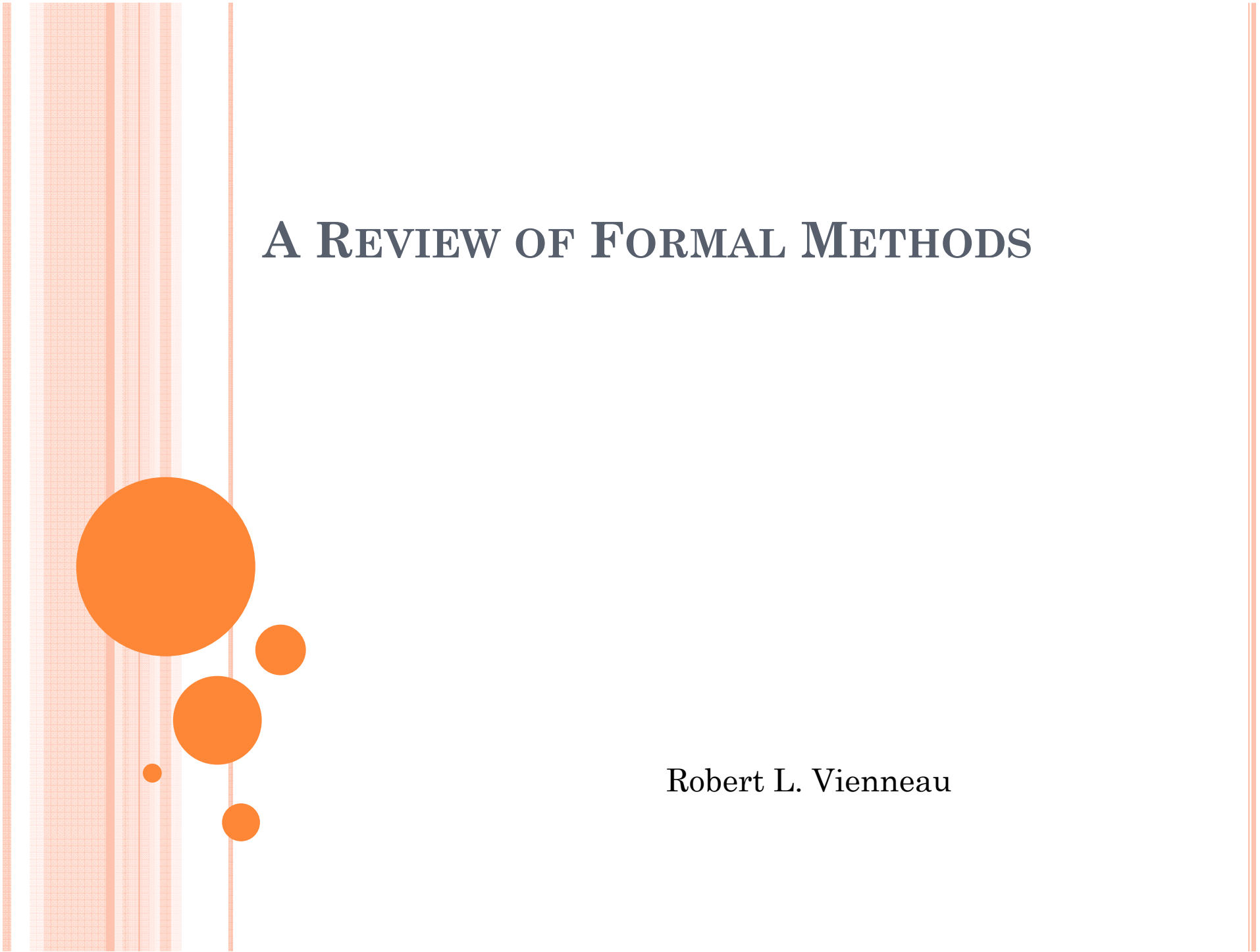

A Review of Formal Methods

Robert L. Vienneau

INTRODUCTION

- 1970년, 구조화 프로그래밍 혁명 때 소프트웨어 엔지니어들은 프로그램 디자인에서의 확실한 개념의 유무에 따라 더 낳은 프로그램이 발생한다고 확신하게 되었고, 프로그래밍 방법론에 있어서 최고의 방법을 찾기 위한 노력이 시작 되었다.

- 이후 최고의 소프트웨어 개발방법에 초점을 맞춘 지속적인 변화가 시작되었다.

- Formal Methods 는 이러한 변화에 기초해 통합된 기본개념과 중심사상을 제공한다.

- Formal Methods 의 기본사상은 지난 25년 동안 조금은 바뀌었지만, 컴퓨터 프로그래밍에 관한 공식적인 이론적 틀에 있어선 여전히 혁명적인 변화이다.

DEFINITION AND OVERVIEW OF FORMAL METHODS

- 좁은 의미의 정의로는 " Formal Methods 의 지지자들에 의해 추천되는 변화들을 실제로 더 좋게 이끌어 낸다." 라는 것이다.
- Formal Methods 는 두 개의 필수 요소를 가지고 있다.
- 첫째, Formal Methods 는 Formal Language 를 사용한다.
- Formal Language 는 정형화 된 문자열의 한 집합이다.
- 예를 들어 영어와 같이 알파벳 같은 기호로 뚜렷하게 구분되는 문자열을 사용해야 한다는 것이다.

- 둘째, 소프트웨어에서의 Formal Methods 는 추론기능을 지원한다.

- 추론 기능이란 전제에 해당하는 어떤 이론이 진실로 규명된 원칙으로부터 추론되어 나왔다면, 결과로 알려진 또 다른 이론들 역시도 추론되어 질 수 있다는 것이다.

- 즉, 소프트웨어 개발에서의 Formal Methods 는 Formal Language를 이용 해서, 표현된 소스코드, 소프트웨어 구조, 명세 등과 같은 결과물들의 속성에 대한 증명이 가능해야 한다는 것이다.

- 다시 말해서, 실질적으로 증명되었다는 것은 하나의 implementation 이미 명세 된 기능대로 정확히 이행된다는 것이고, Formal Methods 는 이러한 것을 입증 할수 있는 도구를 제공한다.

USE OF FORMAL METHODS

- 소프트웨어 엔지니어링분야에서 Formal Method 는 포괄적인 개념이다.
- Formal Method 는 요구, 설계, 코딩작업등 여러 작업에 직접적으로 적용되며, 시스템 테스트 및 유지에도 중대한 역할을 한다.
- Formal Method 는 많은 프로그래머의 기본적인 도구, 프로그래밍언어의 개발과 표준화에 영향을 끼쳐왔다.
- 또한 Formal Method 는 표준 업무, 특히 설계분야와 디자인 방법론의 변화에 있어서 중요한 역할을 한다.

WHAT CAN BE FORMALLY SPECIFIED

- Formal Methods 는 컴퓨터와 관련된 언어로 표현이 가능한 다양한 방면의 것들의 정밀하고 간결한 명세를 제공 한다.

- 시스템이 무엇을 수행해야 하는지, 소프트웨어 엔지니어링 분야에서 가장 성가신 문제인 여러 객체 사이의 관계를 정의 할 수 있게 한다.

- Formal Methods 분야의 전문가들은 정확한 명세를 기록하기 위해 Formal Methods 를 이용한다.

- 가장 잘 알려진 Formal Methods 중 몇몇은 다음과 같다.

- Z (pronounced "Zed")
- ML
- Vienna Development Method (VDM)
- Larch
- Communicating Sequential Processes(CSP)

- Formal Methods 는 기능적인 것보다 시스템의 전체적인 구조를 명세하는데 사용된다.

- 예를 들어 Formal Methods 는 실질적으로 컴퓨터 프로그램의 Software Safety와 Security Property 를 보증하기 위해 사용된다.

- "Unsafe state 가 발생하지 않거나 security 가 완벽히 보장되있다" 라는 것을 증명하는 일의 이점은 소프트웨어의 Formal Verification 완성의 비용적인 정당성을 입증한다.

- 명세는 반드시 시스템이 뭘 해야 하는지 기술해야 하지 어떻게 하는지를 기술하지 않는다.

Reasoning about a Formal Description

- Formal Methods 는 내포된 것을 이끌어 내거나 Specification 추론에 쓰일 수 있는 다양한 테크닉을 제공한다.
- 모든 입력이 오직 한가지 결과만을 산출해 내는가?
- 의도하지 않은 놀라운 결과가 시스템에 생겨날 수 있는가?
- Formal Methods 는 이러한 질의를 추론할 수 테크닉을 제공한다.
- Formal Methods 는 Formal Proof 와 Implements 가 명세를 만족하는지에 대한 Formal Verification 을 제공한다.

Tools and Methodology

- 소프트웨어 개발의 최종적인 결과물은 혼자서 작동하는 시스템이 아니다.
- 최초의 명세와 "프로그램이 그 명세를 충족시키는가" 라는 증명 둘 다 중요하다.
- 결과적으로 증명과 프로그램은 밀접한 연관을 가지고 평행하게 개발되어야 한다.
- 프로그램은 반드시 정확하게 입증되어져야 하기 때문에 명백하게 안정적인 구조를 통해 제작되어져야 한다.

- 새로운 개념들로 고안된 프로그램에 관해서 Formal Methods 만이 정형적으로 분석 할수 있었다.
- 따라서, Formal Methods 는 언어의 특성을 평가 하기 위한 도구로 넓게 인정받았다.
- 이것이 일찍이 많은 전문가들이 지지했던 구조화 프로그램 등장의 주된 동기이다.

SPECIFICATION METHODS

- Formal Methods 는 주로 검증 방식을 지원하기 위해 개발되어 졌다.
- Formal Methods 를 사용해서 진행되는 프로젝트들은 Formal Methods 를 프로젝트의 속성을 Specification 하는 데에만 사용해 왔다.
- 이것은 몇몇의 Methods 와 언어로도 할 수 있는 일이지만, 언어나 Methods 둘 중 한가지 만으로는 응용프로그램영역, 또 다른 Methods, 그리고 시스템의 전반적인 사항을 완벽하게 명세할 수는 없다.
- 따라서, Formal Specification 에 대한 지식을 가진 유저도 그 둘 중 어느 하나를 채택하기 전에 Methods 들과 언어의 차이의 따른 강점과 약점을 이해할 필요가 있다.

Semantic Domains

- Formal Specification Language 는 잘 정의된 문법적인 규칙과 알파벳으로 구성된다.

- 언어는 몇 개의 모델을 가질 수 있지만 대부분은 다른 모델들보다 좀 더 자연스러운 특정 몇 개만의 모델을 찾게 된다.

- 주어진 명세를 만족하는 언어 객체는 non unique하다. 몇몇의 객체는 특정한 명세가 관련되기 전까지는 동등하다.

- 이런 non-unique 때문에 그 명세는 몇몇 관점에선 구문영역에 있다기보단, 추상화의 좀 더 높은 단계에 있다.

구문영역 이란? 원시 언어의 문장을 올바르게 구성하기 위한 규칙. 자연어의 문법에 해당한다.

- Specification Language 는 다른 Implements 로부터 파생되어진 추상화는 받아들이지만, 본래의 필수적인 속성은 유지한다.
- 같은 구문영역으로부터 정의된 Different Specification Methods 는 이미 명세된 객체를 다르게 명세 하는 것을 허용한다.
- 이러한 개념은 수학적 개념을 이용함으로써 좀더 간결하게 정의 될 수 있다.
- 그런 다음 명세는 일관성과 완전성을 검사 받을 수 있다.
- Specification Language 는 구문 영역에 의거해 다음과 같은 3가지로 분류될 수 있다.

- 1. ADT specification language.
- 2. Process specification language.
- 3. Programming language.
- Vienna의 개발방법인 "Z" 와 Larch는 ADT Specification Language 의 한 예로 algebras 를 설명할 수 있고 ADT는 Implementation의 특성을 정의 하지 않아도 데이터 타입의 특성을 정의 할 수 있다.
- Process specification language는 state sequence, event sequence, streams, partial orders and state machines를 설명한다.

- C.A.R. Hoare's Communicating Sequential Processes(CSP) 는 전통적인 프로세스 specification language다.

- 프로그래밍언어는 다양한 언어 모델로 명백한 예를 제시한다.

- Formal Methods 는 프로그래밍에 있어서 사용가치가 높다. 왜냐하면, Physical Machines 를 위한 명령어의 집합이자, 수학적인 객체의 추상화이자, 위 두 개에 대해 양자택일이 가능한 모델을 제시하기 때문이다.

Model oriented and Property-Oriented Methods

- model oriented 와 property oriented method 는 Formal Methods 를 분류하기 위한 또 다른 방식을 제공한다.
- model oriented methods 는 구조적이거나 운영적인 것을 설명한다.
- model oriented methods 에서 명세는 시스템 모델을 증명함으로써, 직접적으로 시스템을 설명한다. 이 모델의 특성은 시스템이 요구하는 특성을 정의한다.
- 전형적으로 모델은 relations, functions, sets, sequences 같은 abstract mathematical structures를 사용한다.

- property oriented methods 는 선언적인 것, 정의적인 것, 또한 "Specification 을 시스템이 반드시 만족해야 한다" 라는 조건을 설명한다.

- 이런 specification 을 만족하는 모든 시스템은 기능적으로 정확하다. 하지만, specification 은 입력으로부터 시스템의 출력을 어떻게 결정하는지 보여주는 기계적인 모델을 제공하지는 않는다.
- algebriac 과 axiomatic 이라는 두 종류의 property oriented methods 가 존재한다.

- algebriac methods 에서 프로그램을 정의하는 속성들은 모두 동등하다. ADTs 는 종종 algebraic methods에 의해 명세 된다.

- axioms 의 다른 타입은 axiomatic methods에서 사용된다.

USE OF SPECIFICATION METHODS

- 일반적으로 Formal Methods 는 더더욱 정확한 명세를 제공한다.
- 버그 같은 것들을 Life-cycle 에서 일찍이 감춰버릴 수 있다.
- 오류가 더 빨리 발견될 수록, 더 쉽게 그것을 제거할 수 있기 때문에 Formal specification methods 는 생산성과 질 둘 모두를 향상시킬수 있었다.
- 일반적으로 프로그래머들에게 model oriented 방식이 더욱 편안하게 여겨진다. 왜냐하면 그 방식이 프로그래밍에 좀 더 밀접한 방식이기 때문이다.

- 하지만, model oriented 방식의 지나친 명세를 이끄는 경향이 모델방식의 복잡성을 가중시킨다.
- 따라서, Operation 의 관계 인식이 더 어려워 지는 경향이 있다.
- Property Oriented Methods 는 일반적으로 구축하는데에 있어 더 어렵다. 명세하기 위한 원리는 평범하지 않고 일관성과 완전성은 구조화하기 힘들다. 보통 완전성이 일관성보다 더욱 해결하기 어렵다.
- 몇몇 원리가 중복 되 있는지 아닌지 혹은 요구가 더 있는지 없는지는 쉽사리 찾을 수가 없다.

LIFE CYCLE AND TECHNOLOGIES WITH INTERGRATED FORMAL METHODS

- Formal Methods 의 효과적인 비용절감에 대한 모든 이점은 얻기 위해서 Formal Methods 는 소프트웨어 조직의 표준 절차에 통합되어져야 한다.
- 소프트웨어 프로세스에서 Formal Methods 를 통합하는 두 가지 방법이 있다.
- 하나는 자동화 도구를 과도하게 사용하는 것이고 나머지 하나는 반대로 비기계적이고 비자동화 된 것을 이용하는 것이다.

Verification Systems and Other Automated Tools

- Automated Verification System 은 유저에게 소프트웨어 시스템의 정형화된 원리의 존재를 증명하기 위한 도구를 제공한다.

- Robert S.Boyer and J Strother Moore's Nqthm system, Robert L.Constable and Joseph L.Bates' Proof Refinement Logics(PRL), Robin Miler's Logic for Computable Functions(LCF), and the Larch Prover 는 초기 검증시스템의 예가 된다.

- Automated Verification System 은 원리로부터 나오는 각각의 단계에 정형화된 증명을 하지 않는다.