Introduction to Program Synthesis (SS 2025)

Chapter 1 - Introduction

Dr. rer. nat. Roman Kalkreuth

Chair for AI Methodology (**AIM**), Department of Computer Science,
RWTH Aachen University, Germany

Center for
Artificial Intelligence

RWTH AACHEN
UNIVERSITY

Why program synthesis? Some major software bug issues...

1962	Mariner 1 Spacecraft → incorrect guidance signals → $320 million
1994	Pentium FDIV Bug → → lookup table bug → $475 million
1996	ESA Ariane 5 Flight V88 → conversion error → $370 million
2019	Boeing 737 Max → MCAS software bug → fatalities
2024	Intel Raptor Lake → stability issues
2024	CrowdStrike Falcon software update → world-wide outage

Why program synthesis?

- Software is **complex** and **fragile**
- Prone to human error
- Automated search and verification → Holy grail of software development?
- Well, its complicated[1]
  - ⇝ Despite the hype, LLMs lack of genuine formal reasoning capabilities[2][3]
  - ⇝ More issues and shortcomings will be addressed later in the course

[1] https://hackernoon.com/testing-llms-on-solving-leetcode-problems-in-2025

[2] https://garymarcus.substack.com/p/llms-dont-do-formal-reasoning-and

[3] https://arxiv.org/abs/2410.05229

General Idea

- Software development process → can be automated (to some extend)
- Synthesis of **correct** and **efficient** computer programs with respect to predefined specifications
- Universal approach → not limited to a specific programming language, paradigm or level

Definition and Problem Statement

Definition (Program Synthesis)

Automated discovery of executable programs that match predefined forms of constraints such as input-output relations.

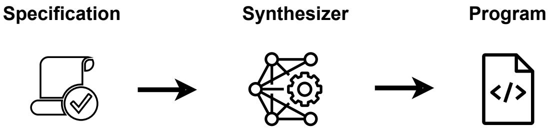
Fundamental Notations

Table: Notation

Symbol	Definition
$\mathcal{P}$	Program
$\mathcal{X}$	Set of inputs
$\mathcal{Y}$	Set of outputs
Φ	Constraints
ψ	Specification

Program Synthesis

- Generation of computer programs from a collection of **artifacts**
  - ⇝ **Automated search** in a space of possible programs
  - ⇝ Matching **semantic** and **syntactic requirements**
- Interaction between **synthesis** and **machine learning**

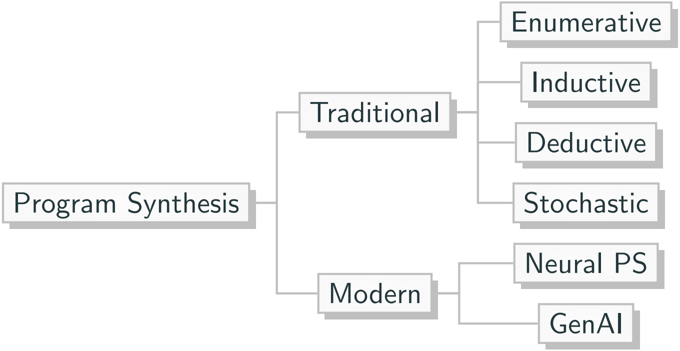


General Definition and Problem Statement

Definition (Program Statement)

Seek a program P that satisfies a specification ψ on a input set $\mathcal{X}$ and is subject to constraints Φ.

Taxonomy (excerpt)



Further Classification

- PS search methods are often performed directly on **high-level symbolic representations** of problems
  - $\rightsquigarrow$ Discovery of human-readable solutions
- Modern PS methodology → based on ML paradigms
- PS can be therefore nowadays considered as **symbolic AI/ML** methodology.

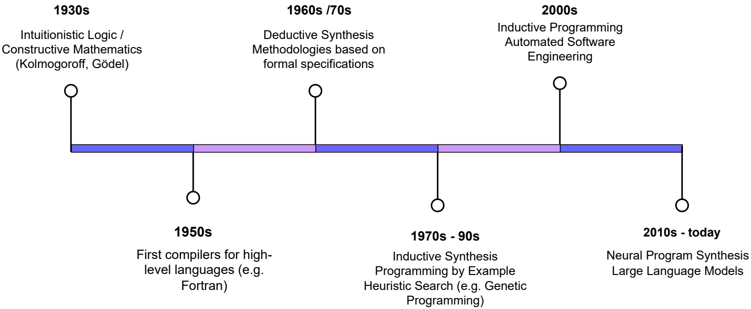
Scope and Distinction

- Synthesis of programs from a collection of given artifacts
  - $\leadsto$ **Artifacts** → represent semantic and syntactic requirements (for the programs)
- Program synthesis by its definition is not (merely):
  - $\leadsto$ **Compilation** → Thus, compilation and synthesis are very closely related as they share the similar goals
  - $\leadsto$ **Machine learning** → Incremental and transformative synthesis process that is nowadays *backed and guided* by ML concepts and techniques
  - $\leadsto$ **Optimization** → Definition of optimization objectives for the synthesis process

Historical Background

1932	Discovery of algorithms with proofs (Kolmogorov, 1932) [Kol32]
1954	FORTRAN Automatic Coding System (Backus et al., 1954) [Bac+57]
1957	Applications of recursive arithmetic to the problem of circuit synthesis (Alonzo Church) [Fri63]
1969	Solving Sequential Conditions by Finite State Strategies (Buchi and Landweber) [BL90]
1971	Toward automatic program synthesis (Mannar et al., 1971) [MW71]
1975	Transformational synthesis (Manna & Waldinger, 1975) [Kol32]
1977	*Synthesis-by-transformations* paradigm (Burstall & Darlington, 1977) [BD77]
1979-80	Automated Deduction (Manna & Waldinger, 1979; Manna & Waldinger, 1980; Bibel, 1980) [MW79; MW80; Bib80]
1980s	Efficient strategies for synthesis (generate & test, divide & conquer, problem reduction) (Smith, 1985b; Smith, 1987b) [Smi83; Smi86]
1985	Adaptive Generation of Simple Sequential Programs (Nichael L. Cramer) [Cra85]
1989	Hierarchical Genetic Algorithms Operating on Populations of Computer Programs (John R. Koza) [Koz89]
1992	Genetic Programming (John R. Koza) [Koz93]

Timeline



1930s

Intuitionistic Logic /
Constructive Mathematics
(Kolmogoroff, Gödel)

1950s

First compilers for high-level languages (e.g. Fortran)

1960s /70s

Deductive Synthesis
Methodologies based on formal specifications

1970s - 90s

Inductive Synthesis
Programming by Example
Heuristic Search (e.g. Genetic Programming)

2000s

Inductive Programming
Automated Software Engineering

2010s - today

Neural Program Synthesis
Large Language Models

Paradigms

- **Proofs-as-programs:**
  - Synthesis of an algorithm → constructive proof of the statement
- **Synthesis by transformations:**
  - Derivation of programs from given specifications by forward propagation
  - Originally based on rewrite rules that encode logical laws

Paradigms

- **Deductive**
  - Deductive reasoning → specific conclusions are drawn from general premises
  - General versions being transformed to a specified version that matches the specification
  - **Top-down approach** → from general information to the specific conclusions
- **Inductive**
  - Inductive reasoning → drawing general conclusions based on specific observations
  - Incremental synthesis based on given examples (i.e. input-output mappings)
  - **Bottom-up approach** → from specific to general
- **Stochastic** → Applying principles of randomized search heuristics
- **Neural** → Use of deep learning based methodologies

Search spaces

- **Symbolic:** Compositions from a finite set of functions $\mathcal{F}$ and set of finite terminals (e.g. variables or constants) $\mathcal{T}$
  - $\mathcal{P} \in \mathcal{F} \times \mathcal{T}$
  - Often represented as non-linear data structures such as trees and graphs
- **Syntactical:** Compositions from nonterminal symbols $\mathcal{N}$ and terminal symbols $\mathcal{T}$ in accordance to production rules $\mathcal{P}$
  - Syntax space is language specific
  - Terminal symbols $\rightarrow$ Functions or operators
  - Nonterminal symbols $\rightarrow$ Variables or constants
- **Semantic:** Discrete or continious output space of candidate programs
  - $\mathcal{P}(\mathcal{X}) \in \mathbb{R}^n$ or $\mathbb{N}^n$

Objectives

- **Feasibility:** Discovery whether a feasible program can be obtained that matches the predefined specification
    - Usually no constraints specified
    - No prior knowledge given → *Synthesis from scratch*
- **Efficiency:** Consideration of optimization objectives to obtain a more efficient solution
    - Minimization of runtime and/or complexity
- **Reliabitlity:** Construction of robust programs that guarantee the predefined specification

Challenges

- Search in symbolic and syntax space is often **ill-conditioned**:
  - Large search spaces
  - Roughness of the cost function landscape
  - No gradient descent → gradient-free methods needed
  - Poor local search features
- **Fragility** of syntactical compositions
- Programming requires **formal reasoning**

References I

[Kol32] A. Kolmogoroff. “Zur Deutung der intuitionistischen Logik”. In: *Mathematische Zeitschrift* 35.1 (1932), pp. 58–65. DOI: 10.1007/BF01186549. URL: https://doi.org/10.1007/BF01186549.

[Bac+57] J. W. Backus et al. “The FORTRAN automatic coding system”. In: *Papers Presented at the February 26-28, 1957, Western Joint Computer Conference: Techniques for Reliability*. IRE-AIEE-ACM ’57 (Western). Los Angeles, California: Association for Computing Machinery, 1957, pp. 188–198. ISBN: 9781450378611. DOI: 10.1145/1455567.1455599. URL: https://doi.org/10.1145/1455567.1455599.

[Fri63] Joyce Friedman. “Alonzo Church. Application of recursive arithmetic to the problem of circuit synthesisSummaries of talks presented at the Summer Institute for Symbolic Logic Cornell University, 1957, 2nd edn., Communications Research Division, Institute for Defense Analyses, Princeton, N. J., 1960, pp. 3–50. 3a-45a.”. In: *Journal of Symbolic Logic* 28.4 (1963), pp. 289–290. DOI: 10.2307/2271310.

References II

[BL90] J. Richard Buchi and Lawrence H. Landweber. "Solving Sequential Conditions by Finite-State Strategies". In: *The Collected Works of J. Richard Büchi*. Ed. by Saunders Mac Lane and Dirk Siefkes. New York, NY: Springer New York, 1990, pp. 525–541. ISBN: 978-1-4613-8928-6. DOI: 10.1007/978-1-4613-8928-6_29. URL: https://doi.org/10.1007/978-1-4613-8928-6_29.

[MW71] Zohar Manna and Richard J. Waldinger. "Toward Automatic Program Synthesis". In: *Commun. ACM* 14.3 (1971), pp. 151–165. DOI: 10.1145/362566.362568. URL: https://doi.org/10.1145/362566.362568.

[BD77] R. M. Burstall and John Darlington. "A Transformation System for Developing Recursive Programs". In: *J. ACM* 24.1 (Jan. 1977), pp. 44–67. ISSN: 0004-5411. DOI: 10.1145/321992.321996. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.19.4684&rep=rep1&type=pdf.

References III

[MW79] Z. Manna and R. Waldinger. "Synthesis: Dreams → Programs". In: *IEEE Transactions on Software Engineering* SE-5.4 (1979), pp. 294–328. DOI: 10.1109/TSE.1979.234198.

[MW80] Zohar Manna and Richard Waldinger. "A Deductive Approach to Program Synthesis". In: *ACM Trans. Program. Lang. Syst.* 2.1 (Jan. 1980), pp. 90–121. ISSN: 0164-0925. DOI: 10.1145/357084.357090. URL: https://doi.org/10.1145/357084.357090.

[Bib80] Wolfgang Bibel. "Syntax-directed, semantics-supported program synthesis". In: *Artificial Intelligence* 14.3 (1980), pp. 243–261. ISSN: 0004-3702. DOI: https://doi.org/10.1016/0004-3702(80)90050-8. URL: https://www.sciencedirect.com/science/article/pii/0004370280900508.

References IV

[Smi83] Douglas R. Smith. "A Problem Reduction Approach to Program Synthesis". In: *Proceedings of the 8th International Joint Conference on Artificial Intelligence. Karlsruhe, FRG, August 1983*. Ed. by Alan Bundy. William Kaufmann, 1983, pp. 32–36. URL: http://ijcai.org/Proceedings/83-1/Papers/005.pdf.

[Smi86] Douglas R. Smith. "Top-Down Synthesis of Divide-and-Conquer Algorithms". In: *Readings in Artificial Intelligence and Software Engineering*. Ed. by Charles Rich and Richard C. Waters. Morgan Kaufmann, 1986, pp. 35–61. ISBN: 978-0-934613-12-5. DOI: https://doi.org/10.1016/B978-0-934613-12-5.50007-0. URL: https://www.sciencedirect.com/science/article/pii/B9780934613125500070.

[Cra85] Nichael Lynn Cramer. "A Representation for the Adaptive Generation of Simple Sequential Programs". In: *Proceedings of the 1st International Conference on Genetic Algorithms*. USA: L. Erlbaum Associates Inc., 1985, pp. 183–187. ISBN: 0805804269.

References V

[Koz89] John R. Koza. “Hierarchical Genetic Algorithms Operating on Populations of Computer Programs”. In: *Proceedings of the 11th International Joint Conference on Artificial Intelligence. Detroit, MI, USA, August 1989*. Ed. by N. S. Sridharan. Morgan Kaufmann, 1989, pp. 768–774. URL: http://ijcai.org/Proceedings/89-1/Papers/123.pdf.

[Koz93] John R. Koza. *Genetic programming - on the programming of computers by means of natural selection*. Complex adaptive systems. MIT Press, 1993. ISBN: 978-0-262-11170-6.