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 Tapicer Tesis de Licenciatura Febrero de 2011 1 / 30

Tesis de Licenciatura

2011-02-10

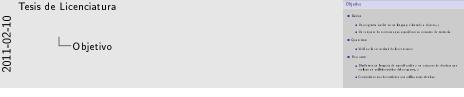
Especificación y vedicación mod ular de consumo de memo do la laculta fraisce successiva de la consumo de memo do laculta fraisce successiva de laculta de

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

notaciones Verificación Limitaciones y trabajo futuro Conclusiones

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



- 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

Jonathan Tapicer

Tesis de Licenciatura

Febrero de 2011

- 1 Introducción
- Anotaciones
- Verificación
- 4 Limitaciones y trabajo futuro
- 5 Conclusiones





Tesis de Licenciatura

-Índice



- 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

1 Introducción

Introducción

Índice

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

notaciones Verificación Limitaciones y trabajo futuro Conclus

Verificación

Introducción

- ¿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

- 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

notaciones Verificación Limitaciones y trabajo futuro Conclusio

Verificación

Introducción

- ¿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

Jonathan Tapicer

2011-02-10

aciones Verificación Limitaciones y trabajo futuro

Verificación

Introducción

- ¿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

Jonathan Tapicer

2011-02-10

Introducción Anotaciones Verificación Limitaciones y trabajo futuro Conclusiones

Overview del trabajo (1)

- 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)

Contract.Requires(n >= 0);
Contract.Ensures(Contract.Result() * Contract.Result() == n);
Contract.Ensures(Contract.Result() >= 0);

return Math.Sqrt(n);
}
```

```
Tesis de Licenciatura

Introducción

Overview del trabajo (1)

Overview del trabajo (1)

Overview del trabajo (1)
```

- 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 Anotaciones Verificación Limitaciones y trabajo futuro

Overview del trabajo (2)

- 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)

Tesis de Licenciatura

Introducción

Introducción

Overview del trabajo (2)

Overview del trabajo (2)

- 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#

Jonathan Tapicer

Tesis de Licenciatura

Febrero de 2011

Introducción Anotaciones Verificación Limitaciones y trabajo futuro Conclusione

Overview del trabajo (2)

- 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)

Tesis de Licenciatura
Location
Overview del trabajo (2)

- Overview deltrabajo (2)
- Proposemos as conjusto de titrakas para serifica e la cometikad de las anotaciones a Apopladas es as antificados escários
 - Une de electrica de antifuis de ciempo de cida
 Torrier de ce casera de esconida del conficer animetrica en fenal
- g Constraines an prototipo de la nolación propuesta para . NET extendiendo Code Contracts
- Barra in organista con la IDE | Maral Sandia |
 Dispusita Bital de la marcia con para antalia escádica e in secono maraldo
 Verificado escádica que requirse mora animencia la marte la monocacida.

1. Herramientas para análisis estático: CCI

2011-02-10

- 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#

Jonathan Tapicer

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

Tesis de Licenciatura 2011-02-10 - Anotaciones └─Índice

M Arotaciona

Jonathan Tapicer

Tesis de Licenciatura

Febrero de 2011

- 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
 - Asumimos que los objetos son recolectados al final del método
 - Sólo tiene en cuenta objetos **creados** por el método analizado y los
 - Objetos temporales: no necesarios después de la ejecución del método,
 - Objetos residuales: exceden el tiempo de vida del método, deben seguir

- 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
 - Asumimos que los objetos son recolectados al final del método
 - Sólo tiene en cuenta objetos **creados** por el método analizado y los
 - Objetos temporales: no necesarios después de la ejecución del método,
 - Objetos residuales: exceden el tiempo de vida del método, deben seguir

- 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)

2011-02-10

Modelando el uso de memoria

- 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
 - Objetos temporales: no necesarios después de la ejecución del método,
 - Objetos residuales: exceden el tiempo de vida del método, deben seguir

- 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)

- 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
 - Objetos temporales: no necesarios después de la ejecución del método,
 - Objetos residuales: exceden el tiempo de vida del método, deben seguir

- 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)

lucción Anotaciones Verificación L

Modelando el uso de memoria

- 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 finalize la ejecución del método.

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

Jonathan Tapicer

2011-02-10

Modelando el uso de memoria

- 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

- · Assertion as the latest and recolors to all final fall mitrals
- 5 th circum company in jour creation par el mitada analizada y la · Objects serperales ne reconsina deputa de la operación del mando
- m panter eliminer te memeria · Oblema resituales: esceten el rieman de cita del microto, deben servi-
- 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)

Jonathan Tapicer

Tesis de Licenciatura

Febrero de 2011

9 / 30

2011-02-10

```
class IntLinkedList
       private Node Head;
       public void PushFront(Node node)
          Logger logger = new Logger();
          node.Next = this.Head;
          this. Head = node:
          logger.Log("PushFront done");
13
       public void Fill(int n)
15
          for (int i = 1; i <= n; i++)</pre>
             Node node = new Node(i);
             this. PushFront(node);
20
21
```



- 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

```
class IntLinkedList
       private Node Head;
       public void PushFront(Node node)
                                                      temporal
          Logger logger = new Logger();
          node. Next = this. Head;
          this. Head = node;
          logger.Log("PushFront done");
13
       public void Fill(int n)
15
          for (int i = 1; i <= n; i++)</pre>
                                                     residuales
              Node node = new Node(i);
              this. PushFront(node);
20
21
```

Tesis de Licenciatura -Anotaciones 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++)
     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

Anotaciones, ejemplo

```
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);
```

Jonathan Tapicer

Tesis de Licenciatura

Febrero de 2011 11 / 30 2011-02-10

-Anotaciones, ejemplo

Tesis de Licenciatura

-Anotaciones



- 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);
```

Jonathan Tapicer

Tesis de Licenciatura

Febrero de 2011

11 / 30



- 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

Anotaciones, ejemplo

```
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);
```

Jonathan Tapicer

Tesis de Licenciatura

Febrero de 2011

11 / 30



- 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);
```

Jonathan Tapicer

Tesis de Licenciatura

Febrero de 2011

11 / 30



- 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

Contract Memory DestRsd(Contract Memory This);

Jonathan Tapicer

Node node = new Node(i); this.PushFront(node);

Tesis de Licenciatura

Febrero de 2011

11 / 30



- 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.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

oducción Anotaciones Verificación Limitaciones y trabajo futuro Conclusion

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

- 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

oducción **Anotaciones** Verificación Limitaciones y trabajo futuro Conclusi

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

- 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

oducción Anotaciones Verificación Limitaciones y trabajo futuro

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

- 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

oducción **Anotaciones** Verificación Limitaciones y trabajo futuro

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

- 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

Jonathan Tapicer

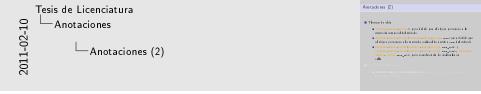
2011-02-10

oducción Anotaciones Verificación Limitaciones y trabajo futuro Conclusio

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



- 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

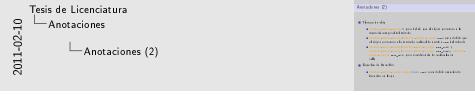
Jonathan Tapicer Tesis de Licenciatura Febrero de 2011

oducción **Anotaciones** Verificación Limitaciones y trabajo futuro Coi

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



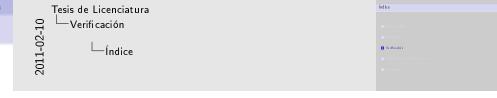
- 1. Tiempo de vida: para anotar cuáles objetos son tmp y cuáles rsd, y qué pasa con los objetos rsd de un método invocado
- 2. Espacios de iteración: para entender la cantidad de veces que ocurren los consumos de memoria adentro de loops

Jonathan Tapicer Tesis de Licenciatura Febrero de 2011 13 / 30

otaciones **Verificación** Limitaciones y trabajo futuro Conclusion

Índice

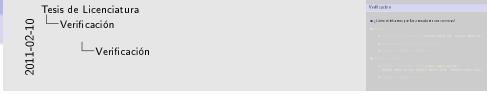
- 1 Introducción
- 2 Anotaciones
- 3 Verificación
- 4 Limitaciones y trabajo futuro
- 5 Conclusiones



otaciones **Verificación** Limitaciones y trabajo futuro Conclusi

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
- 2. 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

notaciones **Verificación** Limitaciones y trabajo futuro Conclusi

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

Tesis de Licenciatura

Verificación

***Communication**

***Commun

- 1. Esta parte es el desarrollo central del trabajo
- 2. 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

taciones **Verificación** Limitaciones y trabajo futuro Concl

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

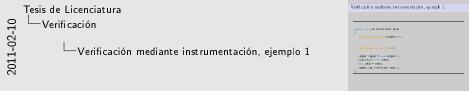
- 1. Esta parte es el desarrollo central del trabajo
- 2. Verificar que el consumo que el usuario dice que tiene su programa sea correcto
- 3. Hay diferentes cosas a verificar, cada una la resolvemos de diferente forma
- 4. Instrumentación: generar un programa equivalente pero con código insertado que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

Jonathan Tapicer Tesis de Licenciatura Febrero de 2011 15 / 30

oducción Anotaciones

es Verificación

```
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");
```



1. Mostrar de a poco y explicar la instrumentación

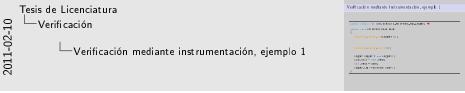
Jonathan Tapicer

Tesis de Licenciatura

Febrero de 2011

16 / 30

```
public static int IntLinkedList_PushFront_Tmp_Logger; 
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");
```



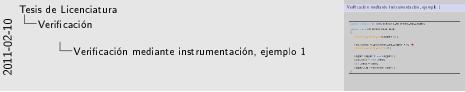
oducción And

otaciones

aciones y trabajo futuro

Verificación

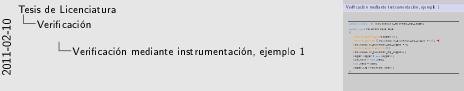
```
public static int IntLinkedList_PushFront_Tmp_Logger;
public void PushFront(Node node)
  Contract.Memory.Tmp<Logger>(1);
  IntLinkedList_PushFront_Tmp_Logger = 0;
  Contract Memory DestTmp();
  Logger logger = new Logger();
  node.Next = this.Head:
  this.Head = node;
  logger.Log("PushFront done");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



2011-02-10

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

```
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 DestTmp();
  IntLinkedList_PushFront_Tmp_Logger++;
  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):
```

Jonathan Tapicer

Tesis de Licenciatura

Febrero de 2011

17 / 30

Tesis de Licenciatura -Verificación

2011-02-10

Verificación mediante instrumentación, ejemplo 2



- 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

2011-02

```
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++)
     Contract Memory DestRsd(Contract Memory This);
     Node node = new Node(i):
     this. PushFront(node):
```

Jonathan Tapicer

Tesis de Licenciatura

Febrero de 2011

17 / 30

Tesis de Licenciatura -Verificación Verificación mediante instrumentación, ejemplo 2



- 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

cción Anotaciones Verificación

ación Limitaciones y trabajo futur

2011-02

Verificación mediante instrumentación, ejemplo 2

```
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++)
     Contract Memory DestRsd(Contract Memory This);
     Node node = new Node(i):
     this. PushFront(node):
```

Jonathan Tapicer

Tesis de Licenciatura

Febrero de 2011

17 / 30

Tesis de Licenciatura Verificación Verificación mediante instrumentación, ejemplo 2



- 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

ción Anotaciones Verificación

ficación Limitaciones y trabajo futur

2011-02-10

Verificación mediante instrumentación, ejemplo 2

```
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++)
     Contract Memory DestRsd(Contract Memory This);
     Node node = new Node(i):
     this. PushFront(node):
```

Jonathan Tapicer

Tesis de Licenciatura

Febrero de 2011

17 / 30

Tesis de Licenciatura Verificación Verificación mediante instrumentación, ejemplo 2



- 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

2011-02

```
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):
```

Jonathan Tapicer

Tesis de Licenciatura

Febrero de 2011

17 / 30

Tesis de Licenciatura -Verifi cación Verificación mediante instrumentación, ejemplo 2



- 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

2011-02-10

```
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):
```

Jonathan Tapicer

Tesis de Licenciatura

Febrero de 2011

17 / 30

Tesis de Licenciatura -Verifi cación Verificación mediante instrumentación, ejemplo 2



- 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

2011-02

Verificación mediante instrumentación, ejemplo 2

```
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):
```

Jonathan Tapicer

Tesis de Licenciatura

Febrero de 2011

17 / 30

Tesis de Licenciatura -Verifi cación

Verificación mediante instrumentación, ejemplo 2



- 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

2011-02

```
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);
```

Jonathan Tapicer

Tesis de Licenciatura

Febrero de 2011

17 / 30

Tesis de Licenciatura -Verifi cación Verificación mediante instrumentación, ejemplo 2



- 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);
  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;
```

Jonathan Tapicer

Tesis de Licenciatura

Febrero de 2011

17 / 30

Tesis de Licenciatura -Verifi cación

2011-02

Verificación mediante instrumentación, ejemplo 2



- 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);
  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;
```

Jonathan Tapicer

Tesis de Licenciatura

Febrero de 2011

17 / 30

Tesis de Licenciatura L Verificación

2011-02

└─Verificación mediante instrumentación, ejemplo 2



- 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);
  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;
```

Jonathan Tapicer

Tesis de Licenciatura

Febrero de 2011

17 / 30

Tesis de Licenciatura -Verifi cación

Verificación mediante instrumentación, ejemplo 2



- 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

2011-02-10

18 / 30

- 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
- A partir del PTG hacemos un análisis de escape de objetos para ver
- Verificamos
 - Que los objetos escapen del método si y sólo si están anotados como
 - Que los objetos residuales escapen del método a través de la expresión

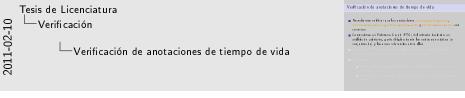
Tesis de Licenciatura Verificación de anotaciones de tiempo de vida Verifi cación 🙀 Necesita mos ve ificar y se la sanotaciones 😁 Verificación de anotaciones de tiempo de vida

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

Jonathan Tapicer Tesis de Licenciatura Febrero de 2011

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
- Verificamos
 - Que los objetos escapen del método si y sólo si están anotados como
 - Que los objetos residuales escapen del método a través de la expresión



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



18 / 30

2011-02-10

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
 - Que los objetos residuales escapen del método a través de la expresión

Tesis de Licenciatura Verificación de anotaciones de tiempo de vida · Verifi cación Necesita mos ve ificar y se la sanotaciones co Constraimos de Poieta-to Galah IPTGI del mitodo haciento de anilidade points-to, gufo dirigido donde los nodos son objetos (o conjuntos del vilos a con referencias entre ellos Verificación de anotaciones de tiempo de vida 🖪 A partir del PTG facernos an análisis de escape de objetos para w sat objetos exceden el tiempo de vida del mitodo.

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

Jonathan Tapicer

Tesis de Licenciatura

Febrero de 2011

18 / 30

oducción Anotaciones **Verificación** Limitaciones y trabajo futuro

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

- Verificación

- Verificación

- Verificación de anotaciones de tiempo de vida

- Verificación de anotación de anotaciones de tiempo de vida d

Verificación de anotaciones de tiempo de vida

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

Tesis de Licenciatura

2011-02-10

Jonathan Tapicer Tesis de Licenciatura Febrero de 2011 18 / 30

```
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);
13
```

- 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

Jonathan Tapicer

Tesis de Licenciatura

Febrero de 2011

19 / 30

Verificación de anotaciones de tiempo de vida, ejemplo

```
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);
13
```

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

■ Points-to Graph:



■ El objeto de la línea 10 escapa

Jonathan Tapicer

Tesis de Licenciatura

Febrero de 2011

19 / 30

Tesis de Licenciatura -Verifi cación

-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

```
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);
13
```

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



■ El objeto de la línea 10 escapa ⇒ es residual, DestRsd es correcta

Jonathan Tapicer

Tesis de Licenciatura

Febrero de 2011

19 / 30

Tesis de Licenciatura -Verifi cación

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 Limita

nitaciones y trabajo futuro

Verificación de anotaciones de tiempo de vida, ejemplo

```
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);
    }
}</pre>
```

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

■ Points-to Graph:

| Head | Node (10) | Next | ⇒ escapa por this

■ El objeto de la línea 10 escapa ⇒ es residual, DestRsd es correcta

Jonathan Tapicer

Tesis de Licenciatura

Febrero de 2011

19 / 30

Tesis de Licenciatura Verificación

Verificación de anotaciones de tiempo de vida, ejemplo

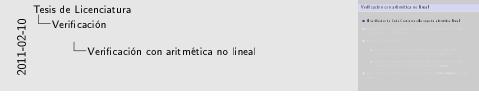


- 1 Método Fill anotado
- 2. Vamos a ver la info que obtener del análisis de points-to y escape
- 3. Y cómo usamos esa info para verificar
- 4. Con la misma info podemos verificar que AddTmp y AddRsd sean correctos, es decir que los objetos residuales de un método invocado se conviertan en lo tmp o rsd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

Anotaciones Verificación Limitaciones y trabajo futuro C

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



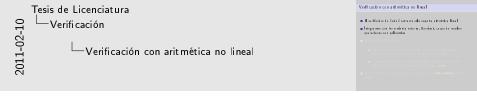
- 1. Aclarar que otros verificadores estáticos también manejan sólo aritmética lineal
- 2. En gral, aparece una multiplicación e ignoran, no pueden probar
- 3. En consumo de memoria aparecen fácil las multiplicaciones, pensar en loop anidado, parecido a análisis de complejidad temporal

Jonathan Tapicer Tesis de Licenciatura Febrero de 2011 20 / 30

notaciones **Verificación** Limitaciones y trabajo futuro Con

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



- 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

Jonathan Tapicer Tesis de Licenciatura Febrero de 2011 20 / 30

Anotaciones Verificación Limitaciones y trabajo futuro (

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

Tesis de Licenciatura

Verificación

Il des discrete Control de seguent e invelación los de la lega man con la discrete control de seguent e invelación los des control de seguent e la lega man con la discrete de seguent e la lega man con la lega man control de seguent e la lega man con la lega man con discrete e mode encodance e el la lega man con la lega man con discrete el la lega man con la l

- 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

Jonathan Tapicer

2011-02-10

otaciones Verificación Limitaciones y trabajo futuro

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

Jonathan Tapicer Tesis de Licenciatura Febrero de 2011 20 / 30

```
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 <= n; j++)
             Contract Memory DestTmp();
             Logger logger = new Logger();
             logger.Log("Log " + (i * j).ToString());
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



Verificación con aritmética no lineal, ejemplo

Verificación

```
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 <= n; j++)
             Contract Memory DestTmp();
             Logger logger = new Logger();
             logger.Log("Log " + (i * j).ToString());
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

Tesis de Licenciatura Verificación con arit mética no lineal, ejemplo -Verifi cación Verificación con aritmética no lineal, ejemplo • d coman date excesses o



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

Tesis de Licenciatura

Verificación

Verificación con aritmética no lineal, ejemplo

Vedicación con admitica no final, ejemplo

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

Tesis de Licenciatura -Verifi cación Verificación con aritmética no lineal, ejemplo Verificación con arit mética no lineal, ejemplo · Il reiteater de Cute Companyon excapaz de reiteach car le inscrepensais - Contracting discourts and associated as a terral property accompanies to exist and

2011-02-10

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++)</pre>
     for (int j = 1; j <= n; j++)
        Contract Memory DestTmp();
        Logger logger = new Logger();
        logger.Log("Log "+ (i * j).ToString());
```

Jonathan Tapicer

Tesis de Licenciatura

Febrero de 2011

22 / 30

Tesis de Licenciatura -Verifi cación

Verificación con aritmética no lineal, ejemplo



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

2011-02

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

Jonathan Tapicer

Tesis de Licenciatura

Febrero de 2011

22 / 30

Tesis de Licenciatura -Verifi cación

Verificación con aritmética no lineal, ejemplo

Verificación con arit mética no lineal, ejemplo Para poder cakalar la carridad de objetos comporales necesarios necesiram es ayuda della mario, del para extender los copacion de icoación: STREET, CONTINUE TO THE REST OF THE

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

2011-02

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

Jonathan Tapicer

Tesis de Licenciatura

Febrero de 2011

22 / 30

Tesis de Licenciatura -Verifi cación

Verificación con aritmética no lineal, ejemplo



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

Verificación con aritmética no lineal, ejemplo

```
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);</pre>
        Contract Memory DestTmp();
        Logger logger = new Logger();
        logger.Log("Log "+ (i * j).ToString());
```

Jonathan Tapicer

Tesis de Licenciatura

Febrero de 2011

23 / 30

Tesis de Licenciatura -Verifi cación

Verificación con aritmética no lineal, ejemplo

```
Verificación con arit mética no lineal, ejemplo
        togger togger * ... togger : ;
```

- 1. Mostrar de a poco y explicar
- 2. En lugar de incrementar el contador después del new, lo incrementamos afuera del loop con la cantidad de veces que se debería incrementar
- 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());
```

Jonathan Tapicer

Tesis de Licenciatura

Febrero de 2011

23 / 30



- 1. Mostrar de a poco y explicar
- 2. En lugar de incrementar el contador después del new, lo incrementamos afuera del loop con la cantidad de veces que se debería incrementar
- 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);
  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());
```

Jonathan Tapicer

Tesis de Licenciatura

Febrero de 2011

23 / 30

Tesis de Licenciatura -Verificación



- 1. Mostrar de a poco y explicar
- 2. En lugar de incrementar el contador después del new, lo incrementamos afuera del loop con la cantidad de veces que se debería incrementar
- 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);
  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());
```

Jonathan Tapicer

Tesis de Licenciatura

Febrero de 2011

23 / 30

Tesis de Licenciatura -Verificación



- 1. Mostrar de a poco y explicar
- 2. En lugar de incrementar el contador después del new, lo incrementamos afuera del loop con la cantidad de veces que se debería incrementar
- 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;
```

Jonathan Tapicer

Tesis de Licenciatura

Febrero de 2011

23 / 30

Tesis de Licenciatura -Verificación



- 1. Mostrar de a poco y explicar
- 2. En lugar de incrementar el contador después del new, lo incrementamos afuera del loop con la cantidad de veces que se debería incrementar
- 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);
  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); <</pre>
  else
             Contract.Assert(false);
```

Jonathan Tapicer

Tesis de Licenciatura

Febrero de 2011

23 / 30

Tesis de Licenciatura LVerificación



- 1. Mostrar de a poco y explicar
- 2. En lugar de incrementar el contador después del new, lo incrementamos afuera del loop con la cantidad de veces que se debería incrementar
- 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

taciones **Verificación** Limitaciones y trabajo futuro

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

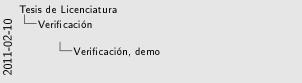




- 1. Vamos a ver un poco el prototipo funcionando
- 2. Para mostrar cómo se integra en la IDE

Jonathan Tapicer Tesis de Licenciatura Febrero de 2011 24 / 30

- 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



Verificación, demo 🛮 Caunto compilamos en Vicad Statio se dispusa la serificación

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

Jonathan Tapicer Tesis de Licenciatura 24 / 30

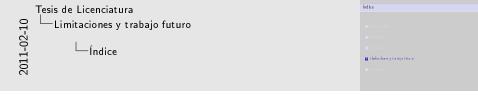
Verificación, demo

- 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

- Tesis de Licenciatura Verificación, demo 2011-02-10 Verifi cación 💣 Caanto compilamos en Vistal Statio se dispasa la serificación Vimos alganos ejemplos en funciona miento -Verificación, demo • Verificación comeca · Verificación com una ampación de ciem os de sida incorrecca.
 - 1. Vamos a ver un poco el prototipo funcionando
- 2. Para mostrar cómo se integra en la IDE

Jonathan Tapicer Tesis de Licenciatura Febrero de 2011 24 / 30

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



ión Anotaciones Verificación **Limitaciones y trabajo futuro** Cond

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

Tesis de Licenciatura

Limitaciones y trabajo futuro

Limitaciones

Limitaciones

- 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

Jonathan Tapicer

Anotaciones Verificación Limitaciones y trabajo futuro Concl

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

Tesis de Licenciatura

Limitaciones y trabajo futuro

Limitaciones y trabajo futuro

Limitaciones

Limitaciones

- 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

Jonathan Tapicer

Anotaciones Verificación Limitaciones y trabajo futuro Co

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

Tesis de Licenciatura

Limitaciones y trabajo futuro

Limitaciones y trabajo futuro

Limitaciones

Unidade de la constituta d

- 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

Jonathan Tapicer

notaciones Verificación Limitaciones y trabajo futuro Concl

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

Tesis de Licenciatura

Limitaciones y trabajo futuro

Limitaciones y trabajo futuro

Limitaciones y trabajo futuro

- Builda de la districtura por la expediente en admitis a fund
- Builda de la districtura con admitis a fund
- Builda de la districtura con admitis a fund
- Builda de la districtura con a confidence en
- Builda de primero en acceptante confidence
- Builda de primero en acceptante confidence
- Builda de primero en acceptante funditis
- Builda de primero en acceptante funditis por una consequente funditis
- Builda de primero en acceptante funditis por una consequente funditis por una consequente

- 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

Jonathan Tapicer

otaciones Verificación <mark>Limitaciones y trabajo futuro C</mark>onclu

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

Tesis de Licenciatura
Limitaciones y trabajo futuro
Limitaciones

2011-02-10

Budhistetiengote siteen untwappoinuto
 Budhistetjanissen oner publima interitible
 Pomisima shirida orificación and cancer an efilia

Limita cio nes

Repartinos que el asur do prover todas las anotaciones de tiempo de dia para la se discación, y los espacios de terración pour atilizar la les semiente para artimitica no lineal

- 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

Jonathan Tapicer

Limitaciones y trabajo futuro

Esta mos li mitados por las capacidades de

2011-02-10

- Hanifeir de tiempo de vida en sobseupsodmado · -- Baribis de criscoso es as crablema indecidible · -- Permisimo a cheixe la conficación para cama en case falla
- Requerimos que el usua do provez todas las anotaciones de tiempo de vida para la verificación, y los espacios de iteración para atilicar la bernehets ann eitmitke milimal - - Er er freez rennemn inferir oan ar oa di ren
- 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

Jonathan Tapicer

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 lava
 - 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

Trabajo futuro

- 1. Podemos integrar otros trabajos sobre inferencia
- 2. 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 contratos existentes

Tesis de Licenciatura

Limitaciones y trabajo futuro

Trabajo futuro

Trabajo futuro

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

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

- 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

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

2011-02-10

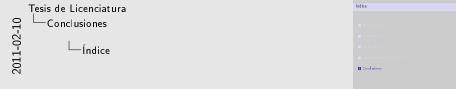


Tra bajo fut uro

- 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

- 1 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 da mamaria

Tesis de Licenciatura

Conclusiones

Indicator provident de la confidencia del confidencia de la confidencia de la confidencia del confidencia de la confidencia de la confidencia de la confidencia del confidencia d

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

Jonathan Tapicer

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

Tesis de Licenciatura

Conclusiones

Branctons a recipito de dipulmo y troine par unida el dipulmo de la constitución de entida el dipulmo de entida el d

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

Jonathan Tapicer

Tesis de Licenciatura

Febrero de 2011

29 / 30

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

Tesis de Licenciatura

Conclusiones

Paratos na recipira de deplina y técino pro voite en de la final de la final

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

Jonathan Tapicer

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

Tesis de Licenciatura

Conclusiones

**Conclusiones*

**Conclusiones*

Conclusiones*

**Conclusiones*

Conclusiones*

**Conclusiones*

**Concl

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

2011-02-10

Jonathan Tapicer Tesis de Licenciatura Febrero de 2011 29 / 30

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

Conclusiones Tesis de Licenciatura El Presenta mos un conjunto de algorit mos y titoricas para serificar el Conclusiones · Haciento ano de las caracidades de arálisis de ar cerificador espícico a lecegando herramientas espersas para incrementar los capacidades d Indenestanos es extotico Para . NET . apar de el reció cador encicies de Code Concraces Con incognación en la IDE -Conclusiones Evalua mos sa aso hajo dife wates condiciones • Identificante las limitaciones Creemos que la solución ideal de be bacer un uso combinado d • life i arradice e y coreaco Selmero delicible · Verificar contrato e complejos anotados por el assario El e u la jo presentado es un buen punto de partida para una solución stifes He ex an estorio real sara obtener as certificado del coma mo

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

2011-02-10

Jonathan Tapicer Tesis de Licenciatura Febrero de 2011 29 / 30

Conclusiones

Tesis de Licenciatura 2011-02-10 Conclusiones └─Fin ■

¿Preguntas?

¿Preguntas?

Fin **I**