Towards StarExec in the Cloud

David Fuenmayor[1], Jack McKeown[2], and Geoff Sutcliffe[2]

[1] University of Bamberg, Bamberg, Germany
david.fuenmayor@uni-bamberg.de
[2] University of Miami, Miami, USA
jam771@miami.edu,geoff@cs.miami.edu

Abstract

StarExec has been central to much progress in logic solvers over the last 10 years. It was recently announced that StarExec Iowa will be decommissioned, and while StarExec Miami will continue to operate while funding is available, it will not be able to support the large number of logic solver communities supported by the larger StarExec Iowa. In the long term StarExec will necessarily have to migrate to a commonly available compute service. David: Modify as to convey the general idea: re-engineering StarExec as a cloud-native application ... using containers technology (podman, k8s, etc.) and infrastructure as code (IaC) practices ... see comments in introduction This paper describes work being done to build StarExec and ATP systems images, so that they can be run in containers on a broad range of computer platforms. Additionally, this work aims to build a Kubernetes backend in StarExec so that Kubernetes can be used to orchestrate distribute of StarExec job pairs over whatever compute nodes are available. One possibility is to host StarExec-Kubernetes in AWS.

1 Introduction

Automated Theorem Proving (ATP) is concerned with the development and use of tools that automate sound reasoning: the derivation of conclusions that follow inevitably from facts. Automated Theorem Proving (ATP) is at the heart of many computational tasks, in particular for verification [13, 11] and security [9].[1] New and emerging application areas include chemistry [42], biology [7], medicine [15], elections [22, 5], auctions [6], privacy [19], law [25], ethics [10], religion [23, 3, 17], and business [12]. ATP systems are also used as components of more complex Artificial Intelligence (AI) systems, and the impact of ATP is thus extended into many facets of society.

The Thousands of Problems for Theorem Provers (TPTP) World [38] is a well established infrastructure that supports research, development, and deployment of ATP systems. The TPTP World includes the TPTP problem library [37], the TSTP solution library [35], standards for writing ATP problems and reporting ATP solutions [39, 34], tools and services for processing ATP problems and solutions [35], and it supports the the annual CADE ATP System Competition (CASC) [36]. Since its first release in 1993 the ATP community has used the TPTP World as an appropriate and convenient infrastructure for ATP system development, evaluation, and application. The TPTP World has a diverse, engaged, and sustained user community, with various parts of the TPTP World being deployed in a range of applications in both academia and industry.[2] The web page www.tptp.org provides access to all components.

The TPTP problem library was motivated by the need to provide support for meaningful ATP system evaluation. This need was also (or became) evident in other logic solver communities, e.g., SAT [16] and SMT [1]. For many years testing of logic solvers was done on individual

[1] In AWS - aws.amazon.com/what-is/automated-reasoning/, aws.amazon.com/security/provable-security/.
[2] TPTP has contributed to recognized research in 627 publications that cite [37], according to Google Scholar.

developer's computers. In 2010 a proposal for centralised hardware and software support was developed, and in 2011 a $2.11 million NSF grant[3] was obtained. This grant led to the development and availability of StarExec Iowa [33] in 2012, and a subsequent $1.00 million grant[4] in 2017 expanded StarExec to Miami. StarExec has been central to much progress in logic solvers over the last 10 years, supporting 16 logic solver communities, used for running many annual competitions [2], and supporting many many users. StarExec Iowa provides community infrastructure for many logic solver communities, e.g., ASP, QBF, SAT, SMT, Termination, etc, while StarExec Miami is used by the TPTP community. StarExec Miami has features that take advantage of TPTP standards, and is also used to host CASC.

It was recently announced that StarExec Iowa will be decommissioned. The maintainer of StarExec Iowa explained that "the plan is to operate StarExec as usual for competitions Summer 2024 and Summer 2025, and then put the system into a read-only mode for one year (Summer 2025 to Summer 2026)". The 2017 grant for StarExec Miami paid for the hardware and three years of system administration. The hardware is still hosted by the University of Miami High Performance Computing group, funded on a shoe-string budget by the TPTP World. While StarExec Miami will continue to operate while funding is available, it will not be able to support the large number of logic solver communities supported by the larger StarExec Iowa. In the long term StarExec will necessarily have to migrate to a commonly available compute service. David: Modify as to convey the general idea: stepwise migration/redesign of StarExec as a cloud-native application so that users can deploy their own (possibly customized) StarExec to their own cloud environments or on-premises servers, by drawing upon open-source standards (podman, kubernetes, etc.) and infrastructure as code (IaC) practices. This paper describes work being done to build StarExec and ATP systems images, so that they can be run in containers on a broad range of computer platforms. David: I would rewrite the previous as just: "containerizing StarExec and ATP systems" possibly adding a footnote that explains the notion of "containerization") Additionally, this work aims to build a Kubernetes backend in StarExec so that Kubernetes can be used to orchestrate distribute of StarExec job pairs over whatever compute nodes are available. One possibility is to host StarExec-Kubernetes in AWS. David: We could say something like: supported by an AWS AR grant, we will implement the proposed approach by deploying a new (re-engineered) version of StarExec in AWS. This StarExec instance will not only be fully functional to the community (e.g. run SystemOnTPTP, CASC competitions, etc. as much as our budget allows), but shall also serve as a an exemplary implementation (IaC template) for those willing to deploy their own (possibly customized) StarExec in their own infrastructure (cloud or on-premises).

David: We shall also note that podman and kubernetes can/will both serve as backends for StarExec (the podman team keeps adding basic orchestration functionalities, so it can work as a lightweight alternative to k8s in simple scenarios). Btw podman is designed to play well with k8s (e.g. it understands k8s yaml config) so there is no big overhead in supporting both I think (this might facilitate adoption, specially since k8s still keeps some of its reputation as being very complex).

This paper is organized as follows: Section 2 provides a short background to ATP systems, StarExec, and containerization. Section 3 describes how StarExec has been containerized, and Section 4 describes how ATP systems have been containerized. Section 5 explains how we plan to deploy the containerized StarExec and ATP systems in a Kubernetes setting. Section 6 concludes and look forward to future work.

[3] NSF Awards 1058748 and 1058925, led by Aaron Stump and Cesare Tinelli at the University of Iowa
[4] NSF Award 1730419

2 Background

2.1 ATP Systems

ATP Systems are complex pieces of software, typically using advanced data structures [28], sophisticated algorithms [41], and tricky code optimizations[27]. They are written in a variety of programming languages: Prolog [24, 14], Scala [32], C [29], C++ [26], OCaml [18], Python [30], etc. Their build processes include techniques such as parser generators [31], Makefiles, code repositories, specific versions of libraries, etc. For a user who is focussed on an application of ATP installing an ATP system can be a deal breaker, and many early users selected a weaker system, e.g., Otter [21], for their experiments because it was readily available and easy enough to install. There have been some proposals for standardising the ATP system build process, e.g., `tptp.org/Proposals/SystemBuild.html`, but the diversity of ATP system software makes conformity nigh impossible. An alternative is to push the task back on the system developers, and one approach to this is packaging the systems into images, as discussed in Section 2.3.

2.2 StarExec

The need to provide support for meaningful system evaluation has been recognized in many logic solver communities, e.g., TPTP [40], SAT [16], SMT [8], Termination [20], etc. For many years testing of logic solvers was done on individual developer's computers. In 2010 a proposal for centralised hardware and software support was developed, and in 2011 a $2.11 million NSF grant[5] was obtained. This grant led to the development and availability of StarExec Iowa [33] in 2012, and a subsequent $1.00 million grant[6] in 2017 expanded StarExec to Miami. StarExec has been central to much progress in logic solvers over the last 10 years, supporting 16 logic solver communities, used for running many annual competitions [2], and supporting many many users.

Figure 1 shows the architecture of the currently deployed StarExec. The hardware consists of a single head node and multiple compute nodes. The head node provides the browser interface for users, in particular it accepts job requests that generate job pairs consisting of an ATP system and a problem file, and distributes the pairs to the compute nodes using the SUN Grid Engine (SGE). (For development and testing, the head node can also run job pairs itself using a local backend.) The head node maintains a relational MariaDB database, and all the nodes access an NFS mounted shared file system. The database records everything, including the ATP systems' files and the problem files in the file system. Job pairs executing on a compute node have their time and memory usage limited and reported by the `runsolver` [] utility (the `BenchExec` [4] utility in StarExec Iowa). The results and resource usage data from completed job pairs are stored in the file system, and recorded in the database. The browser interface provides the necessary facilities to upload ATP systems, upload problem files, browse the ATP systems and problems, create jobs, track job progress, browse and download job results, and delete ATP systems, problems, jobs, etc.

It was recently announced that StarExec Iowa will be decommissioned. The maintainer of StarExec Iowa explained that "the plan is to operate StarExec as usual for competitions Summer 2024 and Summer 2025, and then put the system into a read-only mode for one year (Summer 2025 to Summer 2026)". While StarExec Miami will continue to operate while funding is available, but it will not be able to support the large number of logic solver communities that use the larger StarExec Iowa cluster. In the long run it will be necessary for StarExec users to

[5] NSF Awards 1058748 and 1058925, led by Aaron Stump and Cesare Tinelli at the University of Iowa
[6] NSF Award 1730419

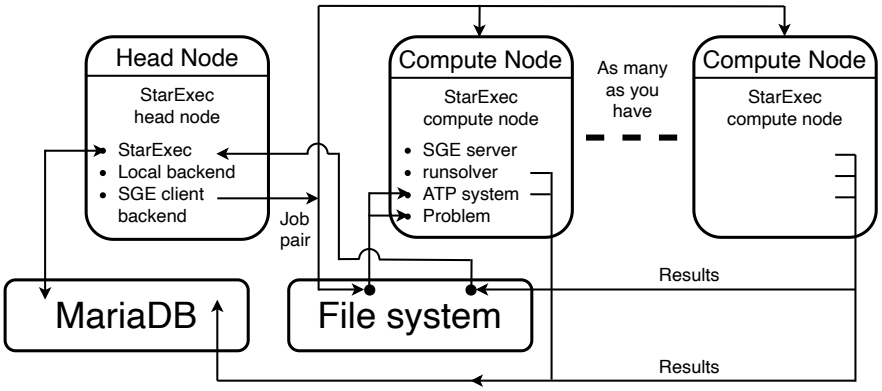

Figure 1: StarExec Architecture

transition to new environments, and several plans are (at the time of writing) being discussed. One effort is that described in the paper.

2.3 Containerization

This section was written with the help of ChatGPT 3.5.

Containerization is a technology that allows developers to package an application and its dependencies into a standardized unit called a container. These containers encapsulate the application code, runtime, libraries, and other necessary components, providing a consistent and isolated environment for running the application across different computing environments. One of the key benefits of containerization is its ability to abstract away the underlying infrastructure. Containers are designed to be lightweight and portable, making it easy to deploy applications across various platforms, such as laptops, servers, virtual machines, and cloud environments. This portability ensures that applications behave consistently regardless of the underlying infrastructure, simplifying the development and deployment process. Furthermore, containerization offers several advantages in terms of scalability, resource efficiency, and security. Containers share the host operating system's kernel, which reduces overhead compared to traditional virtualization techniques. As a result, containers can be started and stopped quickly, allowing for rapid scaling of applications to meet changing demand. Containers provide a level of isolation that helps prevent conflicts between applications and enhances security by limiting the impact of potential vulnerabilities.

Popular containerization platforms like Docker, Podman, and Kubernetes have played a significant role in popularizing container technology. These platforms provide tools for building, distributing, orchestrating, and managing containers at scale. Docker, for example, introduced a user-friendly interface and a standardized format for defining container images, making it easier for developers to adopt containerization in their workflow. Kubernetes, on the other hand, focuses on container orchestration and automating the deployment, scaling, and management of containerized applications in production environments.

3 Containerizing StarExec

A containerized StarExec can be used locally in Docker/Podman/Kubernetes. This allows StarExec users to build and test StarExec installation packages on their own computers before deploying to the StarExec Miami installation.

All the files for building the StarExec image is available from ...

github.com/StarExecMiami/starexec-containerized

4 Containerizing ATP Systems

The ATP systems' images are built in a hierarchy, show in Figure 2. The underlying operating system is `ubuntu:latest` from `dockerhub` ...

hub.docker.com/_/ubuntu

The first image built for this work is `ubuntu-build`, which adds to `ubuntu:latest` using `apt-get` to add common software such as `cmake`, `git`, `tcsh`, `python3`, and `wget`. `ubuntu-build` also creates an `artifacts` directory where the components required for an ATP system execution are placed.

The `tptp-world-build` image provides utilities from the TPTP World [38] that are used by ATP systems, e.g., `SPCForProblem` detects the Specialist Problem Class (SPC) [40] of a problem that is used by some ATP systems to decide on what search parameters to use. Additionally, the `runsolver` utility for limiting and reporting the resources used by an ATP system is part of this image. To support these utilities some libraries that are not part of the `ubuntu-build` have to be added before downloading and building the image. The `/benchmark` directory, where the TPTP problem for the ATP system to solve is placed, is created as part of this image.

On top of the `ubuntu-build` image are the individual ATP system's *ATP-system-name*`-build` images, and on top of that with the `tptp-world-build` image are the final the *ATP-system-name*`-runsolver` images. The details of building the *ATP-system-name*`-*` images are provided below.

The *ATP-system-name*`-runsolver` images are pushed to `dockerhub` in ...

hub.docker.com/repositories/tptpstarexec

which has a directory for each ATP system. The pushed images are tagged as *ATP-system-name*`:`*ATP-system-version*`-runsolver-`*architecture*, where *architecture* is, e.g., `arm64` or `amd64`. All the files for building the ATP system images is available from ...

github.com/StarExecMiami/starexec-kubernetes/tree/main/images

A `Makefile` to build images for E, Leo-III, and Vampire is included. David: Image file below not found

Figure 2: ATP System Image Hierarchy

4.1 Building ATP System Images

An *ATP-system-name*`-build` image adds the ATP system's executables to the `ubuntu-build` image. The ATP system is retrieved online, e.g., from a GitHub repository, and the necessary commands to build the executables are run. The executables are copied into the `/artifacts`

directory. The choice of which version of the ATP system to containerize is made inside the `Dockerfile`, e.g., in Figure 3 `E 3.0.03` is chosen. This localization is necessary because the incantations for selecting and retrieving an particular ATP system version vary from system to system. By convention the image is named *ATP-system-name*`-build`, and by default has the `:latest` tag, Figure 3 shows the `Dockerfile` for building E's `-build` image, using the command `podman build -t eprover-build .`.

```
#------------------------------------------------------------
FROM ubuntu-build

# Clones repository
ARG E_VERSION=E-3.0.03
RUN git clone --depth 1 --branch $E_VERSION https://github.com/eprover/eprover.git

# Set working directory to cloned sources directory
WORKDIR /eprover

# Builds first-order executable
RUN ./configure --bindir=/artifacts && \
    make && \
    make install
# RUN cp PROVER/eprover /artifacts/eprover

# Builds higher-order executable
RUN ./configure --enable-ho && \
    make rebuild
RUN cp PROVER/eprover-ho /artifacts/eprover-ho
#------------------------------------------------------------
```

Figure 3: The `Dockerfile` for E's `-build` image

An *ATP-system-name*`-runsolver` image adds `runsolver` control to the *ATP-system-name*`-build` image, so that when executed the resource usage of the ATP system can be limited and reported. The image is based on the *ATP-system-name*`-build` and `tptp-world-build` images. Note how the image is built with the default `:latest` tagged *ATP-system-name*`-build`. The executables from the *ATP-system-name*`-runsolver` image are copied from its `/artifacts` directory into this image's `/artifacts` directory. Additionally, the `run_system` script, described in Section 4.2, is copied into `/artifacts`. By convention this image is named *ATP-system-name*`:`*ATP-system-version*`-runsolver`, i.e., including the version number so that users know what version of the ATP system has been containerized. Figure 4 shows the `Dockerfile` for building E's `-runsolver` image, using the command `podman build -t eprover:3.0.03-runsolver .`, i.e., it contains E version `3.0.03`.

4.2 Running ATP System Containers

A Python script `run_image.py` is available to run an ATP system image from the command line. The script is shown in Appendix A. Minimally the script must have the *ATP-system-name*`-runsolver` image name as a command line argument. By default `run_image.py` runs the *ATP-system-name*`-runsolver` image in a `podman` container, taking the problem from `stdin`, imposing CPU and wall clock time limits of 60s, imposing no memory limit, with the intention

```
#-------------------------------------------------------------
FROM eprover-build AS builder
FROM tptp-world-build

ENV PATH=".:${PATH}"
WORKDIR /artifacts

# E-specific stuff from ostensibly external image
COPY --from=builder /artifacts/eprover /artifacts/
COPY --from=builder /artifacts/eprover-ho /artifacts/

# run_image script
ADD run_image /artifacts/

# run_E script
ADD run_E /artifacts/

ENTRYPOINT ["runsolver"]
#-------------------------------------------------------------
```

Figure 4: The `Dockerfile` for E's `-runsolver` image

that the ATP system should try prove that the problem's conjecture is a theorem. The problem file is passed into the running container using Podman volume mounting, copying the problem to the `/artifacts/CWD/benchmark` file inside the container. All the parameters can be changed via command line options to `run_image.py`.

The entrypoint in the *ATP-system-name*-`runsolver` image is the `runsolver` utility, which in turn starts the `run_system` script with the problem, CPU limit, wall clock limit, memory limit, and the proof request as arguments. Each *ATP-system-name*-`runsolver` image's `run_system` script is responsible for starting the ATP system – this action varies tremendously between ATP systems, and is thus usually provided by the system developer. For example, E has its own script `run_E` that invokes the `eprover` or `eprover-ho` binary depending on whether the problem is first-order or higher-order, and depending on the intention passed in appropriate command line arguments are given to the selected binary along with the problem file and time limit.

5 StarExec with Kubernetes

StarExec (see Section 2.2) is based around a head node that coordinates activities, in particular the creation of jobs as sets of job pairs, with each pair consisting of an ATP system and a problem. MariaDB is used to store job information and results, and NFS is used to share disk space between the head node and compute nodes. StarExec currently offers two backends for running job pairs: the local backend runs pairs on the same computer as the head node, and the Sun Grid Engine (SGE) backend sends pairs out to compute nodes. This structure is inflexible, and must be configured to the specific hardware available. The plan is to use a containerized StarExec (see Section 3), replace SGE with Kubernetes, remove any MariaDB-specific bindings so that other database products can be used, and use containerized ATP systems (see Section 4). Figure 5 shows the generic architecture.

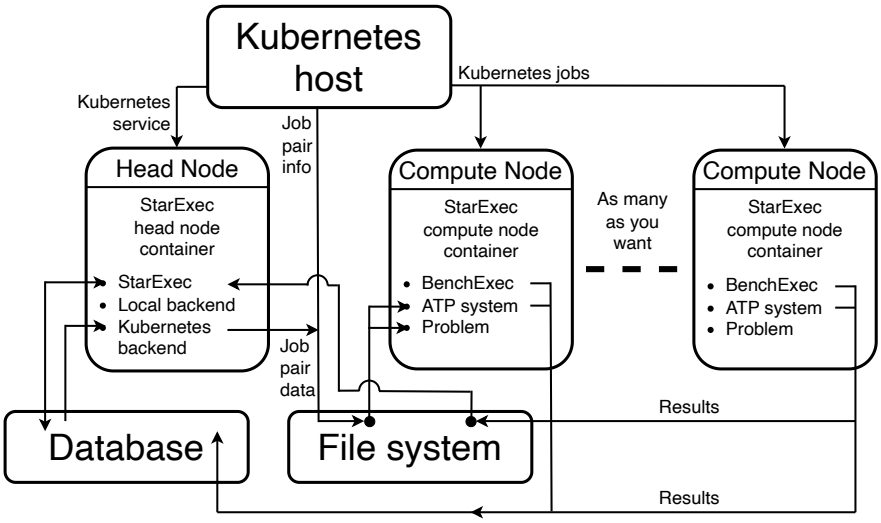

Figure 5: Architecture in Kubernetes

An Amazon Research Award[7] has been granted to implement StarExec with Kubernetes in AWS. This instantiates the the generic implementation as follows:

- The Kubernetes host will be Amazon Elastic Kubernetes Service (EKS)
- The head and compute nodes will be Amazon EC2 nodes.
- The database will be Amazon Relational Database (RDS).
- The file system will be Amazon Elastic File System (EFS).
- The ATP systems' containerization will be made compatible with (possibly be exactly) the Amazon Trusted Solver (ATS) format.

The entire migration of StarExec into AWS will be done using the "infrastructure-as-code" paradigm, using Amazon CloudFormation. The implementation will be tested by copying the TPTP community from StarExec Miami onto the new StarExec AWS. Figure 6 shows the AWS-specific architecture.

6 Conclusion

This paper has described work being done to build StarExec and ATP systems images, so that they can be run in containers on a broad range of computer platforms. Additionally, this work explains plans to build a Kubernetes backend in StarExec so that Kubernetes can be used to orchestrate distribute of StarExec job pairs over whatever compute nodes are available.

This is ongoing work – some of the work is still "in progress", particularly embedding StarExec in Kubernetes on AWS. Hopefully the future will include StarExec being flexibly available in online compute clusters.

[7] Amazon Research Award, Fall 2023. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors, and do not reflect the views of Amazon.

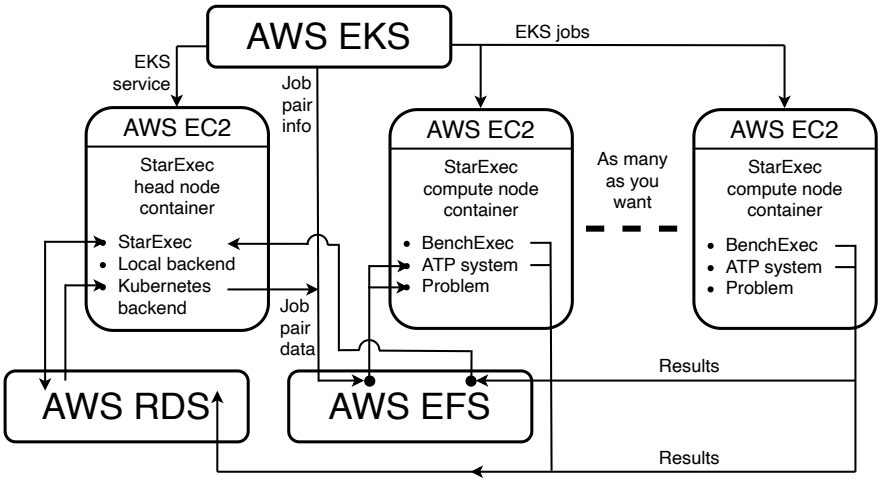

Figure 6: Architecture in EKS on AWS

References

[1] C. Barrett, A. Stump, and C. Tinelli. The SMT-LIB Standard: Version 2.0. In A. Gupta and D. Kroening, editors, *Proceedings of the 8th International Workshop on Satisfiability Modulo Theories*, 2010.

[2] E. Bartocci, D. Beyer, P.E. Black, G. Fedyukovich, H. Garavel, A. Hartmanns, M. Huisman, F. Kordon, J. Nagele, M. Sighireanu, B. Steffen, M. Suda, G. Sutcliffe, T. Weber, and A. Tamada. TOOLympics 2019: An Overview of Competitions in Formal Methods. In T. Vojnar and L. Zhang, editors, *Proceedings of the 2019 International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, number 11429 in Lecture Notes in Computer Science, page To appear. Springer-Verlag, 2019.

[3] C. Benzmüller and B. Woltzenlogel Paleo. Automating Gödel's Ontological Proof of God's Existence with Higher-order Automated Theorem Provers. In T. Schaub, editor, *Proceedings of the 21st European Conference on Artificial Intelligence*, pages 93–98, 2014.

[4] D. Beyer, S. Löwe, and P. Wendler. Reliable Benchmarking: Requirements and Solutions. *International Journal on Software Tools for Technology Transfer*, 21:1–29, 2019.

[5] A. Bruni, E. Drewsen, and C. Schürmann. Towards a Mechanized Proof of Selene Receipt-Freeness and Vote-Privacy. In R. Krimmer, M. Volkamer, N. Braun Binder, N. Kersting, O. Pereira, and C. Schürmann, editors, *Proceedings of the International Joint Conference on Electronic Voting, E-Vote-ID 2017*, number 10615 in Lecture Notes in Computer Science, pages 110–126. Springer-Verlag, 2017.

[6] M. Caminati, M. Kerber, C. Lange, and C. Rowat. Sound Auction Specification and Implementation. In M. Feldman, M. Schwarz, and T. Roughgarden, editors, *Proceedings of the 16th ACM Conference on Economics and Computation*, pages 547–564. ACM Press, 2015.

[7] V. Chaudri, B. Cheng, A. Overholtzer, J. Roschelle, A. Spaulding, P. Clark, M. Greaves, and D. Gunning. Inquire Biology: A Textbook that Answers Questions. *AI Magazine*, 34(3), 2013.

[8] D. Cok, A. Stump, and T. Weber. The 2013 Evaluation of SMT-COMP and SMT-LIB. *Journal of Automated Reasoning*, 55(1):61–90, 2015.

[9] B. Cook. Formal Reasoning About the Security of Amazon Web Services. In H. Chockler and G. Weissenbacher, editors, *Proceedings of the 30th International Conference on Computer Aided*

Verification, number 10981 in Lecture Notes in Computer Science, pages 38–47. Springer-Verlag, 2018.

[10] L. Dennis, M. Fisher, M. Slavkovik, and M. Webster. Formal Verification of Ethical Choices in Autonomous Systems. *Robotics and Autonomous Systems*, 77:1–14, 2016.

[11] R. Hähnle and M. Huisman. Deductive Software Verification: From Pen-and-Paper Proofs to Industrial Tools. In B. Steffen and G. Woeginger, editors, *Computing and Software Science: State of the Art and Perspectives*, number 10000 in Lecture Notes in Computer Science, pages 345–373. Springer-Verlag, 2019.

[12] M.T. Hannan. Rethinking Age Dependence in Organizational Mortality: Logical Formalizations. *American Journal of Sociology*, 104:126–164, 1998.

[13] J. Harrison. Floating-Point Verification using Theorem Proving. In M. Bernardo and A. Cimatti, editors, *Proceedings of the 6th International School on Formal Methods for the Design of Computer, Communication, and Software Systems*, number 3965 in Lecture Notes in Computer Science, pages 211–242. Springer-Verlag, 2006.

[14] S. Holden. Connect++: A New Automated Theorem Prover Based on the Connection Calculus. In J. Otten and W. Bibel, editors, *Proceedings of the 1st International Workshop on Automated Reasoning with Connection Calculi*, number 3613 in CEUR Workshop Proceedings, pages 95–106, 2023.

[15] A. Hommersom, P. Lucas, and P. van Bommel. Automated Theorem Proving for Quality-checking Medical Guidelines. In G. Sutcliffe, B. Fischer, and S. Schulz, editors, *Proceedings of the Workshop on Empirically Successful Classical Automated Reasoning, 20th International Conference on Automated Deduction*, 2005.

[16] H. Hoos and T. Stützle. SATLIB: An Online Resource for Research on SAT. In I. Gent, H. van Maaren, and T. Walsh, editors, *Proceedings of the 3rd Workshop on the Satisfiability Problem*, pages 283–292. IOS Press, 2000.

[17] J. Horner. A Computationally Assisted Reconstruction of an Ontological Argument in Spinoza's The Ethics. *Open Philosophy*, 2:219–229, 2019.

[18] K. Korovin. Implementing an Instantiation-based Theorem Prover for First-order Logic. In C. Benzmüller, B. Fischer, and G. Sutcliffe, editors, *Proceedings of the 6th International Workshop on the Implementation of Logics*, number 212 in CEUR Workshop Proceedings, pages 63–63, 2006.

[19] T. Libal. Towards Automated GDPR Compliance Checking. In F. Heintz, M. Milano, and B. O'Sullivan, editors, *Proceedings of the International Workshop on the Foundations of Trustworthy AI Integrating Learning, Optimization and Reasoning*, number 12641 in Lecture Notes in Computer Science, pages 3–19, 2020.

[20] C. Marché and H. Zantema. The Termination Competition. In F. Baader, editor, *Proceedings of the 18th International Conference on Term Rewriting and Applications*, number 4533 in Lecture Notes in Computer Science, pages 303–313, 2007.

[21] W.W. McCune. Otter 3.3 Reference Manual. Technical Report ANL/MSC-TM-263, Argonne National Laboratory, Argonne, USA, 2003.

[22] T. Nipkow. Social Choice Theory in HOL: Arrow and Gibbard-Satterthwaite. *Journal of Automated Reasoning*, 43(3):289–304, 2009.

[23] P. Oppenheimer and E. Zalta. A Computationally-Discovered Simplification of the Ontological Argument. *Australasian Journal of Philosophy*, 89(2):333–349, 2011.

[24] J. Otten. 20 Years of leanCoP - An Overview of the Provers. In J. Otten and W. Bibel, editors, *Proceedings of the 1st International Workshop on Automated Reasoning with Connection Calculi*, number 3613 in CEUR Workshop Proceedings, pages 4–22, 2023.

[25] H. Prakken and G. Sartor. Law and Logic: A Review from an Argumentation Perspective. *Artificial Intelligence*, 227:214–245, 2015.

[26] A. Riazanov and A. Voronkov. The Design and Implementation of Vampire. *AI Communications*, 15(2-3):91–110, 2002.

[27] S. Schulz. Algorithms and Data Structures for First-Order Equational Deduction. In C. Benzmüller, B. Fischer, and G. Sutcliffe, editors, *Proceedings of the 6th International Workshop on the Implementation of Logics*, number 212 in CEUR Workshop Proceedings, pages 1–6, 2006.

[28] S. Schulz. Simple and Efficient Clause Subsumption with Feature Vector Indexing. In M.P. Bonacina and M. Stickel, editors, *Automated Reasoning and Mathematics: Essays in Memory of William W. McCune*, number 7788 in Lecture Notes in Artificial Intelligence, pages 45–67. Springer-Verlag, 2013.

[29] S. Schulz, S. Cruanes, and P. Vukmirović. Faster, Higher, Stronger: E 2.3. In P. Fontaine, editor, *Proceedings of the 27th International Conference on Automated Deduction*, number 11716 in Lecture Notes in Computer Science, pages 495–507. Springer-Verlag, 2019.

[30] S. Schulz and A. Pease. Teaching Automated Theorem Proving by Example: PyRes 1.2 (system description). In N. Peltier and V. Sofronie-Stokkermans, editors, *Proceedings of the 10th International Joint Conference on Automated Reasoning*, number 12167 in Lecture Notes in Computer Science, pages 158–166, 2020.

[31] A. Steen. Scala TPTP Parser v1.5, 2021. DOI: 10.5281/zenodo.5578872.

[32] A. Steen and C. Benzmüller. The Higher-Order Prover Leo-III. In D. Galmiche, S. Schulz, and R. Sebastiani, editors, *Proceedings of the 8th International Joint Conference on Automated Reasoning*, number 10900 in Lecture Notes in Artificial Intelligence, pages 108–116, 2018.

[33] A. Stump, G. Sutcliffe, and C. Tinelli. StarExec: a Cross-Community Infrastructure for Logic Solving. In S. Demri, D. Kapur, and C. Weidenbach, editors, *Proceedings of the 7th International Joint Conference on Automated Reasoning*, number 8562 in Lecture Notes in Artificial Intelligence, pages 367–373, 2014.

[34] G. Sutcliffe. The SZS Ontologies for Automated Reasoning Software. In G. Sutcliffe, P. Rudnicki, R. Schmidt, B. Konev, and S. Schulz, editors, *Proceedings of the LPAR Workshops: Knowledge Exchange: Automated Provers and Proof Assistants, and the 7th International Workshop on the Implementation of Logics*, number 418 in CEUR Workshop Proceedings, pages 38–49, 2008.

[35] G. Sutcliffe. The TPTP World - Infrastructure for Automated Reasoning. In E. Clarke and A. Voronkov, editors, *Proceedings of the 16th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, number 6355 in Lecture Notes in Artificial Intelligence, pages 1–12. Springer-Verlag, 2010.

[36] G. Sutcliffe. The CADE ATP System Competition - CASC. *AI Magazine*, 37(2):99–101, 2016.

[37] G. Sutcliffe. The TPTP Problem Library and Associated Infrastructure. From CNF to TH0, TPTP v6.4.0. *Journal of Automated Reasoning*, 59(4):483–502, 2017.

[38] G. Sutcliffe. Stepping Stones in the TPTP World. In C. Benzmüller, M. Heule, and R. Schmidt, editors, *Proceedings of the 12th International Joint Conference on Automated Reasoning*, Lecture Notes in Artificial Intelligence, page Invited paper, 2024.

[39] G. Sutcliffe, S. Schulz, K. Claessen, and A. Van Gelder. Using the TPTP Language for Writing Derivations and Finite Interpretations. In U. Furbach and N. Shankar, editors, *Proceedings of the 3rd International Joint Conference on Automated Reasoning*, number 4130 in Lecture Notes in Artificial Intelligence, pages 67–81. Springer, 2006.

[40] G. Sutcliffe and C.B. Suttner. Evaluating General Purpose Automated Theorem Proving Systems. *Artificial Intelligence*, 131(1-2):39–54, 2001.

[41] A' Voronkov. Algorithms, Datastructures, and Other Issues in Efficient Automated Deduction. In R. Gore, A. Leitsch, and T. Nipkow, editors, *Proceedings of the International Joint Conference on Automated Reasoning*, number 2083 in Lecture Notes in Artificial Intelligence, pages 13–28. Springer-Verlag, 2001.

[42] M. Yadav. On the Synthesis of Machine Learning and Automated Reasoning for an Artificial Synthetic Organic Chemist. *New Journal of Chemistry*, 41(4):1411–1416, 2017.

A run_image.py

```
#------------------------------------------------------------------------------
#!/usr/bin/env python3

import argparse
import subprocess
import os, sys
import shutil


def getRunsolverArgs(args):
    mem_part = f" -M {args.memory_limit}" if args.memory_limit > 0 else ""
    return "--timestamp --watcher-data /dev/null -C " + \
f"{args.cpu_limit} -W {args.wall_clock_limit}{mem_part}"


def getRunscriptArgs(args, args_format):
    parts = {
        'P': "/artifacts/CWD/benchmark",
        'C': args.cpu_limit,
        'W': args.wall_clock_limit,
        'I': args.intent,
        'M': args.memory_limit,
    }
    return ' '.join([str(parts[c.upper()]) for c in args_format])

def makeBenchmark(problem):
    if problem:
        shutil.copy(problem, "./benchmark")
    else:
        with open('./benchmark', 'w') as benchmark:
            benchmark.write(sys.stdin.read())


if __name__ == "__main__":
    parser = argparse.ArgumentParser("Wrapper for a podman call to a prover image")
    parser.add_argument("image_name",
help="Image name, e.g., eprover:3.0.03-runsolver-arm64")
    parser.add_argument("-P", "--problem",
help="Problem file if not stdin")
    parser.add_argument("--runscript", default="run_system PCWMI",
help="System script and its args, e.g., 'run_E PWI', default=run_system PCWMI")
    parser.add_argument("-C", "--cpu-limit", default=60, type=int,
help="CPU time limit in seconds, default=60")
    parser.add_argument("-W", "--wall-clock-limit", default=60, type=int,
help="Wall clock time limit in seconds, default=60")
    parser.add_argument("-M", "--memory-limit", default=-1, type=int,
help="Memory limit in MB, default=none")
    parser.add_argument("-I", "--intent", default="THM", choices=["THM", "SAT"],
help="Intention (THM, SAT, etc), default=THM")
    parser.add_argument("--dry-run", action="store_true",
help="dry run")
```

```
    args = parser.parse_args()

    # Format arguments
    runsolverArgs = getRunsolverArgs(args)
    runscript, runscriptArgsFormat = args.runscript.split()
    runscriptArgs = getRunscriptArgs(args, runscriptArgsFormat)

    # Construct podman command
    command = "podman run -v .:/artifacts/CWD -t " + \
f"{args.image_name} {runsolverArgs} {runscript} {runscriptArgs}"

    # Run command or print for dry run
    if args.dry_run:
        print(command)
    else:
        makeBenchmark(args.problem)
        subprocess.run(command, shell=True)
        os.remove("./benchmark")
#------------------------------------------------------------------------------
```