

Safer Unsafe Code for .Net

Pietro Ferrara

Ecole Polytechnique
Paris, France

Ca' Foscari University
Venice, Italy

Francesco Logozzo

Microsoft Research
Redmond, WA, USA

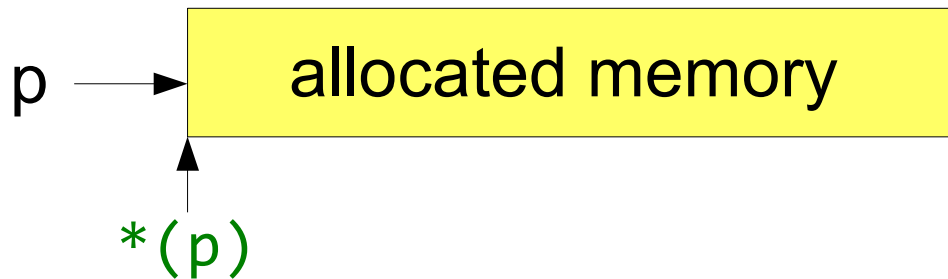
Manuel Fähndrich



OOPSLA 2008, Nashville, U.S.A.

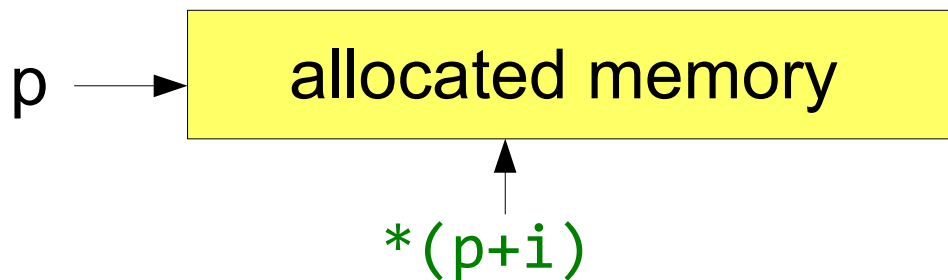
Background

- ▶ .Net: safe environment of execution
 - Exception: unsafe code
 - * **Direct access to the memory**
- ▶ Useful
 - Real-time and performance-critical applications
 - Interface with the operating system
- ▶ No guarantees on direct memory accesses
 - **Buffer overrun**



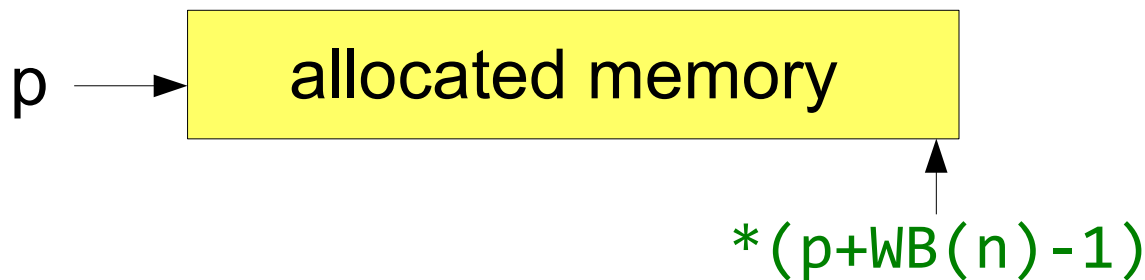
Background

- ▶ .Net: safe environment of execution
 - Exception: unsafe code
 - * **Direct access to the memory**
- ▶ Useful
 - Real-time and performance-critical applications
 - Interface with the operating system
- ▶ No guarantees on direct memory accesses
 - **Buffer overrun**



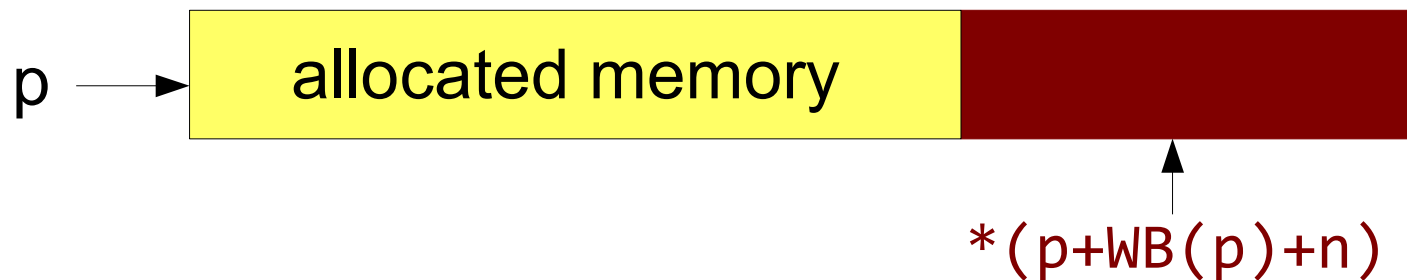
Background

- ▶ .Net: safe environment of execution
 - Exception: unsafe code
 - * Direct access to the memory
- ▶ Useful
 - Real-time and performance-critical applications
 - Interface with the operating system
- ▶ No guarantees on direct memory accesses
 - Buffer overrun



Background

- ▶ .Net: safe environment of execution
 - Exception: unsafe code
 - * Direct access to the memory
- ▶ Useful
 - Real-time and performance-critical applications
 - Interface with the operating system
- ▶ No guarantees on direct memory accesses
 - Buffer overrun



Our solution

- ▶ Language agnostic
- ▶ Static analysis to check buffer overruns
 - Based on abstract interpretation theory
 - Combination with contracts
 - * Pre and post conditions
 - * Class invariants
- ▶ Advantages
 - **Method boundary annotation**
 - * Automatically infer loop invariants
 - **Lightweight** domains
 - **Scalability**
 - * .Net libraries analyzed in a couple of minutes



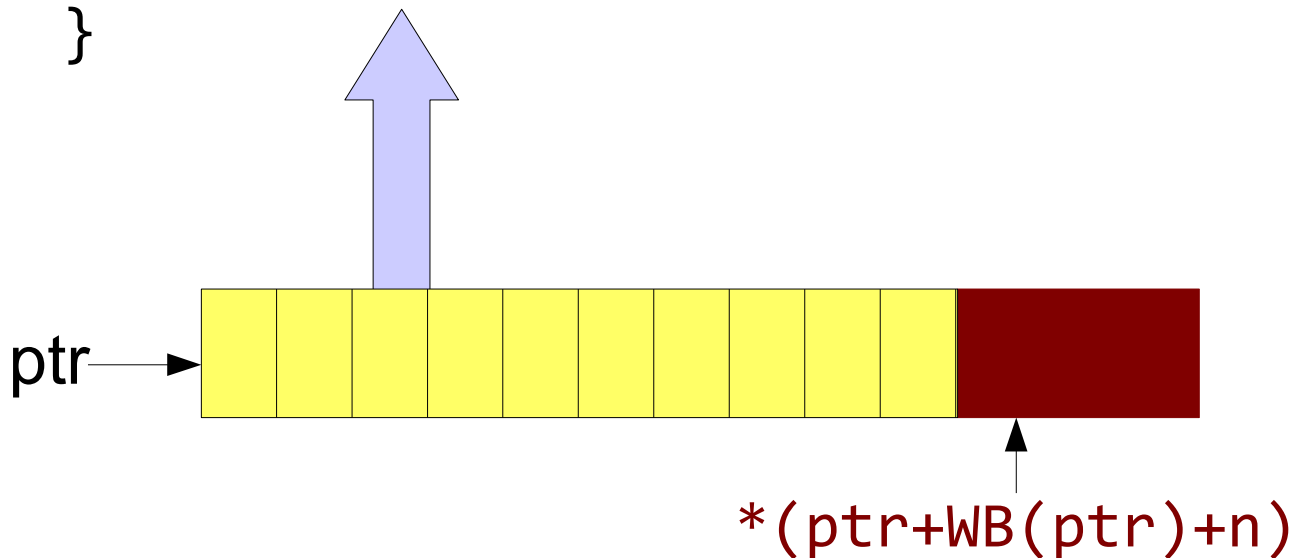
Example: InitToZero

```
unsafe void InitToZero(int* ptr, uint len)
{
    for (int i = 0; i < len; i++)
        *(ptr + i) = 0;
}
```



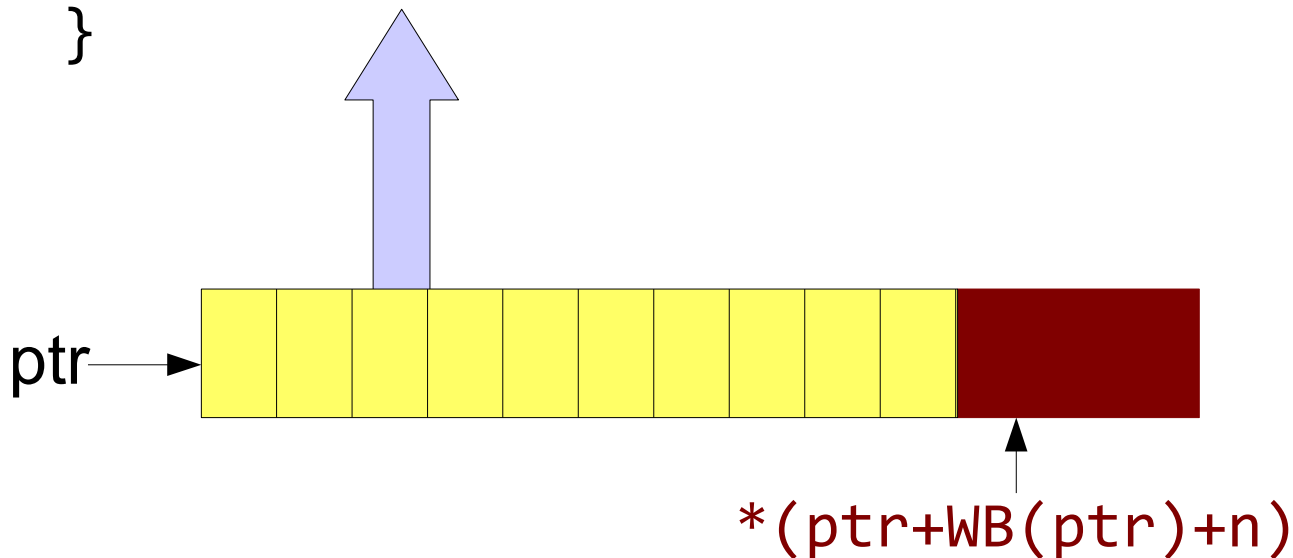
Example: InitToZero

```
unsafe void InitToZero(int* ptr, uint len)
{
    for (int i = 0; i < len; i++)
        *(ptr + i) = 0;
}
```



Example: InitToZero

```
unsafe void InitToZero(int* ptr, uint len)
{
    for (int i = 0; i < len; i++)
        *(ptr + i) = 0;
}
```



Precondition needed:

*“at least $Len * sizeof(int)$ bytes allocated starting from ptr ”*

Example: InitToZero

```
unsafe void InitToZero(int* ptr, uint len)
{
    Contract.Requires(Contract.WB(ptr) ≥ len*4);
    for (int i = 0; i < len; i++)
        *(ptr + i) = 0;
}
```

Infer: loop invariant $0 \leq i < \text{len}$



Specify precondition



Example: InitToZero

```
unsafe void InitToZero(int* ptr, uint len)
{
    Contract.Requires(Contract.WB(ptr) ≥ len*4);
    for (int i = 0; i < len; i++)
        *(ptr + i) = 0;
}
```

Infer: loop invariant $0 \leq i < \text{len}$



Specify precondition



Prove no buffer overrun



Example: InitToZero

```
unsafe void InitToZero(int* ptr, uint len)
{
    Contract.Requires(Contract.WB(ptr) ≥ len*4);
    for (int i = 0; i < len; i++)
        *(ptr + i) = 0;
}
```

```
unsafe void FastInitToZero(int[] arr)
{
    fixed (int* a = arr)
    {
        InitToZero(a, (uint) arr.Length);
    }
}
```



Precondition
satisfied

Foxtrot and Clousot

▶ Foxtrot

- Language-agnostic syntax for contracts
- Standard compilers (csc, vbc, ...)
- Enforceable at runtime and statically

▶ Clousot

- Static analyzer based on abstract interpretation
- Language agnostic
 - * Analyzes MSIL
- Generic
- Static checking of contracts



Abstract Interpretation

- ▶ Formalize static program analysis
- ▶ Idea of approximation
 - Different levels of abstraction
- ▶ Abstract semantics
 - **Rough enough** to be computable
 - **Precise enough** to capture the properties of interest
 - Can be derived from the concrete one
- ▶ Trade-off between precision and performances
- ▶ Two main components
 - **Transfer functions**
 - **Abstract domain**

Concrete Transfer Function

- ▶ Concrete state
 - Environment
 - Memory
 - Set of pinned variables

$$WB(p) \in \text{dom}(f), n = \text{eval}(\text{exp}, (f, g)), n \geq 0$$

$$f(WB(p)) \geq \text{sizeof}(x) + n * \text{sizeof}(*p)$$

$$g' = \text{write}(g, f(p) + n * \text{sizeof}(*p), \text{sizeof}(*p), f(x))$$

$$\mathbb{C}[\text{*}(p + \text{exp}) = x](f, g, t) \rightarrow (f, g', t)$$

$$WB(p) \notin \text{dom}(f) \vee \text{eval}(\text{exp}, (f, g)) < 0 \vee$$

$$f(WB(p)) < \text{sizeof}(x) + \text{eval}(\text{exp}, (f, g)) * \text{sizeof}(*p)$$

$$\mathbb{C}[\text{*}(p + \text{exp}) = x](f, g, t) \rightarrow \Omega$$

Abstract Transfer Function

- ▶ Abstract away values accessed through direct pointers
- ▶ **check**: numerical abstract domain primitive
- ▶ Approximates linear inequalities

check($\text{WB}(p) \geq \text{sizeof}(x) + \text{exp} * \text{sizeof}(*p)$, \bar{s}) = true
check($\text{exp} \geq 0$, \bar{s}) = true

$\mathbb{F}[\text{*(p + exp) = x}](\bar{s}) \rightarrow \bar{s}$

check($\text{WB}(p) \geq \text{sizeof}(x) + \text{exp} * \text{sizeof}(*p)$, \bar{s}) = \top
 \vee **check**($\text{exp} \geq 0$, \bar{s}) = \top

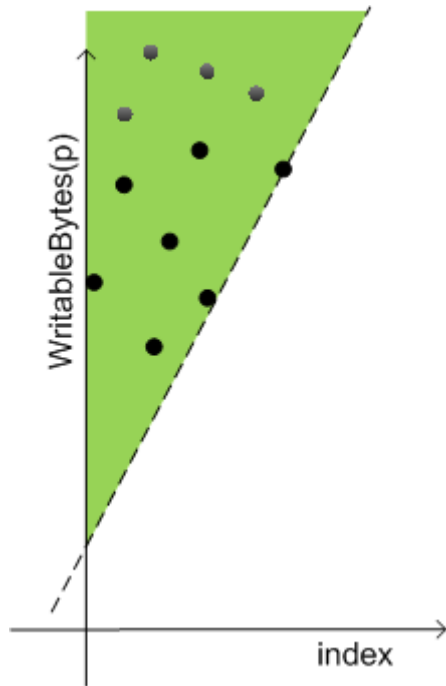
$\mathbb{F}[\text{*(p + exp) = x}](\bar{s}) \rightarrow \Omega?$



Domain Combination

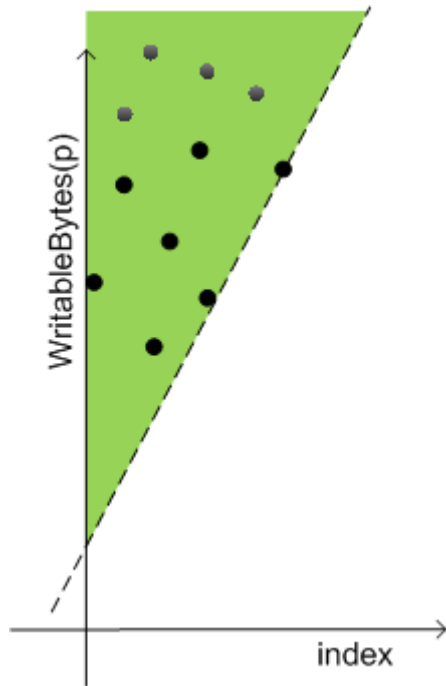
- ▶ Combine different domains
 - Lightweight
 - Focused
- ▶ Reduction
 - Flow of information
 - Mutual refinement
 - Well defined interface
- ▶ Three domains
 - Stripes (new relational domain)
 - Intervals
 - Linear equalities

Numerical Domains

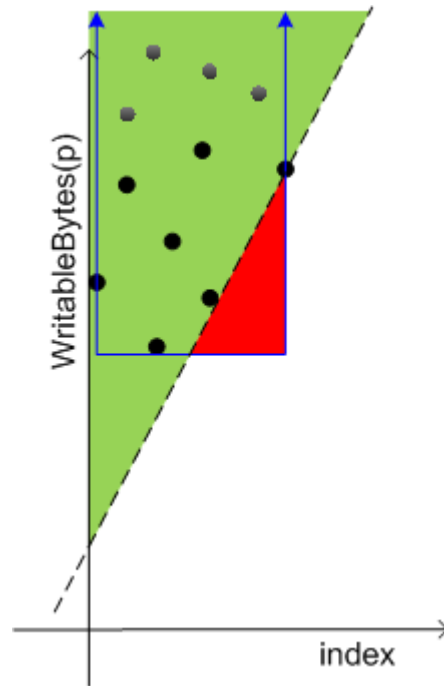


Concrete values

Numerical Domains

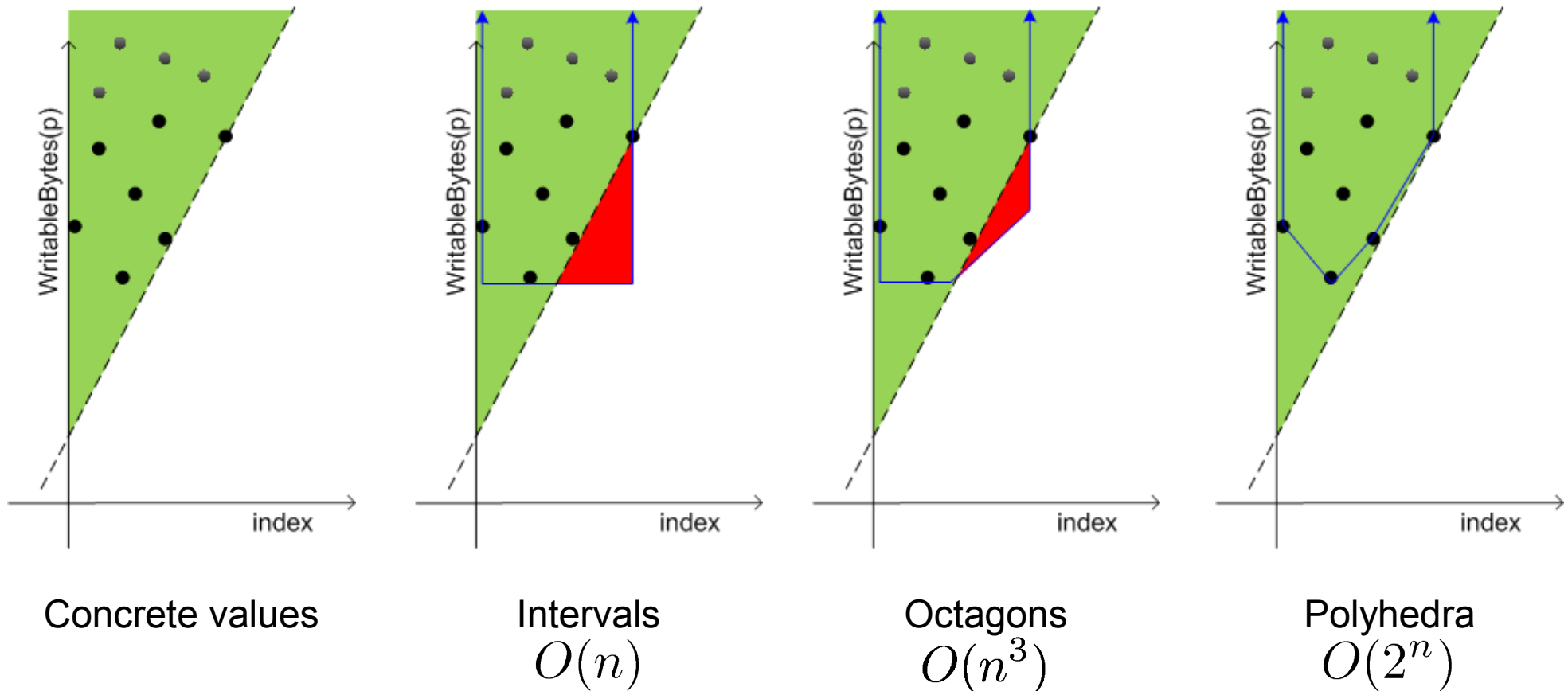


Concrete values



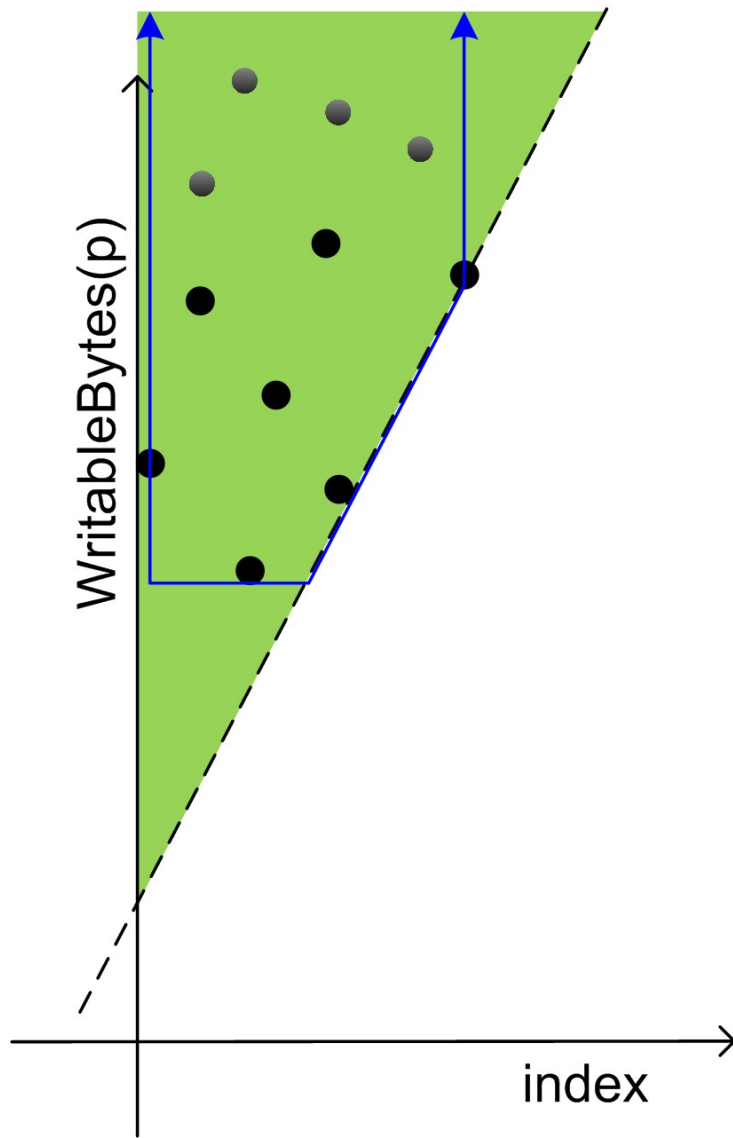
Intervals
 $O(n)$

Numerical Domains



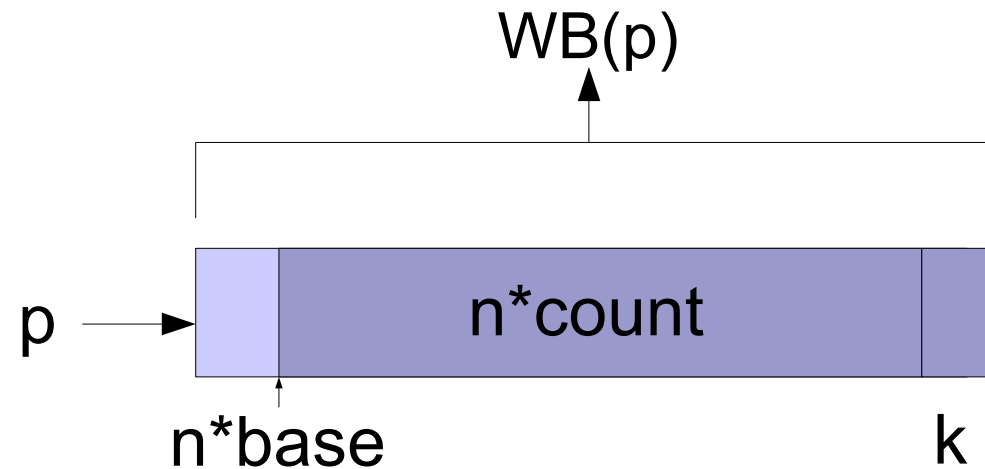
- ▶ Polyhedra: only monolithic domain **precise enough**
 - **Exponential complexity** in practice

Numerical Domains - Stripes



$$WB(p) \geq n * (\text{count}[+base]) + k$$

- Relational
- Precise



- ▶ **Linear complexity** in practice

Example: InitToZero

$\text{check}(\text{WB}(p) \geq \text{sizeof}(x) + \text{exp} * \text{sizeof}(*p), \bar{s}) = \text{true}$
 $\text{check}(\text{exp} \geq 0, \bar{s}) = \text{true}$

$\mathbb{F}[\text{*(p + exp) = x}](\bar{s}) \rightarrow \bar{s}$

```
unsafe void InitToZero(int* ptr, uint len)
{
    Contract.Requires(Contract.WB(ptr) ≥ len*4);
    for (int i = 0; i < len; i++)
        *(ptr + i) = 0;
}
```

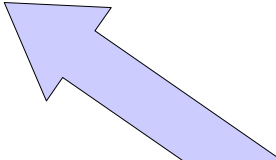


Example: InitToZero

$\text{check}(\text{WB}(p) \geq \text{sizeof}(x) + \text{exp} * \text{sizeof}(*p), \bar{s}) = \text{true}$
 $\text{check}(\text{exp} \geq 0, \bar{s}) = \text{true}$

$\mathbb{F}[\![*(p + \text{exp}) = x]\!](\bar{s}) \rightarrow \bar{s}$

```
unsafe void InitToZero(int* ptr, uint len)
{
    Contract.Requires(Contract.WB(ptr) ≥ len*4);
    for (int i = 0; i < len; i++)
        *(ptr + i) = 0;
}
```



Infer:
 $\text{WB}(ptr) \geq 4 * \text{len}$
 $\text{len} \geq i + 1$
 $\text{WB}(ptr) \geq 4 * i + 4$

Example: InitToZero

$\text{check}(\text{WB}(p) \geq \text{sizeof}(x) + \text{exp} * \text{sizeof}(*p), \bar{s}) = \text{true}$
 $\text{check}(\text{exp} \geq 0, \bar{s}) = \text{true}$

$\mathbb{F}[\text{*(p + exp) = x}](\bar{s}) \rightarrow \bar{s}$

```
unsafe void InitToZero(int* ptr, uint len)
{
    Contract.Requires(Contract.WB(ptr) ≥ len*4);
    for (int i = 0; i < len; i++)
        *(ptr + i) = 0;
}
```

Proof obligation:
 $\text{WB}(ptr) \geq 4 + i * 4$

Infer:
 $\text{WB}(ptr) \geq 4 * \text{len}$
 $\text{len} \geq i + 1$
 $\text{WB}(ptr) \geq 4 * i + 4$

Example: InitToZero

$\text{check}(\text{WB}(p) \geq \text{sizeof}(x) + \text{exp} * \text{sizeof}(*p), \bar{s}) = \text{true}$
 $\text{check}(\text{exp} \geq 0, \bar{s}) = \text{true}$

$\mathbb{F}[\![* (p + \text{exp}) = x]\!](\bar{s}) \rightarrow \bar{s}$

```
unsafe void InitToZero(int* ptr, uint len)
{
    Contract.Requires(Contract.WB(ptr) ≥ len*4);
    for (int i = 0; i < len; i++)
        *(ptr + i) = 0;
}
```

Proof obligation:
 $\text{WB}(ptr) \geq 4 + i * 4$

Infer:
 $\text{WB}(ptr) \geq 4 * \text{len}$
 $\text{len} \geq i + 1$
 $\text{WB}(ptr) \geq 4 * i + 4$

Validate

Intervals

$\text{check}(\text{WB}(p) \geq \text{sizeof}(x) + \text{exp} * \text{sizeof}(*p), \bar{s}) = \text{true}$

$\text{check}(\text{exp} \geq 0, \bar{s}) = \text{true}$

$\mathbb{F}[\text{*(p + exp) = x}](\bar{s}) \rightarrow \bar{s}$

▶ `for (int i = 0; i < len; i++)`
 `*(ptr + i) = 0;`

$\text{WB}(\text{ptr}) \geq 4 * \text{len}$
 $\text{len} \geq i + 1$
 $\text{WB}(\text{ptr}) \geq 4 * i + 4$

▶ Stripes does not prove $i \geq 0$

Intervals

$\text{check}(\text{WB}(p) \geq \text{sizeof}(x) + \text{exp} * \text{sizeof}(*p), \bar{s}) = \text{true}$
 $\text{check}(\text{exp} \geq 0, \bar{s}) = \text{true}$

$\mathbb{F}[\![*(p + \text{exp}) = x]\!](\bar{s}) \rightarrow \bar{s}$

▶ `for (int i = 0; i < len; i++)`
 `*(ptr + i) = 0;`

$\text{WB}(\text{ptr}) \geq 4 * \text{len}$
 $\text{len} \geq i + 1$
 $\text{WB}(\text{ptr}) \geq 4 * i + 4$


▶ Stripes does not prove $i \geq 0$

`for (int i = 0; i < len; i++)`
 `*(ptr + i) = 0;`

$i = [0..+\infty]$

▶ Intervals do it!

C# fixed semantics

fixed(T* p=arr)  C# semantics

```
if(arr == null)
  p = null;
else if(arr.Length == 0)
  p = null;
else
  p = &arr[0];
```

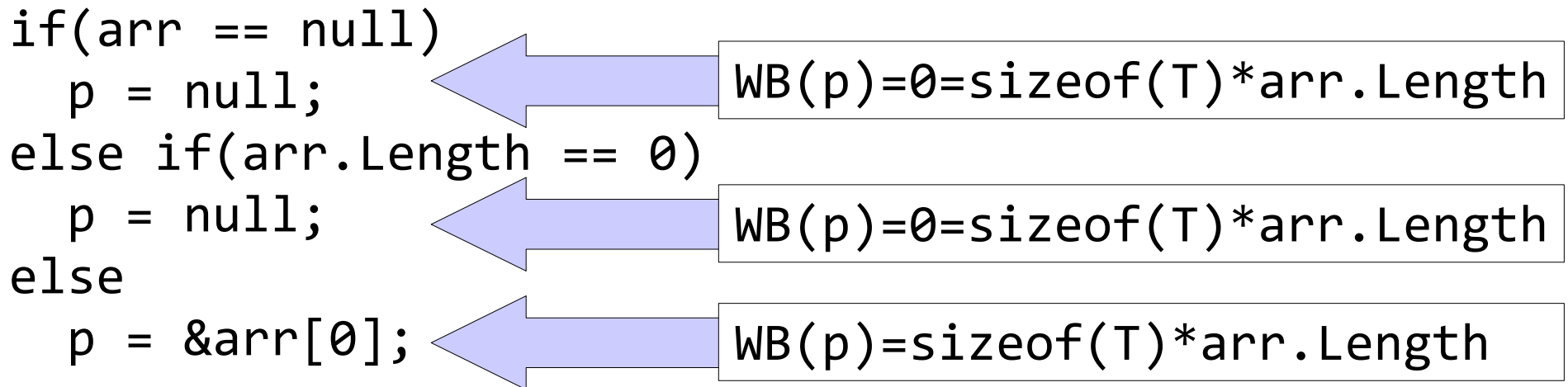
▶ C# fixed statement

- Set a pointer to a managed object
- “Pin” that object
 - * i.e. prevent the garbage collector from relocating it

▶ Intuition

- We want to infer $WB(p) = arr.Length * sizeof(T)$

Linear Equalities



- ▶ Upper bound: $WB(p) = \text{sizeof}(T) * \text{arr.Length}$

Reduced product

- ▶ Reduced product between Stripes, Intervals and LinEq

```
if(arr == null)
  p = null;
else if(arr.Length == 0)
  p = null;
else p = &arr[0];
```



Stripes: T
Intervals: T
LinEq: $WB(p) = \text{sizeof}(T) * \text{arr.Length}$

Reduced product

- ▶ Reduced product between Stripes, Intervals and LinEq

```
if(arr == null)
  p = null;
else if(arr.Length == 0)
  p = null;
else p = &arr[0];
```

Stripes: T
Intervals: T
LinEq: $WB(p) = \text{sizeof}(T) * \text{arr.Length}$

refine

Stripes: $WB(p) \geq \text{sizeof}(T) * \text{arr.Length}$
Intervals: T
LinEq: $WB(p) = \text{sizeof}(T) * \text{arr.Length}$

Experimental Results

Assembly	#Methods	Time	Checked	Val.	%
mscorlib.dll	18084	3m43s	3069	1835	59.79%
System.dll	13776	3m18s	1720	1048	60.93%
System.Data.dll	11333	3m45s	138	59	42.75%
System.Design.dll	11419	2m42s	16	10	62.50%
System.Drawing.dll	3120	19s	48	29	60.42%
System.Web.dll	22076	3m19s	88	44	50.00%
System.Windows.Forms.dll	23180	4m31s	364	266	73.08%
System.XML.dll	10046	2m41s	772	311	40.28%
Average					57.96%

- ▶ Analyzed shipped .Net framework assemblies
- ▶ **Scalable** analysis
- ▶ **Code not annotated**, false alarms

Experimental Results

Assembly	#Methods	Time	Checked	Val.	%
System.Drawing.dll	3120	19s	48	29	60,42%

- ▶ System.Drawing exposes warnings on 5 methods
 - 1) Lack of contracts
 - 2) Complicate invariant on data structure
 - 3) Marshalling not precisely analyzed by Clousot
 - 4) Shortcoming on the implementation
 - 5) **Bug**
 - Public method
 - It causes the crash at runtime

Conclusion

- ▶ New static analysis for checking buffer overrun in unsafe code in .Net
- ▶ Core of the analysis: Stripes (new domain)
- ▶ Refined with Intervals and Linear Equalities
- ▶ Applied successfully to the analysis of all .Net
- ▶ Precise and scalable

- ▶ **Ongoing work**
 - Write contracts on .Net libraries
 - * Reduce false alarms
 - * Find more bugs

