Prof. Dr. Jasmin Blanchette
Dr. Martin Desharnais-Schäfer
Dr. Michael Kirsten
Elisabeth Lempa

Ludwig-Maximilians-Universität München
Institut für Informatik
Discussion on 10.12.2025
Homework due on 17.12.2025 at 16:00

Possible solution for Exercise Sheet 9 in

Scientific and Technical English for Computer Scientists

The exercise sheets consist of in-class exercises and homework. The in-class exercises take place during the second half of the lecture time slots. The homework, which is optional and ungraded, can be submitted via the "Homework" section in Moodle. The homework is subject to peer review.

Unless indicated otherwise, generative artificial intelligence assistants such as ChatGPT may be used, as long as you acknowledge how you use them as specified by the Institute's policy on plagiarism.[1] However, you may not use such tools to generate peer reviews for you. In addition, we strongly recommend that you do not use them to generate entire solutions, since that would defeat the purpose of the exercises.

In-class exercise 9-1 ***Citations*** Check and, if necessary, correct the following passages to use accurate and stylistically correct citations and quotations.

a) Friedrich Gödel pointed out that for every finite subsystem to be satisfiable, it is necessary and sufficient that it is a countably infinite system of formulas. [1]

References

[1] Kurt Gödel. "The completeness of the axioms of the logical function calculus." In: *Monatshefte für Mathematik und Physik* 37.1 (1730), pp. 349–360. DOI: 10.1007/BF01696781.

POSSIBLE SOLUTION:

Kurt Gödel pointed out that for a countably infinite system of formulas to be satisfiable, it is necessary and sufficient that every finite subsystem is satisfiable [1].

[1] https://www.medien.ifi.lmu.de/lehre/Plagiate-IfI.pdf

References

[1] Kurt Gödel. "Die Vollständigkeit der Axiome des logischen Funktionenkalküls." In: *Monatshefte für Mathematik und Physik* 37.1 (1930), pp. 349–360. DOI: 10.1007/BF01696781.

b) In computability theory, Rice's theorem states that all nontrivial semantic properties of programs are undecidable [1].

References

[1] Wikipedia. *Rice's theorem*. 2025. URL: https://en.wikipedia.org/wiki/Rice%27s_theorem.

POSSIBLE SOLUTION:

In computability theory, it follows from Rice's theorem [1] that all nontrivial semantic properties of programs are undecidable.[a]

References

[1] Henry Gordon Rice. "Classes of recursively enumerable sets and their decision problems." In: *Transactions of the American Mathematical Society* 74.2 (1953), pp. 358–366. DOI: 10.2307/1990888.

[a] In the original source, the theorem does not directly mention the property of "decidability," but "complete recursiveness," which Rice intuitively equates in the introduction.

c) In the seminal work "On computable numbers, with an application to the entscheidungsproblem," Alan Turing defined a fundamental machine model that he called after himself as *Turing machine* [2]. He also gave an influential definition of artificial intelligence within his paper "computing machinery and intelligence" by defining a so-called *animation game* [1].

References

[1] Alan Mathison Turing. "computing machinery and intelligence." In: *Mind* LIX.236 (1900), pp. 460–433. DOI: 10.1093/mind/LIX.236.433.

[2] Alan Mathison Turing. "On computable numbers, with an application to the entscheidungsproblem." In: *Proceedings of the London Mathematical Society* s2-42.1 (1800), pp. 230–265. DOI: 10.1112/plms/s2-42.1.230.

POSSIBLE SOLUTION:

In the seminal work "On Computable Numbers, with an Application to the Entscheidungsproblem," Alan Turing defined a fundamental machine model that was later called a *Turing machine* [2]. He also gave an influential definition of artificial intelligence, before this term gained currency, within his paper "Computing Machinery and Intelligence" by defining the so-called *imitation game* [1].

References

[1] Alan Mathison Turing. "Computing machinery and intelligence." In: *Mind* LIX.236 (1950), pp. 433–460. DOI: 10.1093/mind/LIX.236.433.

[2] Alan Mathison Turing. "On computable numbers, with an application to the Entscheidungsproblem." In: *Proceedings of the London Mathematical Society* s2-42.1 (1937), pp. 230–265. DOI: 10.1112/plms/s2-42.1.230.

In-class exercise 9-2 ***Over and Under*** The following passages both overcite and undercite. Remove all superfluous citations, and mark any parts that lack citations but should have them. (You are not asked to find the sources to cite.)

a) Abstract argumentation frameworks (AAFs) [3] are formal systems that represent conflicting pieces of information as a set [1] of arguments, and a binary attack relation. In recent years, AAFs have been widely used for a variety of different tasks in knowledge representation and artificial intelligence systems [4]. Several extensions that have been proposed for AAFs have been presented, among them Bipolar Argumentation Frameworks [2], which include a binary support relation in addition to the attack relation.

References

[1] Georg Cantor. "On a property of the class of all real algebraic numbers." In: *Crelle's Journal for Mathematics* 77.1874 (1874), pp. 258–262.

[2] Claudette Cayrol and Marie-Christine Lagasquie-Schiex. "On the acceptability of arguments in bipolar argumentation frameworks." In: *ECSQARU 2005*. Springer. 2005, pp. 378–389.

[3] Phan Minh Dung. "On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games." In: *Artificial intelligence* 77.2 (1995), pp. 321–357.

[4] Nikos Karacapilidis and Dimitris Papadias. “Computer supported argumentation and collaborative decision making: the HERMES system.” In: *Information systems* 26.4 (2001), pp. 259–277.

POSSIBLE SOLUTION:

Abstract argumentation frameworks (AAFs) [3] are formal systems that represent conflicting pieces of information as a set of arguments, and a binary attack relation. In recent years, AAFs have been widely used for a variety of different tasks in knowledge representation and artificial intelligence systems [4] [more citations needed—such a claim should have multiple sources, and ideally, more current ones]. Several extensions that have been proposed for AAFs have been presented, among them Bipolar Argumentation Frameworks [2], which include a binary support relation in addition to the attack relation.

b) A refinement type is a kind of dependent type that “refines” an existing type by use of a predicate that needs to hold for every value to be admitted to the refined type. The concept was first introduced by Freeman and Pfennig in 1991 [1], who presented a refinement type system for a subset of Standard ML [2]. Since then, refinement type systems have been developed for languages such as Haskell [5], TypeScript [6], Rust [3, 4], and Scala.

References

[1] Tim Freeman and Frank Pfenning. “Refinement types for ML.” In: *PLDI ’91*. 1991, pp. 268–277.

[2] Robert Harper, David MacQueen, and Robin Milner. *Standard ML*. Department of Computer Science, University of Edinburgh, 1986.

[3] Nico Lehmann, Adam T. Geller, Niki Vazou, and Ranjit Jhala. “Flux: Liquid types for Rust.” In: *Proceedings of the ACM on Programming Languages* 7.PLDI (2023), pp. 1533–1557.

[4] Hiromi Ogawa, Taro Sekiyama, and Hiroshi Unno. “Thrust: A prophecy-based refinement type system for Rust.” In: *Proceedings of the ACM on Programming Languages* 9.PLDI (2025), pp. 2056–2080.

[5] Niki Vazou, Eric L. Seidel, Ranjit Jhala, Dimitrios Vytiniotis, and Simon Peyton-Jones. “Refinement types for Haskell.” In: *ICFP ’14*. 2014, pp. 269–282.

[6] Panagiotis Vekris, Benjamin Cosman, and Ranjit Jhala. “Refinement types for TypeScript.” In: *PLDI ’16*. 2016, pp. 310–325.

POSSIBLE SOLUTION:

A refinement type is a kind of dependent type that "refines" an existing type by use of a predicate that needs to hold for every value to be admitted to the refined type. The concept was first introduced by Freeman and Pfennig in 1991 [1], who presented a refinement type system for a subset of Standard ML. Since then, refinement type systems have been developed for languages such as Haskell [5], TypeScript [6], Rust [3, 4], and Scala [citation needed].

In-class exercise 9-3 ***Cite and Quote*** Write a paragraph of at most 200 words explaining generics in the Java programming language. Include at least one citation, one naming of authors, one reference, and one quotation. Potential reference material includes the following:

- *The Java® Language Specification*[2]
- *The Java™ Tutorials: Generics*[3]
- *Java® Platform, Standard Edition & Java Development Kit*[4]

POSSIBLE SOLUTION:

In the Java programming language, *generics* refers to the ability to define type-parameterized classes, interfaces, methods, and constructors [2, Sections 8.1.2, 9.1.2, 8.4.4, and 8.8.4]. This feature avoids code duplication by making it possible to develop code that is independent of any specific type. Consider the interface `List<T>`, which specifies an "ordered collection (also known as a sequence) [where] the user can access elements by their integer index" [3]. The type variable `T` between angle brackets is a placeholder for the type of the elements stored in the collection. In `List<Integer>` and `List<String>`, the placeholder is replaced by concrete class types, and the resulting types specify collections of integers and strings, respectively. The usual compile-time checks ensure that only objects of the specified type can be stored in each collection. Without generics, this type safety can be achieved only by duplicating the code for each concrete type. For a more in-depth introduction, we refer to Bracha's tutorial [1].

[2] `https://docs.oracle.com/javase/specs/jls/se21/html/index.html`
[3] `https://docs.oracle.com/javase/tutorial/extra/generics/index.html`
[4] `https://docs.oracle.com/en/java/javase/21/docs/api/index.html`

References

[1] Gilad Bracha. *The Java™ Tutorials. Generics*. 2024-10-25. URL: https://docs.oracle.com/javase/tutorial/extra/generics/index.html.

[2] James Gosling, Bill Joy, Guy Steele, Gilad Bracha, Alex Buckley, Daniel Smith, and Gavin Bierman. *The Java® Language Specification. Java SE 21 Edition*. 2023-08-23. URL: https://docs.oracle.com/javase/specs/jls/se21/html/index.html.

[3] *Java® Platform, Standard Edition & Java Development Kit. Version 21 API Specification*. URL: https://docs.oracle.com/en/java/javase/21/docs/api/index.html.