \\ \Gamma \vdash e_1 \uparrow^{+} \sigma_1 \rightsquigarrow E_1 \;\cdots\; \Gamma \vdash e_n \uparrow^{+} \sigma_n \rightsquigarrow E_n & \exists i.1 \le i \le n.\sigma_i = \texttt{dynamic}\end{array}}{\Gamma \vdash e\texttt{(}e_1, \ldots, e_n\texttt{)} \uparrow \texttt{dynamic} \rightsquigarrow \mathsf{DInvoke}(E\colon \texttt{D<}\overline{\sigma}\texttt{>}, (E_1\colon \sigma_1, \ldots, E_n\colon \sigma_n))}$$

$$[\text{S-DInvDyn2}]\;\frac{\Gamma \vdash e_1 \uparrow \texttt{dynamic} \rightsquigarrow E_1 \qquad \Gamma \vdash \overline{e_2} \uparrow^{+} \overline{\sigma_2} \rightsquigarrow \overline{E_2}}{\Gamma \vdash e_1\texttt{(}\overline{e_2}\texttt{)} \uparrow \texttt{dynamic} \rightsquigarrow \mathsf{DInvoke}(E_1\colon \texttt{dynamic}, \overline{E_2}\colon \overline{\sigma_2})}$$

$$[\text{S-NewDyn}]\;\frac{\begin{array}{c}\mathrm{CMG} \stackrel{\text{def}}{=} \mathit{mtype}(\texttt{C<}\overline{\sigma}\texttt{>}, \texttt{.ctor}) \\ \Gamma \vdash e_1 \uparrow^{+} \sigma_1 \rightsquigarrow E_1 \;\cdots\; \Gamma \vdash e_n \uparrow^{+} \sigma_n \rightsquigarrow E_n \qquad \exists j.1 \le j \le n.\sigma_j = \texttt{dynamic} \\ \mathrm{AMG} \stackrel{\text{def}}{=} \{\texttt{C<}\overline{X_C}\texttt{><}\overline{\sigma_C}\texttt{>::.ctor}\colon (\overline{\sigma'}) \mid \\ \texttt{C<}\overline{X_C}\texttt{><}\overline{\sigma_C}\texttt{>::.ctor}\colon (\overline{\sigma'}) \in \mathrm{CMG}, \\ |\overline{\sigma'}| = n, \Gamma \vdash \overline{e}_i <:_{\mathsf{i}} \overline{\sigma'}_i[\overline{X_C} := \overline{\sigma_c}]\; i \in 1..n\} \\ |\mathrm{AMG}| \ge 1\end{array}}{\Gamma \vdash \texttt{new C<}\overline{\sigma}\texttt{>(}e_1\texttt{,}\cdots\texttt{,}e_n\texttt{)} \uparrow \texttt{C<}\overline{\sigma}\texttt{>} \rightsquigarrow \mathsf{ObjectCreate}[\texttt{C<}\overline{\sigma}\texttt{>}](E_1\colon \sigma_1, \ldots, E_n\colon \sigma_n)}$$

$$[\text{S-MInvDyn1}]\;\frac{\Gamma \vdash e_1 \uparrow \texttt{dynamic} \rightsquigarrow E_1 \qquad \Gamma \vdash \overline{e_2} \uparrow^{+} \overline{\sigma} \rightsquigarrow \overline{E_2}}{\Gamma \vdash e_1\texttt{.}m\texttt{<}\overline{\sigma_1}\texttt{>(}\overline{e_2}\texttt{)} \uparrow \texttt{dynamic} \rightsquigarrow \mathsf{MInvoke}[m](E_1\colon \texttt{dynamic}, \overline{E_2}\colon \overline{\sigma})}$$

$$[\text{S-MInvDyn2}]\;\frac{\begin{array}{c}\Gamma \vdash e \uparrow \sigma \rightsquigarrow E \qquad \mathrm{CMG} \stackrel{\text{def}}{=} \mathit{mtype}(\sigma, m) \\ \Gamma \vdash e_1 \uparrow^{+} \sigma_1 \rightsquigarrow E_1 \;\cdots\; \Gamma \vdash e_n \uparrow^{+} \sigma_n \rightsquigarrow E_n \qquad \exists j.1 \le j \le n.\sigma_j = \texttt{dynamic} \\ \mathrm{AMG} \stackrel{\text{def}}{=} \{\texttt{C<}\overline{X_C}\texttt{><}\overline{\sigma_C}\texttt{>::}m\texttt{<}\overline{X_m}\texttt{><}\overline{\sigma_1}\texttt{>}\colon (\overline{\sigma'}) \to \sigma_r \mid \\ \texttt{C<}\overline{X_C}\texttt{><}\overline{\sigma_C}\texttt{>::}m\texttt{<}\overline{X_m}\texttt{>}\colon (\overline{\sigma'}) \to \sigma_r \in \mathrm{CMG}, \\ |\overline{X_m}| = |\overline{\sigma_1}|, |\overline{\sigma'}| = n,\; \Gamma \vdash \overline{e}_i <:_{\mathsf{i}} \overline{\sigma'}_i[\overline{X_C} := \overline{\sigma_C}, \overline{X_m} := \overline{\sigma_1}]\; i \in 1..n\} \\ |\mathrm{AMG}| \ge 1\end{array}}{\Gamma \vdash e\texttt{.}m\texttt{<}\overline{\sigma_1}\texttt{>(}e_1\texttt{,}\cdots\texttt{,}e_n\texttt{)} \uparrow \texttt{dynamic} \rightsquigarrow \mathsf{MInvoke}[m](E\colon \sigma, (E_1\colon \sigma_1, \cdots, E_n\colon \sigma_n))}$$

Big Picture

- Any type can be implicitly converted to dynamic
- dynamic converted to object during translation
- Compile-time knowledge (i.e. int literals) retained
- Appropriate run-time casts inserted (subclassing relation)
- Overload resolution uses run-time and compile-time knowledge
- Type preservation (hooray!)

Operational Semantics (1/2)

- Payload component $(r : \sigma)$ = run-time value, type
  - Look up type from context if $\sigma = \text{dynamic}$
  - Do type checking and synthesis

$$\text{[S-PayODyn]}\frac{}{\Gamma, o\colon \tau \vdash o\colon \texttt{dynamic} \uparrow \tau \rightsquigarrow o} \quad \text{[S-PayIntDyn]}\frac{}{\Gamma \vdash \underline{i}\colon \texttt{dynamic} \uparrow \texttt{int} \rightsquigarrow \underline{i}}$$

$$\text{[S-PayStatic]}\frac{\sigma \neq \texttt{dynamic}}{\Gamma \vdash o\colon \sigma \uparrow \sigma \rightsquigarrow o}$$

$$\text{[C-PayODyn]}\frac{\tau <:_{\text{i}} \sigma \rightsquigarrow C}{\Gamma, o\colon \tau \vdash o\colon \texttt{dynamic} <:_{\text{i}} \sigma \rightsquigarrow C[o]}$$

$$\text{[C-PayNullDyn]}\frac{}{\Gamma \vdash \texttt{null}\colon \texttt{dynamic} <:_{\text{i}} \rho \rightsquigarrow \texttt{null}}$$

$$\text{[C-PayIntDyn]}\frac{\texttt{int} <:_{\text{i}} \sigma \rightsquigarrow C}{\Gamma \vdash \underline{i}\colon \texttt{dynamic} <:_{\text{i}} \sigma \rightsquigarrow C[\underline{i}]} \quad \text{[C-PayIntLit]}\frac{1 \leq \underline{i} \leq 255}{\Gamma \vdash \underline{i}\colon \texttt{int}^{l} <:_{\text{i}} \texttt{byte} \rightsquigarrow \underline{i}}$$

$$\text{[C-PayStatic]}\frac{\sigma_1 \neq \texttt{dynamic} \qquad \sigma_1 <:_{\text{i}} \sigma_2 \rightsquigarrow C}{\Gamma \vdash r\colon \sigma_1 <:_{\text{i}} \sigma_2 \rightsquigarrow C[r]}$$

Operational Semantics (2/2)

- It all comes down to the CLR...

$$[\text{E-Convert}]\ \frac{|H| \vdash o\colon \sigma_1 <:_{\text{i}} \sigma_2 \rightsquigarrow E}{\langle H, ST, \mathsf{Convert}[\sigma_2](o\colon \sigma_1), FS\rangle \twoheadrightarrow \langle H, ST, E, FS\rangle}$$

$$[\text{E-DMemAcc}]\ \frac{|H| \vdash (o\colon \sigma)\,.\,f <:_{\text{i}} \texttt{object} \rightsquigarrow E}{\langle H, ST, \mathsf{MemberAccess}[f](o\colon \sigma), FS\rangle \twoheadrightarrow \langle H, ST, E, FS\rangle}$$

$$[\text{E-DNew}]\ \frac{|H| \vdash \texttt{new}\ C\texttt{<}\overline{\sigma}\texttt{>}\texttt{(}\overline{r\colon \sigma'}\texttt{)} <:_{\text{i}} \texttt{object} \rightsquigarrow E}{\langle H, ST, \mathsf{ObjectCreate}[C\texttt{<}\overline{\sigma}\texttt{>}](\overline{r\colon \sigma'}), FS\rangle \twoheadrightarrow \langle H, ST, E, FS\rangle}$$

$$[\text{E-DDelInv}]\ \frac{|H| \vdash o\colon \sigma\,\texttt{(}\overline{r\colon \sigma'}\texttt{)} <:_{\text{i}} \texttt{object} \rightsquigarrow E}{\langle H, ST, \mathsf{DInvoke}(o\colon \sigma, \overline{r\colon \sigma'}), FS\rangle \twoheadrightarrow \langle H, ST, E, FS\rangle}$$

$$[\text{E-DMethInv}]\ \frac{|H| \vdash (o\colon \sigma)\,.\,m\,\texttt{(}\overline{r\colon \sigma'}\texttt{)} <:_{\text{i}} \texttt{object} \rightsquigarrow E}{\langle H, ST, \mathsf{MInvoke}[m](o\colon \sigma, \overline{r\colon \sigma'}), FS\rangle \twoheadrightarrow \langle H, ST, E, FS\rangle}$$

$$[\text{E-DFAss}]\ \frac{|H| \vdash (o\colon \sigma)\,.\,f\texttt{=}(o'\colon \sigma') <:_{\text{i}} \texttt{object} \rightsquigarrow E}{\langle H, ST, \mathsf{Assign}[f](o\colon \sigma, o'\colon \sigma')\,\texttt{;}, FS\rangle \twoheadrightarrow \langle H, ST, E, FS\rangle}$$

Questions?