Especificación y verificación modular de consumo de memoria

Jonathan Tapicer

Departamento de Computación Facultad de Ciencias Exactas y Naturales Universidad de Buenos Aires

Febrero de 2011

Directores: Diego Garbervetsky, Martín Rouaux

Jonathan Tapker Deparament de Compunción Facabad de Cimcio Cosco y Sacardo Balandiad de Danas Alon

Februarie 2011

Directores: Diego Garbarostaly, Martin Rossax

- 1. Decir el título
- 2. Contar que vamos a empezar describiendo a grandes rasgos el objetivo del trabajo

Anotaciones Verificación Limitaciones y trabajo futuro Conclusione

Objetivo

Dados:

- Un programa escrito en un lenguaje orientado a objetos, y
- Un conjunto de contratos que especifican su consumo de memoria

Queremos:

- Verificar la correctitud de los contratos
- Para esto:
 - Diseñamos un lenguaje de especificación y un conjunto de técnicas que realizan un análisis estático del programa, y
 - Construimos una herramienta que utiliza estas técnicas

 $\, \, \bigsqcup_{\mathsf{O}\,\mathsf{bjetivo}} \,$

Tesis de Licenciatura



- 1. Tenemos un programa en un lenguaje orientado a objetos
- 2. El usuario nos dice de alguna forma su consumo de memoria
- 3. Lo verificamos, hacemos algoritmos y una implementación

- Introducción
- 2 Anotaciones
- 3 Verificación
- 4 Limitaciones y trabajo futuro
- 5 Conclusiones

└─Índice

Tesis de Licenciatura



- 1. Cómo se va a estructurar la presentación
- 2. Introducción: discutimos alternativas, damos un pantallazo de la solución
- 3. Anotaciones: algunos conceptos preliminares para introducir las anotaciones, cómo permitimos al usuario darnos los contratos de consumo
- 4. Verificación: cómo verificamos que los contratos son correctos
- 5. Limitaciones y trabajo futuro: contamos limitaciones que encontramos, decimos cómo se puede seguir trabajando
- 6. Conclusiones: resumimos todo lo presentado

Índice

- Introducción
- 2 Anotaciones
- 3 Verificación
- 4 Limitaciones y trabajo futuro
- 5 Conclusiones

- ¿Por qué analizar el consumo de memoria?
 - Sistemas con requerimientos estrictos de memoria: tiempo real, embebidos, misión crítica
 - Entornos donde se cobra el uso de recursos (cloud)
 - Es valioso contar con un certificado de la memoria requerida para una ejecución segura
- Inferir vs. Verificar
 - Análisis de consumo de memoria en general: es indecidible
 - Verificar es "más fácil" que inferir, analogía del laberinto

 - La combinación usuario + verificación puede ser mejor que la inferencia
- Verificación de consumo de memoria: área poco explorada

- Sistemas donde se cobra: no queremos pagar de más, pero hay que saber cuánto consume
- 2. Contar diferencias entre inferir y verificar
- 3. Queremos conocer la memoria necesaria para que el programa se ejecute de forma segura
- 4. Laberinto: difícil de resolver, fácil de verificar solución
- 5. Podemos pensar que tenemos un problema (saber el consumo, como el laberinto) y usuario nos da la solución, tenemos que verificarla
- 6. Existen para inferir: Diego, Martín, extensión de Gastón y Matías
- 7. Existen para verificar, implementaciones: JML, Spec#, Code Contracts

Verificación

- ¿Por qué analizar el consumo de memoria?
 - Sistemas con requerimientos estrictos de memoria: tiempo real, embebidos, misión crítica
 - Entornos donde se cobra el uso de recursos (cloud)
 - Es valioso contar con un certificado de la memoria requerida para una ejecución segura
- Inferir vs. Verificar
 - Análisis de consumo de memoria en general: es indecidible
 - Verificar es "más fácil" que inferir, analogía del laberinto
 - El usuario razona, tiene un conocimiento profundo del programa
 - La combinación usuario + verificación puede ser mejor que la inferencia
- Verificación de consumo de memoria: área poco explorada

- 1. Sistemas donde se cobra: no queremos pagar de más, pero hay que saber cuánto consume
- 2. Contar diferencias entre inferir y verificar
- 3. Queremos conocer la memoria necesaria para que el programa se ejecute de forma segura
- 4. Laberinto: difícil de resolver, fácil de verificar solución
- 5. Podemos pensar que tenemos un problema (saber el consumo, como el laberinto) y usuario nos da la solución, tenemos que verificarla
- 6. Existen para inferir: Diego, Martín, extensión de Gastón y Matías
- 7. Existen para verificar, implementaciones: JML, Spec#, Code Contracts

Verificación

- ¿Por qué analizar el consumo de memoria?
 - Sistemas con requerimientos estrictos de memoria: tiempo real, embebidos, misión crítica
 - Entornos donde se cobra el uso de recursos (cloud)
 - Es valioso contar con un certificado de la memoria requerida para una ejecución segura
- Inferir vs. Verificar
 - Análisis de consumo de memoria en general: es indecidible
 - Verificar es "más fácil" que inferir, analogía del laberinto
 - El usuario razona, tiene un conocimiento profundo del programa
 - La combinación usuario + verificación puede ser mejor que la inferencia
- Verificación de consumo de memoria: área poco explorada

a [For (a) a radion of constant or we want)

a from our expensions continued a month's dimps and,

a from the first with one for commelted property of the constant of the con

Ve rifi cació n

- 1. Sistemas donde se cobra: no queremos pagar de más, pero hay que saber cuánto consume
- 2. Contar diferencias entre inferir y verificar
- 3. Queremos conocer la memoria necesaria para que el programa se ejecute de forma segura
- 4. Laberinto: difícil de resolver, fácil de verificar solución
- 5. Podemos pensar que tenemos un problema (saber el consumo, como el laberinto) y usuario nos da la solución, tenemos que verificarla
- 6. Existen para inferir: Diego, Martín, extensión de Gastón y Matías
- 7. Existen para verificar, implementaciones: JML, Spec#, Code Contracts

Overview del trabajo (1)

- Proponemos un lenguaje de especificación para el consumo de memoria
 - Anotaciones embebidas en el código (à la Code Contracts)
 - Pensado como una extensión natural de los contratos de Code Contracts
 - Modular y expresivo (tiempos de vida)

Especificación de contratos con Code Contracts:

```
public double RaizCuadrada(double n)
2
3
    Contract.Requires(n >= 0);
     Contract.Ensures(Contract.Result() * Contract.Result() == n);
     Contract Ensures(Contract Result() >= 0);
6
     return Math.Sqrt(n);
```



- 1. Contamos el desarrollo del trabajo, detallamos cada parte más adelante
- 2. Usar el estilo de Code Contracts es bueno porque: fácil de escribir (no hace falta aprender nueva sintaxis), integrado a la IDE
- 3. Lenguaje: modular, a nivel método, de un método sólo se ven sus contratos de consumo, no es necesario conocer su implementación
- 4. Lenguaje: expresivo, tiene en cuenta que los objetos creados por un método pueden tener diferentes tiempos de vida

Introducción

- 2 Proponemos un conjunto de técnicas para verificar la correctitud de las anotaciones
 - Apoyándose en un verificador estático
 - Usando técnicas de análisis de tiempo de vida
 - Teniendo en cuenta la necesidad de verificar aritmética no lineal
- 3 Construimos un prototipo de la solución propuesta para .NET extendiendo Code Contracts
 - Buena integración con la IDE (Visual Studio)
 - Disponibilidad de herramientas para análisis estático e instrumentación
 - Verificador estático que requiere poca asistencia (uso de Interpretación Abstracta)

- 1. Herramientas para análisis estático: CCI
- 2. A nivel CIL (\sim bytecode): permite usar la herramienta en cualquier lenguaje de .NET: C#, VB, C++, IronPython, IronRuby
- 3. Nosotros probamos (y mostramos ejemplos) sólo con C#

Overview del trabajo (2)

Introducción

- 2 Proponemos un conjunto de técnicas para verificar la correctitud de las anotaciones
 - Apoyándose en un verificador estático
 - Usando técnicas de análisis de tiempo de vida
 - Teniendo en cuenta la necesidad de verificar aritmética no lineal
- 3 Construimos un prototipo de la solución propuesta para .NET extendiendo Code Contracts
 - Buena integración con la IDE (Visual Studio)
 - Disponibilidad de herramientas para análisis estático e instrumentación
 - Verificador estático que requiere poca asistencia (uso de Interpretación Abstracta)

Introducción -Overview del trabajo (2) 🔁 Proponemos un conjunto de técnicas para verifica e la correcticad de las · Apople from er an conficulto contains . Unan de etenicas de antificio de ciempo de cida

. Teriodo er como la receitad de ceitor arienteira er freal 🛂 Construimos un prototipo de la soleción propuesta para "NET

Overview del trabajo (2)

extendiendo Code Contracto

Bons integración con la IDE (Maral Scotio)

. Disposibilitad de homamientas para an Meir eschice e inscrementacido . Verili culter encluice que requiere poca animercia (a se de le sepresacido Abstractal

- 1. Herramientas para análisis estático: CCI
- 2. A nivel CIL (~bytecode): permite usar la herramienta en cualquier lenguaje de .NET: C#, VB, C++, IronPython, IronRuby
- 3. Nosotros probamos (y mostramos ejemplos) sólo con C#

- Introducción
- 2 Anotaciones
- 3 Verificación
- 4 Limitaciones y trabajo futuro
- 5 Conclusiones

Tesis de Licenciatura

8 / 30

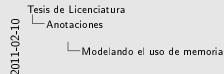
- Los lenguajes orientados a objetos modernos usan un GC para administrar la memoria, impredecible
- Vamos a usar un modelo del consumo de memoria predecible
 - Sobreaproxima los requerimientos reales, permitiendo obtener una cota del consumo
 - Asumimos que los objetos son recolectados al final del método
 - Sólo tiene en cuenta objetos creados por el método analizado y los métodos que este invoca
 - Objetos temporales: no necesarios después de la ejecución del método, se pueden eliminar de memoria
 - Objetos residuales: exceden el tiempo de vida del método, deben seguir vivos luego de que finalice la ejecución del método

- 1. Necesitamos modelar el uso de memoria de una forma donde la alocación y liberación sea predecible
- 2. Obtenemos una cota superior: los GC son mejores (más agresivos) que al final del método, por eso sobreaproximamos
- 3. Partimos al conjunto de objetos creados por un método en dos: tmp y rsd
- 4. EXPLICAR RÁPIDO Y DETALLAR EN EL EJEMPLO

2011-02-10

- Temporales: objetos usados para un cálculo o proceso auxiliar durante la ejecución
- 6. Residuales: de diferentes formas (devuelto, parámetros, globales)

- Los lenguajes orientados a objetos modernos usan un GC para administrar la memoria, impredecible
- Vamos a usar un modelo del consumo de memoria predecible
 - Sobreaproxima los requerimientos reales, permitiendo obtener una cota del consumo
 - Asumimos que los objetos son recolectados al final del método
 - Sólo tiene en cuenta objetos creados por el método analizado y los métodos que este invoca
 - Objetos temporales: no necesarios después de la ejecución del método, se pueden eliminar de memoria
 - Objetos residuales: exceden el tiempo de vida del método, deben seguir vivos luego de que finalice la ejecución del método



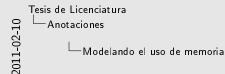
Lou lenguajes orientados a objetos modernos essense GC para administración memoris, impodecible
 Westerna a sasterna modelo del comuno de memoria padecible.

Modelando el uso de memoria

1. Necesitamos modelar el uso de memoria de una forma donde la alocación y liberación sea predecible

- 2. Obtenemos una cota superior: los GC son mejores (más agresivos) que al final del método, por eso sobreaproximamos
- 3. Partimos al conjunto de objetos creados por un método en dos: tmp y rsd
- 4. EXPLICAR RÁPIDO Y DETALLAR EN EL EJEMPLO
- Temporales: objetos usados para un cálculo o proceso auxiliar durante la ejecución
- 6. Residuales: de diferentes formas (devuelto, parámetros, globales)

- Los lenguajes orientados a objetos modernos usan un GC para administrar la memoria, impredecible
- Vamos a usar un modelo del consumo de memoria predecible
 - Sobreaproxima los requerimientos reales, permitiendo obtener una cota del consumo
 - Asumimos que los objetos son recolectados al final del método
 - Sólo tiene en cuenta objetos creados por el método analizado y los métodos que este invoca
 - Objetos temporales: no necesarios después de la ejecución del método, se pueden eliminar de memoria
 - Objetos residuales: exceden el tiempo de vida del método, deben seguir vivos luego de que finalice la ejecución del método



Los lenguijes odiestados a objetos modernos asses as GC para ad ministra la memoia, impedestible "Vermas asses ar modelo del coma mode memoia padestible "Sabragansima ha regenimiento a las permisinado absenta as casa.

Modelando el uso de memoria

1. Necesitamos modelar el uso de memoria de una forma donde la alocación y liberación sea predecible

- 2. Obtenemos una cota superior: los GC son mejores (más agresivos) que al final del método, por eso sobreaproximamos
- 3. Partimos al conjunto de objetos creados por un método en dos: tmp y rsd
- 4. EXPLICAR RÁPIDO Y DETALLAR EN EL EJEMPLO
- 5. Temporales: objetos usados para un cálculo o proceso auxiliar durante la ejecución
- 6. Residuales: de diferentes formas (devuelto, parámetros, globales)

- Los lenguajes orientados a objetos modernos usan un GC para administrar la memoria, impredecible
- Vamos a usar un modelo del consumo de memoria predecible
 - Sobreaproxima los requerimientos reales, permitiendo obtener una cota del consumo
 - Asumimos que los objetos son recolectados al final del método
 - Sólo tiene en cuenta objetos creados por el método analizado y los métodos que este invoca
 - Objetos temporales: no necesarios después de la ejecución del método, se pueden eliminar de memoria
 - Objetos residuales: exceden el tiempo de vida del método, deben seguir vivos luego de que finalice la ejecución del método

- 1. Necesitamos modelar el uso de memoria de una forma donde la alocación y liberación sea predecible
- 2. Obtenemos una cota superior: los GC son mejores (más agresivos) que al final del método, por eso sobreaproximamos
- 3. Partimos al conjunto de objetos creados por un método en dos: tmp y rsd
- 4. EXPLICAR RÁPIDO Y DETALLAR EN EL EJEMPLO
- 5. Temporales: objetos usados para un cálculo o proceso auxiliar durante la ejecución
- 6. Residuales: de diferentes formas (devuelto, parámetros, globales)

- Los lenguajes orientados a objetos modernos usan un GC para administrar la memoria, impredecible
- Vamos a usar un modelo del consumo de memoria predecible
 - Sobreaproxima los requerimientos reales, permitiendo obtener una cota del consumo
 - Asumimos que los objetos son recolectados al final del método
 - Sólo tiene en cuenta objetos creados por el método analizado y los métodos que este invoca
 - Objetos temporales: no necesarios después de la ejecución del método, se pueden eliminar de memoria
 - Objetos residuales: exceden el tiempo de vida del método, deben seguir vivos luego de que finalice la ejecución del método

Venora carrar motalo tel comuno te menola patecida
 Sabragacina la regresimiente natos, penicinale alcone na cara
falca cama.

Assertions que les objects son recollecte des al final del mitorion
 Sitte cione en contra abjects certa des par el mitorio analizado y los

1. Necesitamos modelar el uso de memoria de una forma donde la alocación y liberación sea predecible

- 2. Obtenemos una cota superior: los GC son mejores (más agresivos) que al final del método, por eso sobreaproximamos
- 3. Partimos al conjunto de objetos creados por un método en dos: tmp y rsd
- 4. EXPLICAR RÁPIDO Y DETALLAR EN EL EJEMPLO
- Temporales: objetos usados para un cálculo o proceso auxiliar durante la ejecución
- 6. Residuales: de diferentes formas (devuelto, parámetros, globales)

- Los lenguajes orientados a objetos modernos usan un GC para administrar la memoria, impredecible
- Vamos a usar un modelo del consumo de memoria predecible
 - Sobreaproxima los requerimientos reales, permitiendo obtener una cota del consumo
 - Asumimos que los objetos son recolectados al final del método
 - Sólo tiene en cuenta objetos creados por el método analizado y los métodos que este invoca
 - Objetos temporales: no necesarios después de la ejecución del método, se pueden eliminar de memoria
 - Objetos residuales: exceden el tiempo de vida del método, deben seguir vivos luego de que finalice la ejecución del método

Modelando el uso de memoria

- Los lenguajes orientados a objetos modernos asan an GC para
 ad ministrar la memoria, impreducible
- Valencia and an establish common de memoria partecible
 Sabragani ina la conquerimi in un make, permitin de abende ara condella common.
- Asserting the landing and produce do all hall delimited a
 - Silla cione en cuenca abjecto crea dos per el mitodo analizado y los metodos que este inoca
 - Objects on paralex on necessitis despuis de la ejecución del mando, se parades eliminas de memoria
 Objects espirades secretar el rienzo de ejecución del mando, deber secrificados.
 - direktop te ga trilke kapendir tellende
- 1. Necesitamos modelar el uso de memoria de una forma donde la alocación y liberación sea predecible
- 2. Obtenemos una cota superior: los GC son mejores (más agresivos) que al final del método, por eso sobreaproximamos
- 3. Partimos al conjunto de objetos creados por un método en dos: tmp y rsd
- 4. EXPLICAR RÁPIDO Y DETALLAR EN EL EJEMPLO
- 5. Temporales: objetos usados para un cálculo o proceso auxiliar durante la ejecución
- 6. Residuales: de diferentes formas (devuelto, parámetros, globales)

Modelando el uso de memoria, ejemplo

```
class IntLinkedList
2
3
       private Node Head;
4
5
       public void PushFront(Node node)
6
7
          Logger logger = new Logger();
8
          node. Next = this. Head;
9
          this. Head = node:
10
          logger.Log("PushFront done");
11
12
13
       public void Fill(int n)
14
15
          for (int i = 1; i <= n; i++)
16
17
              Node node = new Node(i):
18
              this. PushFront(node);
19
       }
20
21
```

Tesis de Licenciatura

10 / 30

- 1. Explicar código
- 2. Decir que lo vamos a usar de ejemplo en el resto de la presentación
- 3. Explicar por qué son tmp y rsd cada uno

-Modelando el uso de memoria, ejemplo

Modelando el uso de memoria, ejemplo

```
class IntLinkedList
3
       private Node Head;
4
5
       public void PushFront(Node node)
6
7
          Logger logger = new Logger();
                                                       tempora
8
          node. Next = this. Head;
9
          this. Head = node:
10
          logger.Log("PushFront done");
11
12
13
       public void Fill(int n)
14
15
          for (int i = 1; i <= n; i++)
16
                                                      residuales
17
              Node node = new Node(i);
18
              this. PushFront(node);
19
20
21
```

-Modelando el uso de memoria, ejemplo



- 1. Explicar código
- 2. Decir que lo vamos a usar de ejemplo en el resto de la presentación
- 3. Explicar por qué son tmp y rsd cada uno

```
public void PushFront(Node node)
  Logger logger = new Logger();
  node.Next = this.Head;
   this. Head = node;
  logger.Log("PushFront done");
}
public void Fill(int n)
  for (int i = 1; i <= n; i++)</pre>
  {
     Node node = new Node(i);
     this. PushFront(node);
```



- 1. Mostrar de a poco y explicar cada anotación
- 2. Vean cómo se parecen a las de Code Contracts
- 3. Explicar que Fill tiene tmp porque para la ejecución del método es necesario el espacio de memoria de un Logger
- 4. Para que el contrato sea correcto n tiene que ser positivo

```
public void PushFront(Node node)
  Contract.Memory.Tmp<Logger>(1); ◀
  Logger logger = new Logger();
  node.Next = this.Head:
  this. Head = node;
  logger.Log("PushFront done");
}
public void Fill(int n)
  for (int i = 1; i <= n; i++)
  {
     Node node = new Node(i);
     this. PushFront(node);
```



- 1. Mostrar de a poco y explicar cada anotación
- 2. Vean cómo se parecen a las de Code Contracts
- 3. Explicar que Fill tiene tmp porque para la ejecución del método es necesario el espacio de memoria de un Logger
- 4. Para que el contrato sea correcto n tiene que ser positivo

```
public void PushFront(Node node)
  Contract Memory Tmp<Logger>(1);
  Contract Memory DestTmp(); 
  Logger logger = new Logger();
  node.Next = this.Head:
  this. Head = node;
  logger.Log("PushFront done");
}
public void Fill(int n)
  for (int i = 1; i <= n; i++)
  {
     Node node = new Node(i);
     this. PushFront(node);
```



- 1. Mostrar de a poco y explicar cada anotación
- 2. Vean cómo se parecen a las de Code Contracts
- 3. Explicar que Fill tiene tmp porque para la ejecución del método es necesario el espacio de memoria de un Logger
- 4. Para que el contrato sea correcto n tiene que ser positivo

```
public void PushFront(Node node)
  Contract Memory Tmp<Logger>(1);
  Contract.Memory.DestTmp();
  Logger logger = new Logger();
  node.Next = this.Head:
  this. Head = node;
  logger.Log("PushFront done");
}
public void Fill(int n)
  Contract Memory Rsd<Node>(Contract Memory This, n); ◀
  for (int i = 1; i <= n; i++)
  {
     Node node = new Node(i);
     this. PushFront(node);
```



- 1. Mostrar de a poco y explicar cada anotación
- 2. Vean cómo se parecen a las de Code Contracts
- 3. Explicar que Fill tiene tmp porque para la ejecución del método es necesario el espacio de memoria de un Logger
- 4. Para que el contrato sea correcto n tiene que ser positivo

```
public void PushFront(Node node)
  Contract Memory Tmp<Logger>(1);
  Contract.Memory.DestTmp();
  Logger logger = new Logger();
  node.Next = this.Head:
   this. Head = node;
   logger.Log("PushFront done");
}
public void Fill(int n)
  Contract Memory Rsd<Node>(Contract Memory This, n);
   Contract Memory Tmp < Logger > (1); ◀
  for (int i = 1; i <= n; i++)</pre>
  {
     Node node = new Node(i);
     this. PushFront(node);
```



- 1. Mostrar de a poco y explicar cada anotación
- 2. Vean cómo se parecen a las de Code Contracts
- 3. Explicar que Fill tiene tmp porque para la ejecución del método es necesario el espacio de memoria de un Logger
- 4. Para que el contrato sea correcto n tiene que ser positivo

```
public void PushFront(Node node)
  Contract Memory Tmp<Logger>(1);
  Contract Memory DestImp();
  Logger logger = new Logger();
  node.Next = this.Head:
  this. Head = node;
  logger.Log("PushFront done");
}
public void Fill(int n)
  Contract Memory Rsd<Node>(Contract Memory This, n);
  Contract.Memory.Tmp<Logger>(1);
  for (int i = 1; i <= n; i++)
  {
     Contract Memory DestRsd(Contract Memory This); ◀
     Node node = new Node(i);
     this. PushFront(node);
```



- 1. Mostrar de a poco y explicar cada anotación
- 2. Vean cómo se parecen a las de Code Contracts
- 3. Explicar que Fill tiene tmp porque para la ejecución del método es necesario el espacio de memoria de un Logger
- 4. Para que el contrato sea correcto n tiene que ser positivo

```
public void PushFront(Node node)
  Contract Memory Tmp<Logger>(1);
  Contract Memory DestImp();
  Logger logger = new Logger();
  node.Next = this.Head:
  this. Head = node;
  logger.Log("PushFront done");
}
public void Fill(int n)
  Contract.Requires(n > 0); ◀
  Contract Memory Rsd<Node>(Contract Memory This, n);
  Contract.Memory.Tmp<Logger>(1);
  for (int i = 1; i <= n; i++)
  {
     Contract Memory DestRsd(Contract Memory This);
     Node node = new Node(i);
     this. PushFront(node);
```



- 1. Mostrar de a poco y explicar cada anotación
- 2. Vean cómo se parecen a las de Code Contracts
- 3. Explicar que Fill tiene tmp porque para la ejecución del método es necesario el espacio de memoria de un Logger
- 4. Para que el contrato sea correcto n tiene que ser positivo

12 / 30

Anotaciones (1)

- Objetos temporales y residuales: los categorizamos por clases
- Residuales: categorizados según la forma en que escapan del método
- Contratos de consumo
 - Contract.Memory.Tmp<T>(int n, bool cond)
 - Contract.Memory.Rsd<T>(Contract.Memory.RsdType name, int n, bool cond)
- Tipos de residuales
 - Predefinidos: Contract. Memory. This, Contract. Memory. Return
 - Tipo Contract. Memory. RsdType para definir nuevos tipos (como atributos
 - Contract.Memory.BindRsd(Contract.Memory.RsdType name, object expr) para

Jonathan Tapicer Tesis de Licenciatura Febrero de 2011

- 1. Vimos ejemplos, ahora definimos formalmente las anotaciones
- 2. Rsd: aclarar escapa por, porque los agrupamos por tiempo de vida similar, lo necesitamos para identificar la forma en que se usan los objetos residuales en el método llamador
- 3. Contratos: para definir el consumo de memoria de un método, discriminado por tipos
- 4. Tipos de residuales: los que hay, y para definir nuevos tipos

Anotaciones (1)

- Objetos temporales y residuales: los categorizamos por clases
- Residuales: categorizados según la forma en que escapan del método
- Contratos de consumo
 - Contract.Memory.Tmp<T>(int n, bool cond)
 - Contract.Memory.Rsd<T>(Contract.Memory.RsdType name, int n, bool cond)
- Tipos de residuales
 - Predefinidos: Contract. Memory. This, Contract. Memory. Return
 - Tipo Contract. Memory. RsdType para definir nuevos tipos (como atributos
 - Contract.Memory.BindRsd(Contract.Memory.RsdType name, object expr) para

- 1. Vimos ejemplos, ahora definimos formalmente las anotaciones
- 2. Rsd: aclarar escapa por, porque los agrupamos por tiempo de vida similar, lo necesitamos para identificar la forma en que se usan los objetos residuales en el método llamador
- 3. Contratos: para definir el consumo de memoria de un método, discriminado por tipos
- 4. Tipos de residuales: los que hay, y para definir nuevos tipos

Anotaciones (1)

- Objetos temporales y residuales: los categorizamos por clases
- Residuales: categorizados según la forma en que escapan del método
- Contratos de consumo
 - Contract.Memory.Tmp<T>(int n, bool cond)
 - Contract.Memory.Rsd<T>(Contract.Memory.RsdType name, int n, bool cond)
- Tipos de residuales
 - Predefinidos: Contract.Memory.This, Contract.Memory.Return
 - Tipo Contract.Memory.RsdType para definir nuevos tipos (como atributos
 - Contract.Memory.BindRsd(Contract.Memory.RsdType name, object expr) para

- 1. Vimos ejemplos, ahora definimos formalmente las anotaciones
- Rsd: aclarar escapa por, porque los agrupamos por tiempo de vida similar, lo necesitamos para identificar la forma en que se usan los objetos residuales en el método llamador
- 3. Contratos: para definir el consumo de memoria de un método, discriminado por tipos
- 4. Tipos de residuales: los que hay, y para definir nuevos tipos

Anotaciones (1)

- Objetos temporales y residuales: los categorizamos por clases
- Residuales: categorizados según la forma en que escapan del método
- Contratos de consumo
 - Contract.Memory.Tmp<T>(int n, bool cond)
 - Contract.Memory.Rsd<T>(Contract.Memory.RsdType name, int n, bool cond)
- Tipos de residuales
 - Predefinidos: Contract Memory This, Contract Memory Return
 - Tipo Contract.Memory.RsdType para definir nuevos tipos (como atributos en la clase)
 - Contract Memory BindRsd(Contract Memory RsdType name, object expr) para asociarle una expresión

motaciones (1)		
r O tjetos temposiles y militurles: los categoria mos por class Parifurles: categorizatos segán la forma en que esca pará el mitodo		
Contratos de comuneo		
Tributt, milit, ripedict v, vill. soul. Tributt, milit, rice (military, military) vill. v, v, vill. soul.		
Tipos de militados		
Probletidus currers very very para defeir excess charge access The currers charge accepts para defeir excess charge have a with ma- on la charge.		
 contract, many or non-elementary, energy, energies man, object mosel para associate and expression 		

- 1. Vimos ejemplos, ahora definimos formalmente las anotaciones
- Rsd: aclarar escapa por, porque los agrupamos por tiempo de vida similar, lo necesitamos para identificar la forma en que se usan los objetos residuales en el método llamador
- 3. Contratos: para definir el consumo de memoria de un método, discriminado por tipos
- 4. Tipos de residuales: los que hay, y para definir nuevos tipos

Anotaciones (2)

■ Tiempo de vida

- Contract.Memory.DestTmp() para definir que el objeto pertenece a la memoria temporal del método
- Contract.Memory.DestRsd(Contract.Memory.RsdType name) para definir que el objeto pertenece a la memoria residual de nombre name del método
- Contract Memory AddTmp(Contract Memory RsdType name_call) y Contract.Memory.AddRsd(Contract.Memory.RsdType name_local, Contract. Memory RsdType name_call) para transferencia de residuales en calls
- - Contract.Memory.IterationSpace(bool cond) para definir espacios de

Jonathan Tapicer Tesis de Licenciatura 13 / 30



- 1. Tiempo de vida: para anotar cuáles objetos son tmp y cuáles rsd, y qué pasa con los objetos rsd de un método invocado
- 2. Espacios de iteración: para entender la cantidad de veces que ocurren los consumos de memoria adentro de loops

13 / 30

Anotaciones (2)

■ Tiempo de vida

- Contract.Memory.DestTmp() para definir que el objeto pertenece a la memoria temporal del método
- Contract.Memory.DestRsd(Contract.Memory.RsdType name) para definir que el objeto pertenece a la memoria residual de nombre name del método
- Contract Memory AddTmp(Contract Memory RsdType name_call) y Contract.Memory.AddRsd(Contract.Memory.RsdType name_local, Contract. Memory RsdType name_call) para transferencia de residuales en calls
- Espacios de iteración
 - Contract Memory IterationSpace(bool cond) para definir espacios de iteración en loops

Jonathan Tapicer Tesis de Licenciatura Febrero de 2011



- 1. Tiempo de vida: para anotar cuáles objetos son tmp y cuáles rsd, y qué pasa con los objetos rsd de un método invocado
- 2. Espacios de iteración: para entender la cantidad de veces que ocurren los consumos de memoria adentro de loops

Índice

- Introducción
- 2 Anotaciones
- 3 Verificación
- 4 Limitaciones y trabajo futuro
- 5 Conclusiones

Verificación

- ¿Cómo verificamos que las anotaciones son correctas?
- Contratos
 - Correctitud de las anotaciones Contract. Memory. Tmp y Contract. Memory. Rsd
 - Instrumentación y uso de verificador estático
 - Soporte de herramienta aritmética
- Tiempo de vida
 - Correctitud de las anotaciones Contract.Memory.DestTmp,
 - Análisis de points-to y escape

- 1. Esta parte es el desarrollo central del trabajo
- correcto

2. Verificar que el consumo que el usuario dice que tiene su programa sea

- 3. Hay diferentes cosas a verificar, cada una la resolvemos de diferente forma
- 4. Instrumentación: generar un programa equivalente pero con código insertado que nos va a permitir que Code Contracts verifique los contratos de memoria
- 5. Empezamos viendo un ejemplo de la instrumentación básica

Verificación

¿Cómo verificamos que las anotaciones son correctas?

- Contratos
 - Correctitud de las anotaciones Contract.Memory.Tmp y Contract.Memory.Rsd
 - Instrumentación y uso de verificador estático
 - Soporte de herramienta aritmética
- Tiempo de vida
 - Correctitud de las anotaciones Contract.Memory.DestTmp,
 Contract.Memory.DestRsd, Contract.Memory.AddTmp y Contract.Memory.AddRsd
 - Análisis de points-to y escape

- 1. Esta parte es el desarrollo central del trabajo
- Verificar que el consumo que el usuario dice que tiene su programa sea correcto
- 3. Hay diferentes cosas a verificar, cada una la resolvemos de diferente forma
- 4. Instrumentación: generar un programa equivalente pero con código insertado que nos va a permitir que Code Contracts verifique los contratos de memoria
- 5. Empezamos viendo un ejemplo de la instrumentación básica

Verificación

- ¿Cómo verificamos que las anotaciones son correctas?
- Contratos
 - Correctitud de las anotaciones Contract. Memory. Tmp y Contract. Memory. Rsd
 - Instrumentación y uso de verificador estático
 - Soporte de herramienta aritmética
- Tiempo de vida
 - Correctitud de las anotaciones Contract Memory DestImp, Contract Memory DestRsd, Contract Memory AddTmp V Contract Memory AddRsd
 - Análisis de points-to y escape

- 1. Esta parte es el desarrollo central del trabajo
- Verificar que el consumo que el usuario dice que tiene su programa sea correcto
- 3. Hay diferentes cosas a verificar, cada una la resolvemos de diferente forma
- 4. Instrumentación: generar un programa equivalente pero con código insertado que nos va a permitir que Code Contracts verifique los contratos de memoria
- 5. Empezamos viendo un ejemplo de la instrumentación básica

Verificación mediante instrumentación, ejemplo 1

```
public void PushFront(Node node)
{
  Contract.Memory.Tmp<Logger>(1);
  Contract Memory DestImp();
  Logger logger = new Logger();
  node.Next = this.Head:
  this.Head = node;
  logger.Log("PushFront done");
```

Țesis de Licenciatura
└─Verifi cación
1

igsqcup Verificación mediante instrumentación, ejemplo 1

```
Verificación mediante instrumentación, ejemplo 1
```

1. Mostrar de a poco y explicar la instrumentación

Verificación mediante instrumentación, ejemplo 1

```
public static int IntLinkedList_PushFront_Tmp_Logger; 
public void PushFront(Node node)
{
  Contract.Memory.Tmp<Logger>(1);
  Contract Memory DestImp();
  Logger logger = new Logger();
  node.Next = this.Head:
  this. Head = node;
  logger.Log("PushFront done");
```

	Tesis de Licenciatura
1	└─Verifi cación
1	1

*	ificación mediante instrumentación, ejemplo 1
,.	and the contract of the contra
į.	NAME AND ADDRESS A
	mananing mpaggeous
	control control con Control Co
	togger meger togger :;
	constant + territory
	COLUMN * 1841;
	togger.org ("Fill Affects Albert);

1. Mostrar de a poco y explicar la instrumentación

```
public static int IntLinkedList_PushFront_Tmp_Logger;
public void PushFront(Node node)
{
  Contract Memory Tmp<Logger>(1);
  IntLinkedList_PushFront_Tmp_Logger = 0;
  Contract Memory DestImp();
  Logger logger = new Logger();
  node.Next = this.Head:
  this. Head = node;
  logger.Log("PushFront done");
```

	Tesis de Licenciatura
2	Verifi cación
0	Verificación mediante instrumentación, ejemplo 1

, 1111	· · toto · · M. little lived. 15_Pilliprint_tip_trippir;
	· · · · · · · PEL METELS BEAR SEAR!
	management and management (
	manuscriptions (mppinger + 1) -
	ann.mg.may#;
	mer meger retter :
	Autor - Columbia
	H-1886 * 1881)

1. Mostrar de a poco y explicar la instrumentación

```
public static int IntLinkedList_PushFront_Tmp_Logger;
public void PushFront(Node node)
{
  Contract Memory Tmp<Logger>(1);
   IntLinkedList_PushFront_Tmp_Logger = 0;
   Contract Memory DestImp();
   IntLinkedList_PushFront_Tmp_Logger++;
   Logger logger = new Logger();
  node.Next = this.Head:
   this.Head = node;
  logger.Log("PushFront done");
```

Verifi cación			
└─Verificación	mediante instrumentación,	ejemplo :	1



1. Mostrar de a poco y explicar la instrumentación

```
public static int IntLinkedList_PushFront_Tmp_Logger;
public void PushFront(Node node)
{
  Contract.Memory.Tmp<Logger>(1);
  Contract.Ensures(IntLinkedList_PushFront_Tmp_Logger <= 1); <</pre>
  IntLinkedList_PushFront_Tmp_Logger = 0;
  Contract Memory DestImp();
  IntLinkedList_PushFront_Tmp_Logger++;
  Logger logger = new Logger();
  node.Next = this.Head:
  this.Head = node;
  logger.Log("PushFront done");
```

	Țesis de Licenciatura
1	└─Verifi cación
,	1

```
Ve dicadi n mediante in tramentación, ejemph. 1
```

1. Mostrar de a poco y explicar la instrumentación

```
public void Fill(int n)
  Contract.Requires(n > 0);
   Contract.Memory.Rsd<Node>(Contract.Memory.This, n);
   Contract.Memory.Tmp<Logger>(1);
  for (int i = 1; i <= n; i++)</pre>
     Contract Memory DestRsd(Contract Memory This);
     Node node = new Node(i);
     this. PushFront(node):
```



- 1. Este método es más complejo porque hay un call y un loop
- 2. Hay que tener en cuenta el tmp del método llamado
- 3. Si tuviese rsd también hay que tenerlo en cuenta
- 4. Maximizamos el tmp del callee porque necesitamos conocer el máximo de memoria temporal necesario en cualquier invocación, no la suma porque se libera cuando termina el método
- 5. Aclarar: anotaciones AddTmp y AddRsd no las mostramos porque no son necesarias, pero la instrumentación es muy similar

```
public static int IntLinkedList_Fill_Rsd_This_Node; 
public void Fill(int n)
  Contract.Requires(n > 0);
  Contract.Memory.Rsd<Node>(Contract.Memory.This, n);
  Contract.Memory.Tmp<Logger>(1);
  for (int i = 1; i <= n; i++)</pre>
     Contract Memory DestRsd(Contract Memory This);
     Node node = new Node(i);
     this. PushFront(node):
```



- 1. Este método es más complejo porque hay un call y un loop
- 2. Hay que tener en cuenta el tmp del método llamado
- 3. Si tuviese rsd también hay que tenerlo en cuenta
- 4. Maximizamos el tmp del callee porque necesitamos conocer el máximo de memoria temporal necesario en cualquier invocación, no la suma porque se libera cuando termina el método
- 5. Aclarar: anotaciones AddTmp y AddRsd no las mostramos porque no son necesarias, pero la instrumentación es muy similar

```
public static int IntLinkedList_Fill_Rsd_This_Node;
public static int IntLinkedList_Fill_Tmp_Logger; 
public void Fill(int n)
  Contract.Requires(n > 0);
  Contract Memory Rsd<Node>(Contract Memory This, n);
  Contract.Memory.Tmp<Logger>(1);
  for (int i = 1; i <= n; i++)</pre>
     Contract Memory DestRsd(Contract Memory This);
     Node node = new Node(i):
     this. PushFront(node):
```



- 1. Este método es más complejo porque hay un call y un loop
- 2. Hay que tener en cuenta el tmp del método llamado
- 3. Si tuviese rsd también hay que tenerlo en cuenta
- 4. Maximizamos el tmp del callee porque necesitamos conocer el máximo de memoria temporal necesario en cualquier invocación, no la suma porque se libera cuando termina el método
- Aclarar: anotaciones AddTmp y AddRsd no las mostramos porque no son necesarias, pero la instrumentación es muy similar

```
public static int IntLinkedList_Fill_Rsd_This_Node;
public static int IntLinkedList_Fill_Tmp_Logger;
public void Fill(int n)
  Contract.Requires(n > 0);
  Contract.Memory.Rsd<Node>(Contract.Memory.This, n);
  Contract.Memory.Tmp<Logger>(1);
  IntLinkedList_Fill_Rsd_This_Node = 0;
  for (int i = 1; i <= n; i++)</pre>
     Contract Memory DestRsd(Contract Memory This);
     Node node = new Node(i):
     this. PushFront(node):
```



- 1. Este método es más complejo porque hay un call y un loop
- 2. Hay que tener en cuenta el tmp del método llamado
- 3. Si tuviese rsd también hay que tenerlo en cuenta
- 4. Maximizamos el tmp del callee porque necesitamos conocer el máximo de memoria temporal necesario en cualquier invocación, no la suma porque se libera cuando termina el método
- Aclarar: anotaciones AddTmp y AddRsd no las mostramos porque no son necesarias, pero la instrumentación es muy similar

```
public static int IntLinkedList_Fill_Rsd_This_Node;
public static int IntLinkedList_Fill_Tmp_Logger;
public void Fill(int n)
  Contract.Requires(n > 0);
  Contract.Memory.Rsd<Node>(Contract.Memory.This, n);
  Contract.Memory.Tmp<Logger>(1);
  IntLinkedList_Fill_Rsd_This_Node = 0;
  IntLinkedList_Fill_Tmp_Logger = 0;
  for (int i = 1; i <= n; i++)
     Contract Memory DestRsd(Contract Memory This);
     Node node = new Node(i):
     this. PushFront(node):
```



- 1. Este método es más complejo porque hay un call y un loop
- 2. Hay que tener en cuenta el tmp del método llamado
- 3. Si tuviese rsd también hay que tenerlo en cuenta
- 4. Maximizamos el tmp del callee porque necesitamos conocer el máximo de memoria temporal necesario en cualquier invocación, no la suma porque se libera cuando termina el método
- Aclarar: anotaciones AddTmp y AddRsd no las mostramos porque no son necesarias, pero la instrumentación es muy similar

```
public static int IntLinkedList_Fill_Rsd_This_Node;
public static int IntLinkedList_Fill_Tmp_Logger;
public void Fill(int n)
  Contract.Requires(n > 0);
  Contract.Memory.Rsd<Node>(Contract.Memory.This, n);
  Contract.Memory.Tmp<Logger>(1);
  IntLinkedList_Fill_Rsd_This_Node = 0;
  IntLinkedList_Fill_Tmp_Logger = 0;
  for (int i = 1; i <= n; i++)
     Contract Memory DestRsd(Contract Memory This);
     IntLinkedList_Fill_Rsd_This_Node++;
     Node node = new Node(i):
     this. PushFront(node):
```



- 1. Este método es más complejo porque hay un call y un loop
- 2. Hay que tener en cuenta el tmp del método llamado
- 3. Si tuviese rsd también hay que tenerlo en cuenta
- 4. Maximizamos el tmp del callee porque necesitamos conocer el máximo de memoria temporal necesario en cualquier invocación, no la suma porque se libera cuando termina el método
- Aclarar: anotaciones AddTmp y AddRsd no las mostramos porque no son necesarias, pero la instrumentación es muy similar

```
public static int IntLinkedList_Fill_Rsd_This_Node;
public static int IntLinkedList_Fill_Tmp_Logger;
public void Fill(int n)
  Contract.Requires(n > 0);
  Contract.Memory.Rsd<Node>(Contract.Memory.This, n);
  Contract.Memory.Tmp<Logger>(1);
  IntLinkedList_Fill_Rsd_This_Node = 0;
  IntLinkedList_Fill_Tmp_Logger = 0;
  int max_PushFront_Logger = 0;
  for (int i = 1; i <= n; i++)
     Contract Memory DestRsd(Contract Memory This);
     IntLinkedList Fill Rsd This Node++:
     Node node = new Node(i):
     this. PushFront(node):
```



- 1. Este método es más complejo porque hay un call y un loop
- 2. Hay que tener en cuenta el tmp del método llamado
- 3. Si tuviese rsd también hay que tenerlo en cuenta
- 4. Maximizamos el tmp del callee porque necesitamos conocer el máximo de memoria temporal necesario en cualquier invocación, no la suma porque se libera cuando termina el método
- Aclarar: anotaciones AddTmp y AddRsd no las mostramos porque no son necesarias, pero la instrumentación es muy similar

```
public static int IntLinkedList_Fill_Rsd_This_Node;
public static int IntLinkedList_Fill_Tmp_Logger;
public void Fill(int n)
  Contract.Requires(n > 0);
  Contract.Memory.Rsd<Node>(Contract.Memory.This, n);
  Contract.Memory.Tmp<Logger>(1);
  IntLinkedList_Fill_Rsd_This_Node = 0;
  IntLinkedList_Fill_Tmp_Logger = 0;
  int max_PushFront_Logger = 0;
  for (int i = 1; i <= n; i++)
     Contract Memory DestRsd(Contract Memory This);
     IntLinkedList Fill Rsd This Node++:
     Node node = new Node(i);
     this. PushFront(node):
     max_PushFront_Logger = Math.Max(IntLinkedList_PushFront_Tmp_Logger,
                                      max_PushFront_Logger);
```



- 1. Este método es más complejo porque hay un call y un loop
- 2. Hay que tener en cuenta el tmp del método llamado
- 3. Si tuviese rsd también hay que tenerlo en cuenta
- 4. Maximizamos el tmp del callee porque necesitamos conocer el máximo de memoria temporal necesario en cualquier invocación, no la suma porque se libera cuando termina el método
- Aclarar: anotaciones AddTmp y AddRsd no las mostramos porque no son necesarias, pero la instrumentación es muy similar

```
public static int IntLinkedList_Fill_Rsd_This_Node;
public static int IntLinkedList_Fill_Tmp_Logger;
public void Fill(int n)
  Contract.Requires(n > 0);
  Contract.Memory.Rsd<Node>(Contract.Memory.This, n);
  Contract.Memory.Tmp<Logger>(1);
  IntLinkedList_Fill_Rsd_This_Node = 0;
  IntLinkedList_Fill_Tmp_Logger = 0;
  int max_PushFront_Logger = 0;
  for (int i = 1; i <= n; i++)
     Contract Memory DestRsd(Contract Memory This);
     IntLinkedList Fill Rsd This Node++:
     Node node = new Node(i);
     this. PushFront(node):
     max_PushFront_Logger = Math.Max(IntLinkedList_PushFront_Tmp_Logger,
                                      max_PushFront_Logger);
  IntLinkedList_Fill_Tmp_Logger += max_PushFront_Logger;
```



- 1. Este método es más complejo porque hay un call y un loop
- 2. Hay que tener en cuenta el tmp del método llamado
- 3. Si tuviese rsd también hay que tenerlo en cuenta
- 4. Maximizamos el tmp del callee porque necesitamos conocer el máximo de memoria temporal necesario en cualquier invocación, no la suma porque se libera cuando termina el método
- Aclarar: anotaciones AddTmp y AddRsd no las mostramos porque no son necesarias, pero la instrumentación es muy similar

```
public static int IntLinkedList_Fill_Rsd_This_Node;
public static int IntLinkedList_Fill_Tmp_Logger;
public void Fill(int n)
  Contract.Requires(n > 0);
  Contract.Memory.Rsd<Node>(Contract.Memory.This, n);
  Contract.Memory.Tmp<Logger>(1);
  Contract.Ensures(IntLinkedList_Fill_Rsd_This_Node <= n); ◀</pre>
  IntLinkedList_Fill_Rsd_This_Node = 0;
  IntLinkedList_Fill_Tmp_Logger = 0;
  int max_PushFront_Logger = 0;
  for (int i = 1; i <= n; i++)
     Contract Memory DestRsd(Contract Memory This);
     IntLinkedList Fill Rsd This Node++:
     Node node = new Node(i);
     this. PushFront(node):
     max_PushFront_Logger = Math.Max(IntLinkedList_PushFront_Tmp_Logger,
                                      max_PushFront_Logger);
  IntLinkedList_Fill_Tmp_Logger += max_PushFront_Logger;
```



- 1. Este método es más complejo porque hay un call y un loop
- 2. Hay que tener en cuenta el tmp del método llamado
- 3. Si tuviese rsd también hay que tenerlo en cuenta
- 4. Maximizamos el tmp del callee porque necesitamos conocer el máximo de memoria temporal necesario en cualquier invocación, no la suma porque se libera cuando termina el método
- Aclarar: anotaciones AddTmp y AddRsd no las mostramos porque no son necesarias, pero la instrumentación es muy similar

```
public static int IntLinkedList_Fill_Rsd_This_Node;
public static int IntLinkedList_Fill_Tmp_Logger;
public void Fill(int n)
  Contract.Requires(n > 0);
  Contract Memory Rsd<Node>(Contract Memory This, n);
  Contract.Memory.Tmp<Logger>(1);
  Contract.Ensures(IntLinkedList_Fill_Rsd_This_Node <= n);</pre>
  Contract.Ensures(IntLinkedList_Fill_Tmp_Logger <= 1); <</pre>
  IntLinkedList_Fill_Rsd_This_Node = 0;
  IntLinkedList_Fill_Tmp_Logger = 0;
  int max_PushFront_Logger = 0;
  for (int i = 1; i <= n; i++)
     Contract Memory DestRsd(Contract Memory This);
     IntLinkedList Fill Rsd This Node++:
     Node node = new Node(i);
     this. PushFront(node):
     max_PushFront_Logger = Math.Max(IntLinkedList_PushFront_Tmp_Logger,
                                       max_PushFront_Logger);
  IntLinkedList_Fill_Tmp_Logger += max_PushFront_Logger;
```

```
We fileach a mediante intrumentación, ejembo 2

***Transport de l'accionation de l'accionat
```

- 1. Este método es más complejo porque hay un call y un loop
- 2. Hay que tener en cuenta el tmp del método llamado
- 3. Si tuviese rsd también hay que tenerlo en cuenta
- 4. Maximizamos el tmp del callee porque necesitamos conocer el máximo de memoria temporal necesario en cualquier invocación, no la suma porque se libera cuando termina el método
- Aclarar: anotaciones AddTmp y AddRsd no las mostramos porque no son necesarias, pero la instrumentación es muy similar

Verificación de anotaciones de tiempo de vida

- Necesitamos verificar que las anotaciones Contract.Memory.DestTmp,
 Contract.Memory.DestRsd, Contract.Memory.AddTmp y Contract.Memory.AddRsd SON correctas
- Construimos un Points-to Graph (PTG) del método haciendo un análisis de points-to, grafo dirigido donde los nodos son objetos (o conjuntos de) y los arcos referencias entre ellos
- A partir del PTG hacemos un análisis de escape de objetos para ver qué objetos exceden el tiempo de vida del método
- Verificamos
 - Que los objetos escapen del método si y sólo si están anotados como residuales
 - Que los objetos residuales escapen del método a través de la expresión indicada

Jonathan Tapicer Tesis de Licenciatura Febrero de 2011 18 / 30

Ve ificació n de antacio nor de tiempo de vida

Bionio no vefico en las constitues monocomposarios

Bionio no vefico en las constitues monocomposarios

Bionio no vefico en las constitues de y monocomposarios

Bionio no vefico en la constitue de y monocomposarios

Bionio no vefico en la constitue de y monocomposarios

Bionio no vefico en la constitue de y monocomposarios

Bionio no vefico en la constitue de y monocomposarios

Bionio no vefico en la constitue de y monocomposarios

Bionio no vefico en la constitue de y monocomposarios

Bionio no vefico en la constitue de y monocomposarios

Bionio no vefico en la constitue de y monocomposarios

Bionio no vefico en la constitue de y monocomposarios

Bionio no vefico en la constitue de y monocomposarios

Bionio no vefico en la constitue de y monocomposarios

Bionio no vefico en la constitue de y monocomposarios

Bionio no vefico en la constitue de y monocomposarios

Bionio no vefico en la constitue de y monocomposarios

Bionio no vefico en la constitue de y monocomposarios

Bionio no vefico en la constitue de y monocomposarios

Bionio no vefico en la constitue de y monocomposarios

Bionio no vefico en la constitue de y monocomposarios

Bionio no vefico en la constitue de y monocomposarios

Bionio no vefico en la constitue de y monocomposarios

Bionio no vefico en la constitue de y monocomposarios

Bionio no vefico en la constitue de y monocomposarios

Bionio no vefico en la constitue de y monocomposarios

Bionio no vefico en la constitue de y monocomposarios

Bionio no vefico en la constitue de y monocomposarios

Bionio no vefico en la constitue de y monocomposarios

Bionio no vefico en la constitue de y monocomposarios

Bionio no vefico en la constitue de y monocomposarios

Bionio no vefico en la constitue de y monocomposarios

Bionio no vefico en la constitue de y monocomposarios

Bionio no vefico en la constitue de y monocomposarios

Bionio no vefico en la constitue de y monocomposarios

Bionio no vefico en la constitue de y monocomposarios

Bionio no vefico en la

1. Para que los contratos sean correctos, las anotaciones de tiempo de vida tienen que serlo

Verificación de anotaciones de tiempo de vida

- Necesitamos verificar que las anotaciones Contract.Memory.DestTmp,
 Contract.Memory.DestRsd, Contract.Memory.AddTmp y Contract.Memory.AddRsd SON correctas
- Construimos un Points-to Graph (PTG) del método haciendo un análisis de points-to, grafo dirigido donde los nodos son objetos (o conjuntos de) y los arcos referencias entre ellos
- A partir del PTG hacemos un análisis de escape de objetos para ver qué objetos exceden el tiempo de vida del método
- Verificamos
 - Que los objetos escapen del método si y sólo si están anotados como residuales
 - Que los objetos residuales escapen del método a través de la expresión indicada

Jonathan Tapicer Tesis de Licenciatura Febrero de 2011 18 / 30

Ve ifficación de antación se de tiempo de vida

Brancho ma evitar que la candidata con accessora,

accessora en el contra que la candidata con accessora,

accessora en el contra de contra con

1. Para que los contratos sean correctos, las anotaciones de tiempo de vida tienen que serlo

- Necesitamos verificar que las anotaciones Contract.Memory.DestTmp,

 Contract.Memory.DestRsd, Contract.Memory.AddTmp y Contract.Memory.AddRsd SON

 correctas
- Construimos un Points-to Graph (PTG) del método haciendo un análisis de points-to, grafo dirigido donde los nodos son objetos (o conjuntos de) y los arcos referencias entre ellos
- A partir del PTG hacemos un análisis de escape de objetos para ver qué objetos exceden el tiempo de vida del método
- Verificamos
 - Que los objetos escapen del método si y sólo si están anotados como residuales
 - Que los objetos residuales escapen del método a través de la expresión indicada

Jonathan Tapicer Tesis de Licenciatura Febrero de 2011 18 / 30

−Verificación de anotaciones de tiempo de vida

We file cards on the a material read of the length of a wide

Broads on the wide of the cards discussed and a supplier of the cards of

1. Para que los contratos sean correctos, las anotaciones de tiempo de vida tienen que serlo

Verificación de anotaciones de tiempo de vida

- Necesitamos verificar que las anotaciones Contract.Memory.DestTmp,
 Contract.Memory.DestRsd, Contract.Memory.AddTmp y Contract.Memory.AddRsd SON correctas
- Construimos un Points-to Graph (PTG) del método haciendo un análisis de points-to, grafo dirigido donde los nodos son objetos (o conjuntos de) y los arcos referencias entre ellos
- A partir del PTG hacemos un análisis de escape de objetos para ver qué objetos exceden el tiempo de vida del método
- Verificamos
 - Que los objetos escapen del método si y sólo si están anotados como residuales
 - Que los objetos residuales escapen del método a través de la expresión indicada

Jonathan Tapicer Tesis de Licenciatura Febrero de 2011 18 / 30

─Verificación de anotaciones de tiempo de vida

We file called the a material was de thimps de vida

In troub me orient and a sendant manual material and a material material plan and a final material material plan and a final material material material plan and a final material material

intikata

1. Para que los contratos sean correctos, las anotaciones de tiempo de vida tienen que serlo

Verificación de anotaciones de tiempo de vida, ejemplo

```
public void Fill(int n)
2
3
      Contract.Requires(n > 0);
4
      Contract . Memory . Rsd < Node > (Contract . Memory . This, n);
5
      Contract.Memory.Tmp < Logger > (1);
6
7
      for (int i = 1; i <= n; i++)</pre>
8
9
        Contract Memory DestRsd(Contract Memory This);
10
        Node node = new Node(i);
11
         this. PushFront (node);
12
      }
13
```

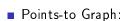
Verificación de anotaciones de tiempo de vida, ejemplo

- 1. Método Fill anotado
- 2. Vamos a ver la info que obtener del análisis de points-to y escape
- 3. Y cómo usamos esa info para verificar
- 4. Con la misma info podemos verificar que AddTmp y AddRsd sean correctos, es decir que los objetos residuales de un método invocado se conviertan en lo tmp o rsd locales según la anotación dada diga

Verificación de anotaciones de tiempo de vida, ejemplo

```
public void Fill(int n)
2
3
      Contract.Requires(n > 0);
4
      Contract . Memory . Rsd < Node > (Contract . Memory . This, n);
5
      Contract.Memory.Tmp < Logger > (1);
6
7
      for (int i = 1; i <= n; i++)</pre>
8
9
        Contract Memory DestRsd(Contract Memory This);
10
         Node node = new Node(i):
11
         this. PushFront (node);
12
13
```

Del análisis de points-to y escape obtenemos los siguientes datos:





El objeto de la línea 10 escapa

Jonathan Tapicer

-Verificación de anotaciones de tiempo de vida, ejemplo

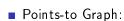


- 1. Método Fill anotado
- 2. Vamos a ver la info que obtener del análisis de points-to y escape
- 3. Y cómo usamos esa info para verificar
- 4. Con la misma info podemos verificar que AddTmp y AddRsd sean correctos, es decir que los objetos residuales de un método invocado se conviertan en lo tmp o rsd locales según la anotación dada diga

Verificación de anotaciones de tiempo de vida, ejemplo

```
public void Fill(int n)
2
3
      Contract.Requires(n > 0);
4
      Contract . Memory . Rsd < Node > (Contract . Memory . This, n);
5
      Contract.Memory.Tmp < Logger > (1);
6
7
      for (int i = 1; i <= n; i++)</pre>
8
9
        Contract Memory DestRsd(Contract Memory This);
10
         Node node = new Node(i):
11
         this. PushFront (node);
12
13
```

Del análisis de points-to y escape obtenemos los siguientes datos:





El objeto de la línea 10 escapa \Longrightarrow es residual, DestRsd es correcta

Jonathan Tapicer

Ve dicación de a natacione de tiempo de vida, ejemplo

| Proposition | P

- 1. Método Fill anotado
- 2. Vamos a ver la info que obtener del análisis de points-to y escape

-Verificación de anotaciones de tiempo de vida, ejemplo

- 3. Y cómo usamos esa info para verificar
- 4. Con la misma info podemos verificar que AddTmp y AddRsd sean correctos, es decir que los objetos residuales de un método invocado se conviertan en lo tmp o rsd locales según la anotación dada diga

Verificación de anotaciones de tiempo de vida, ejemplo

```
public void Fill(int n)
2
3
      Contract.Requires(n > 0);
4
      Contract . Memory . Rsd < Node > (Contract . Memory . This, n);
5
      Contract . Memory . Tmp < Logger > (1);
6
7
      for (int i = 1; i <= n; i++)</pre>
8
9
         Contract Memory DestRsd(Contract Memory This);
10
         Node node = new Node(i):
11
         this. PushFront (node);
12
13
```

Del análisis de points-to y escape obtenemos los siguientes datos:

■ Points-to Graph:



El objeto de la línea 10 escapa \Longrightarrow es residual, DestRsd es correcta

-Verificación de anotaciones de tiempo de vida, ejemplo



- 1. Método Fill anotado
- 2. Vamos a ver la info que obtener del análisis de points-to y escape
- 3. Y cómo usamos esa info para verificar
- 4. Con la misma info podemos verificar que AddTmp y AddRsd sean correctos, es decir que los objetos residuales de un método invocado se conviertan en lo tmp o rsd locales según la anotación dada diga

Anotaciones Verificación Limitaciones y trabajo futuro Conclusiones

Verificación con aritmética no lineal

■ El verificador de Code Contracts sólo soporta aritmética lineal

- Integramos una herramienta externa, Barvinok, capaz de resolver operaciones con polinomios
- Durante la verificación
 - Usamos la herramienta para calcular expresiones polinomiales e incrementamos los contadores con valores pre-calculados
 - Obtenemos aserciones acerca de la validez de los contratos y las brindamos al verificador mediante la anotación Assume
- Requerimos para estos casos que se anote un IterationSpace para los ciclos

- 1. Aclarar que otros verificadores estáticos también manejan sólo aritmética lineal
- 2. En gral, aparece una multiplicación e ignoran, no pueden probar
- 3. En consumo de memoria aparecen fácil las multiplicaciones, pensar en loop anidado, parecido a análisis de complejidad temporal

Anotaciones **Verificación** Limitaciones y trabajo futuro Conc

Verificación con aritmética no lineal

- El verificador de Code Contracts sólo soporta aritmética lineal
- Integramos una herramienta externa, Barvinok, capaz de resolver operaciones con polinomios
- Durante la verificación
 - Usamos la herramienta para calcular expresiones polinomiales e incrementamos los contadores con valores pre-calculados
 - Obtenemos aserciones acerca de la validez de los contratos y las brindamos al verificador mediante la anotación Assume
- Requerimos para estos casos que se anote un IterationSpace para los ciclos

- 1. Aclarar que otros verificadores estáticos también manejan sólo aritmética lineal
- 2. En gral, aparece una multiplicación e ignoran, no pueden probar
- 3. En consumo de memoria aparecen fácil las multiplicaciones, pensar en loop anidado, parecido a análisis de complejidad temporal

Verificación con aritmética no lineal

- El verificador de Code Contracts sólo soporta aritmética lineal
- Integramos una herramienta externa, Barvinok, capaz de resolver operaciones con polinomios
- Durante la verificación
 - Usamos la herramienta para calcular expresiones polinomiales e incrementamos los contadores con valores pre-calculados
 - Obtenemos aserciones acerca de la validez de los contratos y las brindamos al verificador mediante la anotación Assume
- Requerimos para estos casos que se anote un IterationSpace para los ciclos

- 1. Aclarar que otros verificadores estáticos también manejan sólo aritmética lineal
- 2. En gral, aparece una multiplicación e ignoran, no pueden probar
- 3. En consumo de memoria aparecen fácil las multiplicaciones, pensar en loop anidado, parecido a análisis de complejidad temporal

Verificación con aritmética no lineal

- El verificador de Code Contracts sólo soporta aritmética lineal
- Integramos una herramienta externa, Barvinok, capaz de resolver operaciones con polinomios
- Durante la verificación
 - Usamos la herramienta para calcular expresiones polinomiales e incrementamos los contadores con valores pre-calculados
 - Obtenemos aserciones acerca de la validez de los contratos y las brindamos al verificador mediante la anotación Assume
- Requerimos para estos casos que se anote un IterationSpace para los ciclos

Welfinerin can all milica no final

multiple of the control of th

- 1. Aclarar que otros verificadores estáticos también manejan sólo aritmética lineal
- 2. En gral, aparece una multiplicación e ignoran, no pueden probar
- 3. En consumo de memoria aparecen fácil las multiplicaciones, pensar en loop anidado, parecido a análisis de complejidad temporal

```
public void ConsumoCuadratico(int n)
3
       Contract.Requires(n > 0);
4
       Contract.Memory.Tmp < Logger > (n * n);
5
6
       for (int i = 1; i <= n; i++)</pre>
7
8
           for (int j = 1; j <= n; j++)</pre>
9
10
              Contract Memory DestImp();
11
              Logger logger = new Logger();
              logger.Log("Log " + (i * j).ToString());
12
13
14
15
```

- El contrato dado es correcto
- El verificador de Code Contracts no es capaz de verificarlo con la instrumentación descripta hasta el momento
- Con un poco de ayuda del usuario y de una herramienta externa podemos verificarlo



1. Ejemplo que no puede verificar Code Contracts

```
public void ConsumoCuadratico(int n)
3
       Contract.Requires(n > 0);
       Contract.Memory.Tmp < Logger > (n * n);
5
6
       for (int i = 1; i <= n; i++)</pre>
7
8
           for (int j = 1; j <= n; j++)</pre>
9
10
              Contract Memory DestImp();
11
              Logger logger = new Logger();
              logger.Log("Log " + (i * j).ToString());
12
13
14
15
```

El contrato dado es correcto

- El verificador de Code Contracts no es capaz de verificarlo con la instrumentación
- Con un poco de ayuda del usuario y de una herramienta externa podemos verificarlo

Jonathan Tapicer



1. Ejemplo que no puede verificar Code Contracts

```
public void ConsumoCuadratico(int n)
3
       Contract.Requires(n > 0);
       Contract.Memory.Tmp < Logger > (n * n);
5
6
       for (int i = 1; i <= n; i++)</pre>
7
8
           for (int j = 1; j <= n; j++)</pre>
9
10
              Contract Memory DestImp();
11
              Logger logger = new Logger();
              logger.Log("Log " + (i * j).ToString());
12
13
14
15
```

- El contrato dado es correcto
- El verificador de Code Contracts no es capaz de verificarlo con la instrumentación descripta hasta el momento
- Con un poco de ayuda del usuario y de una herramienta externa podemos verificarlo

Jonathan Tapicer



1. Ejemplo que no puede verificar Code Contracts

```
public void ConsumoCuadratico(int n)
3
       Contract.Requires(n > 0);
       Contract.Memory.Tmp < Logger > (n * n);
5
6
       for (int i = 1; i <= n; i++)</pre>
7
8
          for (int j = 1; j <= n; j++)
9
10
              Contract Memory DestImp();
11
              Logger logger = new Logger();
              logger.Log("Log " + (i * j).ToString());
12
13
14
15
```

- El contrato dado es correcto
- El verificador de Code Contracts no es capaz de verificarlo con la instrumentación descripta hasta el momento
- Con un poco de ayuda del usuario y de una herramienta externa podemos verificarlo

Jonathan Tapicer



1. Ejemplo que no puede verificar Code Contracts

Para poder calcular la cantidad de objetos temporales necesarios necesitamos ayuda del usuario del para entender los espacios de iteración:

```
public void ConsumoCuadratico(int n)
{
  Contract.Requires(n > 0);
  Contract.Memory.Tmp<Logger>(n * n);
  for (int i = 1; i <= n; i++)
  {
     for (int j = 1; j \le n; j++)
        Contract.Memory.DestTmp();
        Logger logger = new Logger();
        logger.Log("Log "+ (i * j).ToString());
```



- 1. Si el usuario anota los espacios de iteración (fácil), podemos hacer una instrumentación diferente
- 2. Ahora vemos la instrumentación nueva

Para poder calcular la cantidad de objetos temporales necesarios necesitamos ayuda del usuario del para entender los espacios de iteración:

```
public void ConsumoCuadratico(int n)
{
  Contract.Requires(n > 0);
  Contract.Memory.Tmp<Logger>(n * n);
  for (int i = 1; i <= n; i++)
  {
     Contract. Memory. IterationSpace(1 <= i && i <= n); ◀
     for (int j = 1; j \le n; j++)
        Contract.Memory.DestTmp();
        Logger logger = new Logger();
        logger.Log("Log "+ (i * j).ToString());
```



- 1. Si el usuario anota los espacios de iteración (fácil), podemos hacer una instrumentación diferente
- 2. Ahora vemos la instrumentación nueva

Para poder calcular la cantidad de objetos temporales necesarios necesitamos ayuda del usuario del para entender los espacios de iteración:

```
public void ConsumoCuadratico(int n)
{
  Contract.Requires(n > 0);
  Contract.Memory.Tmp<Logger>(n * n);
  for (int i = 1; i <= n; i++)
  {
     Contract.Memory.IterationSpace(1 <= i && i <= n);</pre>
     for (int j = 1; j \le n; j++)
        Contract.Memory.IterationSpace(1 <= j && j <= n); ◀
        Contract.Memory.DestTmp();
        Logger logger = new Logger();
        logger.Log("Log "+ (i * j).ToString());
```



- 1. Si el usuario anota los espacios de iteración (fácil), podemos hacer una instrumentación diferente
- 2. Ahora vemos la instrumentación nueva

```
public void ConsumoCuadratico(int n)
  Contract.Requires(n > 0);
  Contract.Memory.Tmp<Logger>(n * n);
  for (int i = 1; i <= n; i++)
     Contract.Memory.IterationSpace(1 <= i && i <= n);</pre>
     for (int j = 1; j <= n; j++)
        Contract.Memory.IterationSpace(1 <= j && j <= n);</pre>
        Contract.Memory.DestTmp();
        Logger logger = new Logger();
        logger.Log("Log "+ (i * j).ToString());
```



- 1. Mostrar de a poco y explicar
- afuera del loop con la cantidad de veces que se debería incrementar

2. En lugar de incrementar el contador después del new, lo incrementamos

- 3. El n * n sale de la herramienta aritmética usando la info de los espacios de iteración
- 4. El if assume también, pero de verificar que el contrato es correcto, en casos más complejos es necesario, pensar en sumas de varias cosas

```
public static int ConsumoCuadratico_Tmp_Logger;
public void ConsumoCuadratico(int n)
  Contract.Requires(n > 0);
  Contract.Memory.Tmp<Logger>(n * n);
  for (int i = 1; i <= n; i++)
  {
     Contract.Memory.IterationSpace(1 <= i && i <= n);</pre>
     for (int j = 1; j <= n; j++)
        Contract.Memory.IterationSpace(1 <= j && j <= n);</pre>
        Contract.Memory.DestTmp();
        Logger logger = new Logger();
        logger.Log("Log "+ (i * j).ToString());
```

Tesis de Licenciatura

23 / 30



1. Mostrar de a poco y explicar

Tesis de Licenciatura

afuera del loop con la cantidad de veces que se debería incrementar

2. En lugar de incrementar el contador después del new, lo incrementamos

- 3. El n * n sale de la herramienta aritmética usando la info de los espacios de iteración
- 4. El if assume también, pero de verificar que el contrato es correcto, en casos más complejos es necesario, pensar en sumas de varias cosas

```
public static int ConsumoCuadratico_Tmp_Logger;
public void ConsumoCuadratico(int n)
  Contract.Requires(n > 0);
  Contract.Memory.Tmp<Logger>(n * n);
  ConsumoCuadratico_Tmp_Logger = 0;
  for (int i = 1; i <= n; i++)
  {
     Contract.Memory.IterationSpace(1 <= i && i <= n);</pre>
     for (int j = 1; j <= n; j++)
        Contract.Memory.IterationSpace(1 <= j && j <= n);</pre>
        Contract.Memory.DestTmp();
        Logger logger = new Logger();
        logger.Log("Log "+ (i * j).ToString());
```



- 1. Mostrar de a poco y explicar
- afuera del loop con la cantidad de veces que se debería incrementar

2. En lugar de incrementar el contador después del new, lo incrementamos

- 3. El n * n sale de la herramienta aritmética usando la info de los espacios de iteración
- 4. El if assume también, pero de verificar que el contrato es correcto, en casos más complejos es necesario, pensar en sumas de varias cosas

Verificación con aritmética no lineal, ejemplo

```
public static int ConsumoCuadratico_Tmp_Logger;
public void ConsumoCuadratico(int n)
  Contract.Requires(n > 0);
  Contract.Memory.Tmp<Logger>(n * n);
  Contract Ensures(ConsumoCuadratico_Tmp_Logger <= n * n); <</pre>
  ConsumoCuadratico_Tmp_Logger = 0;
  for (int i = 1; i <= n; i++)
  {
     Contract.Memory.IterationSpace(1 <= i && i <= n);</pre>
     for (int j = 1; j <= n; j++)
        Contract.Memory.IterationSpace(1 <= j && j <= n);</pre>
        Contract.Memory.DestTmp();
        Logger logger = new Logger();
        logger.Log("Log "+ (i * j).ToString());
```

-Verificación con aritmética no lineal, ejemplo



- 1. Mostrar de a poco y explicar
- afuera del loop con la cantidad de veces que se debería incrementar

2. En lugar de incrementar el contador después del new, lo incrementamos

- 3. El n * n sale de la herramienta aritmética usando la info de los espacios de iteración
- 4. El if assume también, pero de verificar que el contrato es correcto, en casos más complejos es necesario, pensar en sumas de varias cosas

Verificación con aritmética no lineal, ejemplo

```
public static int ConsumoCuadratico_Tmp_Logger;
public void ConsumoCuadratico(int n)
  Contract.Requires(n > 0);
  Contract.Memory.Tmp<Logger>(n * n);
  Contract.Ensures(ConsumoCuadratico_Tmp_Logger <= n * n);</pre>
  ConsumoCuadratico_Tmp_Logger = 0;
  for (int i = 1; i <= n; i++)
  {
     Contract.Memory.IterationSpace(1 <= i && i <= n);</pre>
     for (int j = 1; j <= n; j++)
        Contract.Memory.IterationSpace(1 <= j && j <= n);</pre>
        Contract.Memory.DestTmp();
        Logger logger = new Logger();
        logger.Log("Log "+ (i * j).ToString());
  }
  ConsumoCuadratico_Tmp_Logger += n * n;
```

Tesis de Licenciatura

- 1. Mostrar de a poco y explicar
- afuera del loop con la cantidad de veces que se debería incrementar

2. En lugar de incrementar el contador después del new, lo incrementamos

-Verificación con aritmética no lineal, ejemplo

- 3. El n * n sale de la herramienta aritmética usando la info de los espacios de iteración
- 4. El if assume también, pero de verificar que el contrato es correcto, en casos más complejos es necesario, pensar en sumas de varias cosas

Verificación con aritmética no lineal, ejemplo

```
public static int ConsumoCuadratico_Tmp_Logger;
public void ConsumoCuadratico(int n)
  Contract.Requires(n > 0);
  Contract.Memory.Tmp<Logger>(n * n);
  Contract.Ensures(ConsumoCuadratico_Tmp_Logger <= n * n);</pre>
  ConsumoCuadratico_Tmp_Logger = 0;
  for (int i = 1; i <= n; i++)
  {
     Contract.Memory.IterationSpace(1 <= i && i <= n);</pre>
     for (int j = 1; j <= n; j++)
        Contract.Memory.IterationSpace(1 <= j && j <= n);</pre>
        Contract.Memory.DestTmp();
        Logger logger = new Logger();
        logger.Log("Log "+ (i * j).ToString());
  }
  ConsumoCuadratico_Tmp_Logger += n * n;
  if (n > 0) Contract.Assume(ConsumoCuadratico_Tmp_Logger <= n * n); ◀
  else
             Contract.Assert(false);
```

Ve di cació o con adi metica no final, njemph

- 1. Mostrar de a poco y explicar
- afuera del loop con la cantidad de veces que se debería incrementar

2. En lugar de incrementar el contador después del new, lo incrementamos

- 3. El n * n sale de la herramienta aritmética usando la info de los espacios de iteración
- 4. El if assume también, pero de verificar que el contrato es correcto, en casos más complejos es necesario, pensar en sumas de varias cosas

- Los resultados se ven en el área de mensajes del compilador
- Vimos algunos ejemplos en funcionamiento
 - Verificación correcta
 - Verificación con contrato incorrecto
 - Verificación con una anotación de tiempo de vida incorrecta

Jonathan Tapicer

- 1. Vamos a ver un poco el prototipo funcionando
- 2. Para mostrar cómo se integra en la IDE

- Cuando compilamos en Visual Studio se dispara la verificación
- Los resultados se ven en el área de mensajes del compilador
- Vimos algunos ejemplos en funcionamiento
 - Verificación correcta
 - Verificación con contrato incorrecto
 - Verificación con una anotación de tiempo de vida incorrecta

Jonathan Tapicer

- 1. Vamos a ver un poco el prototipo funcionando
- 2. Para mostrar cómo se integra en la IDE

Verificación, demo

- Cuando compilamos en Visual Studio se dispara la verificación
- Los resultados se ven en el área de mensajes del compilador
- Vimos algunos ejemplos en funcionamiento
 - Verificación correcta
 - Verificación con contrato incorrecto
 - Verificación con una anotación de tiempo de vida incorrecta

Jonathan Tapicer

- 1. Vamos a ver un poco el prototipo funcionando
- 2. Para mostrar cómo se integra en la IDE

- Introducción
- 2 Anotaciones
- 3 Verificación
- 4 Limitaciones y trabajo futuro
- 5 Conclusiones

Tesis de Licenciatura

25 / 30

Estamos limitados por las capacidades de:

- El verificador estático
- La herramienta para resolver operaciones con aritmética no lineal
- → El diseño de las técnicas permite reemplazarlas por herramientas con
- El análisis de tiempo de vida es sobre-aproximado
 - → El análisis de points-to es un problema indecidible
 - Permitimos obviar la verificación para casos en que falla
- - → En un futuro pensamos inferir estas anotaciones

- 1. Limitaciones, tanto por las herramientas usadas como inherentes a las técnicas propuestas
- 2. Como dependemos de otras herramientas (verificador, aritmética), siempre vamos a estar limitados
- 3. Análisis de tiempo de vida, siempre va a ser un problema
- 4. Muchas anotaciones requeridas, inferencia ayudaría

Limitaciones

- Estamos limitados por las capacidades de:
 - El verificador estático
 - La herramienta para resolver operaciones con aritmética no lineal
 - → El diseño de las técnicas permite reemplazarlas por herramientas con mayores o mejores capacidades cuando existan
- El análisis de tiempo de vida es sobre-aproximado
 - → El análisis de points-to es un problema indecidible
 - → Permitimos obviar la verificación para casos en que falla
- Requerimos que el usuario provea todas las anotaciones de tiempo de vida para la verificación, y los espacios de iteración para utilizar la herramienta para aritmética no lineal
 - → En un futuro pensamos inferir estas anotaciones

- 1. Limitaciones, tanto por las herramientas usadas como inherentes a las técnicas propuestas
- 2. Como dependemos de otras herramientas (verificador, aritmética), siempre vamos a estar limitados
- 3. Análisis de tiempo de vida, siempre va a ser un problema
- 4. Muchas anotaciones requeridas, inferencia ayudaría

Estamos limitados por las capacidades de:

- El verificador estático
- La herramienta para resolver operaciones con aritmética no lineal
- → El diseño de las técnicas permite reemplazarlas por herramientas con mayores o mejores capacidades cuando existan
- El análisis de tiempo de vida es sobre-aproximado
 - → El análisis de points-to es un problema indecidible
 - Permitimos obviar la verificación para casos en que falla
- - → En un futuro pensamos inferir estas anotaciones

- 1. Limitaciones, tanto por las herramientas usadas como inherentes a las técnicas propuestas
- 2. Como dependemos de otras herramientas (verificador, aritmética), siempre vamos a estar limitados
- 3. Análisis de tiempo de vida, siempre va a ser un problema
- 4. Muchas anotaciones requeridas, inferencia ayudaría

Limitaciones

- Estamos limitados por las capacidades de:
 - El verificador estático
 - La herramienta para resolver operaciones con aritmética no lineal
 - → El diseño de las técnicas permite reemplazarlas por herramientas con mayores o mejores capacidades cuando existan
- El análisis de tiempo de vida es sobre-aproximado
 - → El análisis de points-to es un problema indecidible
 - → Permitimos obviar la verificación para casos en que falla
- - → En un futuro pensamos inferir estas anotaciones

- 1. Limitaciones, tanto por las herramientas usadas como inherentes a las técnicas propuestas
- 2. Como dependemos de otras herramientas (verificador, aritmética), siempre vamos a estar limitados
- 3. Análisis de tiempo de vida, siempre va a ser un problema
- 4. Muchas anotaciones requeridas, inferencia ayudaría

Limitaciones

- Estamos limitados por las capacidades de:
 - El verificador estático
 - La herramienta para resolver operaciones con aritmética no lineal
 - → El diseño de las técnicas permite reemplazarlas por herramientas con mayores o mejores capacidades cuando existan
- El análisis de tiempo de vida es sobre-aproximado
 - → El análisis de points-to es un problema indecidible
 - → Permitimos obviar la verificación para casos en que falla
- Requerimos que el usuario provea todas las anotaciones de tiempo de vida para la verificación, y los espacios de iteración para utilizar la herramienta para aritmética no lineal
 - → En un futuro pensamos inferir estas anotaciones

8 mile	a der en	ide	
La homa	nina ;	and conditioning	reacions on admitika m final
			mice compliantes per terraminas con-
may11111	n mejar	n capacidades	crants criscar

Limitacio nes

— Pensisiona abate la cesticação para com con que falla gregarimos que el acurio provas todas las acotaciones de tiempo de cida para la cesticação, y las especias de itenación pera utilizar la la ceministra que critentidas no liment.

- - Barthin te rainesco es as mattema interitible

- 1. Limitaciones, tanto por las herramientas usadas como inherentes a las técnicas propuestas
- 2. Como dependemos de otras herramientas (verificador, aritmética), siempre vamos a estar limitados
- 3. Análisis de tiempo de vida, siempre va a ser un problema
- 4. Muchas anotaciones requeridas, inferencia ayudaría

Limitaciones

- Estamos limitados por las capacidades de:
 - El verificador estático
 - La herramienta para resolver operaciones con aritmética no lineal
 - → El diseño de las técnicas permite reemplazarlas por herramientas con mayores o mejores capacidades cuando existan
- El análisis de tiempo de vida es sobre-aproximado
 - → El análisis de points-to es un problema indecidible
 - → Permitimos obviar la verificación para casos en que falla
- Requerimos que el usuario provea todas las anotaciones de tiempo de vida para la verificación, y los espacios de iteración para utilizar la herramienta para aritmética no lineal
 - → En un futuro pensamos inferir estas anotaciones

It is transitive per coulor symmetric or identifies a final are III all a fix starting protect coulomb are per transitive as a consistence of the coulomb are performed as a fixed of the coulomb are a course are obtained as a fixed of the coulomb are a course are obtained as a fixed of the coulomb are a course are obtained as a fixed of the coulomb are a counter as a fixed of the coulomb are obtained as a coulomb are deliverable.

By consistency and a major process that he consistency are collected as a collected of the collected as a c

· -- Er ar from annumn infeir own ar na di no

Limitacio nes

Bata non limitados por las capacidades de:

Breifer de residen

- 1. Limitaciones, tanto por las herramientas usadas como inherentes a las técnicas propuestas
- 2. Como dependemos de otras herramientas (verificador, aritmética), siempre vamos a estar limitados
- 3. Análisis de tiempo de vida, siempre va a ser un problema
- 4. Muchas anotaciones requeridas, inferencia ayudaría

Trabajo futuro

- Incorporar capacidades de inferencia permitiendo al usuario proveer sólo los contratos de consumo y no las anotaciones adicionales actualmente requeridas
 - De las anotaciones de tiempo de vida
 - De los espacios de iteración de los ciclos
- Experimentar con otros verificadores estáticos
 - ESC/Java2 para Java: requeriría implementar la instrumentación para
 - Z3 para Java o .NET: requeriría traducir el código a un lenguaje
- Extender las capacidades del análisis con aritmética no lineal
 - Mejorar la capacidad de cálculo de máximos entre polinomios
 - Evaluar/modificar la herramienta utilizada o reemplazarla por otra
- Mejorar la usabilidad e integración con la IDE
 - Desarrollando un plug-in que autocomplete anotaciones de acuerdo a los

Jonathan Tapicer Tesis de Licenciatura 27 / 30

- 1. Podemos integrar otros trabajos sobre inferencia
- Hay muchos verificadores, Z3 podría integrarse sin ningún trabajo cuando Code Contracts lo soporte
- Ya desarrollamos un plugin para instalar el tool como ext de VS y habilitar/deshabilitar contratos de mem, la idea es extenderlo para asistencia de anotaciones

Trabajo futuro

- Incorporar capacidades de inferencia permitiendo al usuario proveer sólo los contratos de consumo y no las anotaciones adicionales actualmente requeridas
 - De las anotaciones de tiempo de vida
 - De los espacios de iteración de los ciclos
- Experimentar con otros verificadores estáticos
 - ESC/Java2 para Java: requeriría implementar la instrumentación para Java
 - Z3 para Java o .NET: requeriría traducir el código a un lenguaje intermedio que utiliza (Boogie)
- Extender las capacidades del análisis con aritmética no lineal
 - Mejorar la capacidad de cálculo de máximos entre polinomios
 - Evaluar/modificar la herramienta utilizada o reemplazarla por otra
- Mejorar la usabilidad e integración con la IDE
 - Desarrollando un plug-in que autocomplete anotaciones de acuerdo a los

Tesis de Licenciatura
Limitaciones y trabajo futuro
Trabajo futuro

Respects printed is intensing continuous allowed resourcible in continuous chicaches and the continuous chicaches at chicaches at the continuous chicaches at the continuous chicaches at the continuous chicaches at the continuous chicaches at the chicaches at the continuous chicaches at the chi

Trabajo futuro

- 1. Podemos integrar otros trabajos sobre inferencia
- Hay muchos verificadores, Z3 podría integrarse sin ningún trabajo cuando Code Contracts lo soporte
- 3. Ya desarrollamos un plugin para instalar el tool como ext de VS y habilitar/deshabilitar contratos de mem, la idea es extenderlo para asistencia de anotaciones

Trabajo futuro

- Incorporar capacidades de inferencia permitiendo al usuario proveer sólo los contratos de consumo y no las anotaciones adicionales actualmente requeridas
 - De las anotaciones de tiempo de vida
 - De los espacios de iteración de los ciclos
- Experimentar con otros verificadores estáticos
 - ESC/Java2 para Java: requeriría implementar la instrumentación para Java
 - Z3 para Java o .NET: requeriría traducir el código a un lenguaje intermedio que utiliza (Boogie)
- Extender las capacidades del análisis con aritmética no lineal
 - Mejorar la capacidad de cálculo de máximos entre polinomios
 - Evaluar/modificar la herramienta utilizada o reemplazarla por otra
- Mejorar la usabilidad e integración con la IDE
 - Desarrollando un plug-in que autocomplete anotaciones de acuerdo a los contratos existentes

Tesis de Licenciatura			
Limitaciones y trabajo futuro			
Trabajo futuro			

I benner confidente fristende predicted description records be creation for commany who contains entitled extraduction of the confidence is believed by the confidence of the confidence of the settled way to be confidence of the confidence of the

Trabajo futuro

- 1. Podemos integrar otros trabajos sobre inferencia
- Hay muchos verificadores, Z3 podría integrarse sin ningún trabajo cuando Code Contracts lo soporte
- 3. Ya desarrollamos un plugin para instalar el tool como ext de VS y habilitar/deshabilitar contratos de mem, la idea es extenderlo para asistencia de anotaciones

Trabajo futuro

- Incorporar capacidades de inferencia permitiendo al usuario proveer sólo los contratos de consumo y no las anotaciones adicionales actualmente requeridas
 - De las anotaciones de tiempo de vida
 - De los espacios de iteración de los ciclos
- Experimentar con otros verificadores estáticos
 - ESC/Java2 para Java: requeriría implementar la instrumentación para Java
 - Z3 para Java o .NET: requeriría traducir el código a un lenguaje intermedio que utiliza (Boogie)
- Extender las capacidades del análisis con aritmética no lineal
 - Mejorar la capacidad de cálculo de máximos entre polinomios
 - Evaluar/modificar la herramienta utilizada o reemplazarla por otra
- Mejorar la usabilidad e integración con la IDE
 - Desarrollando un plug-in que autocomplete anotaciones de acuerdo a los contratos existentes

Jonathan Tapicer Tesis de Licenciatura

| Security or printers to informing conditions of a conditional conditions or conditi

Trabajo futuro

- 1. Podemos integrar otros trabajos sobre inferencia
- 2. Hay muchos verificadores, Z3 podría integrarse sin ningún trabajo cuando Code Contracts lo soporte
- 3. Ya desarrollamos un plugin para instalar el tool como ext de VS y habilitar/deshabilitar contratos de mem, la idea es extenderlo para asistencia de anotaciones

Índice

- Introducción
- 2 Anotaciones
- 3 Verificación
- 4 Limitaciones y trabajo futuro
- 5 Conclusiones

Conclusiones

- Presentamos un conjunto de algoritmos y técnicas para verificar el consumo de memoria de un programa
 - Haciendo uso de las capacidades de análisis de un verificador estático
 - Integrando herramientas externas para incrementar las capacidades de análisis
- Implementamos un prototipo
 - Para .NET, usando el verificador estático de Code Contracts
 - Con integración en la IDE
- Evaluamos su uso bajo diferentes condiciones
 - Identificando las limitaciones
- Creemos que la solución ideal debe hacer un uso combinado de inferencia y verificación
 - Inferir anotaciones y contratos fácilmente deducibles
 - Verificar contratos complejos anotados por el usuario
- El trabajo presentado es un buen punto de partida para una solución utilizable en un entorno real para obtener un certificado del consumo de memoria

- 1. Ideal: inferir lo fácil, verificar lo dificíl que anota el usuario
- 2. Bueno punto de partida para seguir trabajando

Conclusiones

 Presentamos un conjunto de algoritmos y técnicas para verificar el consumo de memoria de un programa

- Haciendo uso de las capacidades de análisis de un verificador estático
- Integrando herramientas externas para incrementar las capacidades de análisis
- Implementamos un prototipo
 - Para .NET, usando el verificador estático de Code Contracts
 - Con integración en la IDE
- Evaluamos su uso bajo diferentes condiciones
 - Identificando las limitaciones
- Creemos que la solución ideal debe hacer un uso combinado de inferencia y verificación
 - Inferir anotaciones y contratos fácilmente deducibles
 - Verificar contratos complejos anotados por el usuario
- El trabajo presentado es un buen punto de partida para una solución utilizable en un entorno real para obtener un certificado del consumo de memoria

- 1. Ideal: inferir lo fácil, verificar lo dificíl que anota el usuario
- 2. Bueno punto de partida para seguir trabajando

Conclusiones

 Presentamos un conjunto de algoritmos y técnicas para verificar el consumo de memoria de un programa

- Haciendo uso de las capacidades de análisis de un verificador estático
- Integrando herramientas externas para incrementar las capacidades de análisis
- Implementamos un prototipo
 - Para .NET, usando el verificador estático de Code Contracts
 - Con integración en la IDE
- Evaluamos su uso bajo diferentes condiciones
 - Identificando las limitaciones
- Creemos que la solución ideal debe hacer un uso combinado de inferencia y verificación
 - Inferir anotaciones y contratos fácilmente deducibles
 - Verificar contratos complejos anotados por el usuario
- El trabajo presentado es un buen punto de partida para una solución utilizable en un entorno real para obtener un certificado del consumo de memoria

Ganchaines

Bernsteines conjunte à algebra y titolor providered

consent en mois à ce raggene

a district ne la terrapier de destruction providered

a district ne la terrapier de destruction de des profits

india

indi

- 1. Ideal: inferir lo fácil, verificar lo dificíl que anota el usuario
- 2. Bueno punto de partida para seguir trabajando

Conclusiones

 Presentamos un conjunto de algoritmos y técnicas para verificar el consumo de memoria de un programa

- Haciendo uso de las capacidades de análisis de un verificador estático
- Integrando herramientas externas para incrementar las capacidades de análisis
- Implementamos un prototipo
 - Para .NET, usando el verificador estático de Code Contracts
 - Con integración en la IDE
- Evaluamos su uso bajo diferentes condiciones
 - Identificando las limitaciones
- Creemos que la solución ideal debe hacer un uso combinado de inferencia y verificación
 - Inferir anotaciones y contratos fácilmente deducibles
 - Verificar contratos complejos anotados por el usuario
- El trabajo presentado es un buen punto de partida para una solución utilizable en un entorno real para obtener un certificado del consumo de memoria

Corchain or a contract of algorithms of their per solution of contract to a way in a company of their per solution of contract to a way in a contract to the contract to a contract to a

- 1. Ideal: inferir lo fácil, verificar lo dificíl que anota el usuario
- 2. Bueno punto de partida para seguir trabajando

Conclusiones

 Presentamos un conjunto de algoritmos y técnicas para verificar el consumo de memoria de un programa

- Haciendo uso de las capacidades de análisis de un verificador estático
- Integrando herramientas externas para incrementar las capacidades de análisis
- Implementamos un prototipo
 - Para .NET, usando el verificador estático de Code Contracts
 - Con integración en la IDE
- Evaluamos su uso bajo diferentes condiciones
 - Identificando las limitaciones
- Creemos que la solución ideal debe hacer un uso combinado de inferencia y verificación
 - Inferir anotaciones y contratos fácilmente deducibles
 - Verificar contratos complejos anotados por el usuario
- El trabajo presentado es un buen punto de partida para una solución utilizable en un entorno real para obtener un certificado del consumo de memoria

Presenta mos en conjunto de algoritmos y técnicas para verificar al
consumo de memodo de sus programa.

Heciards are de las capacidades de arálliós de ar confrad er cadicia
 Inceparda horamientas encoras para incomercia las capacidades de
arálliós
 Implementa mos are postotipo

Por a .NET, usuado al melicular esclaire de Cade Contracto
 Car integrada en la 10 E
 Evaluate en a casa la contracto contracto
 Mandicanto la Ministractoro

Conclusiones

Corema que la solición ideal de la lacer un una combinato de inferencia y verificación
 Inferencia y verificación
 Inferencia y concursos Siclmenta deducidos

Welk concerne una complejeramentale per el musele

 Hera la jo presentado es un hera pareo de prelia para una solución

 difera la en en entorno real para obtener un contificado del consenso
de memoria.

- 1. Ideal: inferir lo fácil, verificar lo dificíl que anota el usuario
- 2. Bueno punto de partida para seguir trabajando

Fin **I**

¿Preguntas?

Jonathan Tapicer Tesis de Licenciatura