Foundational ontologies meet automatic theorem provers

Pawel Garbacz[1]

[1]The John Paul II Catholic University of Lublin

Abstract

The paper exemplifies various problems one may encounter using automatic proving systems for upper-level formal ontologies. I report here a number of results on using such tools for BFO, DOLCE, GUM, TUpper, UFO, and YAMATO ontologies.

Keywords

upper-level ontology, prover, model finder, decidability

1. Introduction

Recent years brought about the important shift from formal to computational upper-level ontologies. As a result, one may use a theorem prover to prove that a given ontology is inconsistent or a model finder to provide a model that shows that the ontology is consistent. In principle, the ontologists can be assisted by software in various reasoning tasks – although the scope of this assistance is limited by the well-known meta-theorems, in particular by the fact that the set of first-order tautologies is undecidable.[1]

The recent volume of Applied Ontology, Applied Ontology 17 (2022), evidences this shift – out of seven foundational ontologies presented there, at least six are formalised in a language that is prover-friendly.[2] As a philosopher by training, I cannot resist here an observation that computer scientists are yet again behind philosophers as far as the use of software is concerned because the idea of using automatic proving systems in ontology was first tested 15 years ago in [2].[3]

The Eighth Joint Ontology Workshops (JOWO'22), August 15-19, 2022, Jönköping University, Sweden
garbacz@kul.pl (P. Garbacz)
0000-0002-0877-7063 (P. Garbacz)

[1]I am not making here the claim, obviously false, that any set of first-order formulae is undecidable, but just that using the full expressivity of first-order logic, as opposed to some description logic languages, *may* lead to an undecidable applied ontology.

[2]The only one that is not explicitly "automatable" is GFO. [1] does not mention any prover-friendly formalisation of this ontology. The paper points to https://www.onto-med.de/gfo website, but this is a dead link. Some OWL versions of this ontology are available from https://www.onto-med.de/ontologies, but the most recent there is dated to 2007. For these reasons I decided not to include this ontology in this review.

[3]Incidentally, the whole point of doing ontology *using* just calculations goes back at least to G. W. Leibniz:

> Quo facto quando orientur controversiae, non magis disputatione opus erit inter duos philosophos, quam inter duos Computistas. Sufficiet enim calamos in manus sumere sedereque ad abacos, et sibi mutuo (: accito si placet amico :) dicere: calculemus. (G. W. Leibniz, *De arte characteristica ad*

Factoring in the practical context of all applied ontologies, one should consider not just the theoretical limits of automatic reasoning but also the practical requirements thereof. There probably exists, for example, a limit of CPU time or memory that one is ready to allocate for a simple consistency check for an applied ontology. And if one needs more time or memory, one is justified in assuming that the problem of its consistency is, given the technical infrastructure at hand, practically undecidable even if it is decidable in polynomial time or even if more powerful hardware setup or more time would allow a theorem prover decide this problem.

The current paper investigates a couple of such issues of practical decidability using the aforementioned foundational ontologies as case studies. All issues are related to the task of checking whether a given ontology is a consistent theory. The results reported here are documented in the https://github.com/mereolog/computational_ontologies/ repository.

2. *Calculemeus!*

Since the computational resources are crucial for the results I report below, let me note that all tests were run on a MacBook Pro (Intel Core i7 with six 2.6 GHz cores and 16 GB of DDR4 RAM).

BFO Basic Formal Ontology is a top-level ontology designed to support information integration, retrieval, and analysis across all domains of scientific investigation – for the most recent exposition see [3]. Previous versions of BFO were either partially formalised or restricted to the DL dialect of OWL. Fairly recently, the ontological community got access to the formalisation of BFO in Common Logic and in the syntax of Prover9/Mace4 application – see https://github.com/BFO-ontology/BFO-2020. Because the latter version allows for a broader selection of provers, I will make use of it in this paper. I also should note that I used the version of BFO from commit `8792dde`.

When one concatenates all files that contain Prover9/Mace4 BFO axioms, one can get a single theory that can be run through the Prover9/Mace4 application. I ran the command-line versions thereof from the `LADR-2009-11A` build. Unfortunately, running both Prover9 and Mace4, with the maximal memory allocation available in the system, on this theory did not produce any conclusive results. Mace4 terminated with an out of memory error while it was investigating models of 7 elements. This is worth mentioning because the BFO axioms in `universal-declaration.prover9` file imply that any model thereof should have at least 35 elements – these are universals whose existence is stated by BFO, e.g., continuant, occurrent, role, etc. Prover9's process was terminated by me after approx. 3 hours with no inconsistency proof.

Since Prover9/Mace4 comes with the `ladr_to_tptp` script, I was able to translate the Prover9/Mace4 BFO axioms into the TPTP BFO axioms. Then I used all appropriate systems available from https://www.tptp.org/cgi-bin/SystemOnTPTP to check the consistency of this theory (e.g., Darwin, E, iProver, Isabelle, Paradox, SPASS, Vampire, Z3), but each system timed out despite the fact that I set the maximal timeout of 999 seconds.

perficiendas scientias ratione nitentes, 1688)

Finally, I ran the full TPTP BFO locally through the system mentioned above the Vampire prover (build `f9f55df` from https://github.com/vprover/vampire) with these settings. `-t 3600 --mode casc_sat`. The prover did not find any conclusion after the specified one-hour limit.

The next step was to check subtheories of the TPTP BFO theory. Given the fact that it consists of 369 axioms, one cannot check all its subtheories in a reasonable time. Even if we do not check any subtheory of a consistent theory and any supertheory of an inconsistent theory, the sheer number of remaining checks makes this task practically impossible. Suppose, for example, that one needs to check all subtheories with 367 axioms – in the "worst" case scenario this would require checking 67896 such theories, which may take days using the average ontologist's hardware setup. Fortunately and interestingly enough, all subtheories of the size 368 timed out with the exception of one. If you remove the following axiom from the TPTP BFO theory, the resulting subtheory is shown to be consistent by the Vampire prover, which I ran again locally (using the option `--mode casc_sat`).[4]

The "offending" axiom states that every universal had, has or will have at least one instance:

```
% Every universal is instantiated at least once
all u  ((universal(u)) -> (exists p exists t  (instanceOf(p,u,t))))
```

The default setting of Vampire, which I used, is that the prover times out after 60 seconds. In principle one could increase this limit but note that even with the default setting this exercise is computationally expensive since it took 368 minutes to complete.

Since all subtheories in which this axiom is present time out after 60 seconds and the only theory in which it is absent is found consistent in less than 1 second within the same system, this seems to implicate that this axiom is a performance hotspot – at least as far as the basic reasoning tasks are concerned. Since the axiom is usually considered as an essential part of ontological realism (cf. [4, p. 13-14]), one may say, rather whimsically, that realism is computationally expensive, i.e., realism increases the computational complexity of the decidability problem of any consistent theory that expresses it.

Obviously, it may be the case that there is a subtheory of the TPTP BFO theory with 367 axioms which (i) includes the axiom and (ii) can be shown to be consistent. The problem is that to find such a subtheory in a systematic and exhaustive way is again computationally expensive, as indicated above.

It is worth mentioning that the only "decided" subtheory of the TPTP BFO theory, which is identified as 70eb4f4ad57c832a484346a04a81c4fd in https://github.com/mereolog/computational_ontologies/tree/main/python/computable_ontologies, produced the same error as before.

I was not able to decide the logical status of BFO when the axiom above was replaced with a weaker claim:

```
% Some universal is instantiated at least once.
exists u  (universal(u) & (exists p exists t  (instanceOf(p,u,t))))
```

This weak version of BFO timed out on all suitable provers available from https://www.tptp.org/cgi-bin/SystemOnTPTP.

[4]These results were found using a short Python script available in the ./main/python/computable_ontologies folder in the https://github.com/mereolog/computational_ontologies/ repository.

DOLCE DOLCE is a foundational ontology based on cognitive and linguistic principles. It aims to model a commonsense view of reality, like the one human beings exploit in everyday life in areas as diverse as socio-technical systems, manufacturing, financial transactions and cultural heritage ([5]).

The current formalisation of DOLCE is provided by the link http://www.loa.istc.cnr.it/wp-content/uploads/2021/07/dolce-mace4-prover9.zip, where one can get Prover9/Mace4 input file that indeed contains a consistent first-order theory.[5]

GUM GUM is a linguistically-motivated ontology originally developed to support natural language processing systems by offering a level of representation intermediate between linguistic forms and domain knowledge – see [8]. The ontology went through several rounds of expansion, which affected not just its content or name, but also the methodological principles that its developers followed.

The current OWL version of GUM is provided by the link http://www.fb10.uni-bremen.de/anglistik/langpro/webspace/jb/gum/index.htm. Two ontologies accessible there can be proved consistent by an OWL reasoner, e.g., by Pellet. In addition the reasoner also can check for unsatisfiable classes – in the case of these theories, there are none. The link also reveals a CASL version of the GUM ontology – unfortunately there are probably some syntactic issues with it because HETS reports the following error on reading this file:

```
GUM:644.1-644.4: *** Error: illegal CASL morphism
```

TUpper The TUpper Ontology is a top-level ontology that supports the ontological analysis of relevant existing standards, in particular ([9]):

- ISO 18629
- ISO 19150
- ISO 80000
- OWL-Time

The formalisation of TUpper ontology is provided as the repository https://github.com/gruninger/colore of CLIF files. The commit `bcb03f3` there, which I made use of in this paper, contains two sorts of issues:

1. CLIF files contain import references to the CLIF modules that do not resolve as such, e.g., `embed_occupy.clif` imports http://colore.oor.net/occupy/occupy_root.clif text, but the latter resolves to an HTML page and not the CLIF file itself – this feature may confuse some provers, e.g., HETS complains about a missing file in such cases.
2. There is a number of files that attempt to import http://colore.oor.net/psl_actocc/actocc.clif, which gives a 404 error. The correct reference should be http://colore.oor.net/psl_actocc/psl_actocc.clif.

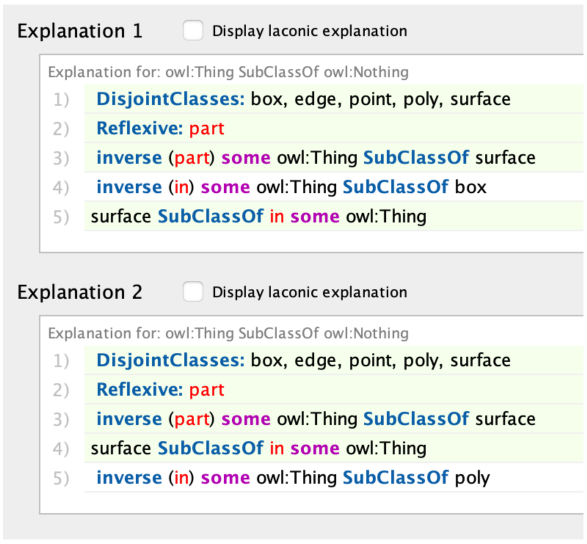
[5]The consistency of the first "release" of DOLCE, i.e., from [6], was proven in [7]. I ignore in this report simplified, OWL versions of this ontology, e.g., http://www.loa.istc.cnr.it/old/ontologies/DLP3971.zip. On the other hand, applying the changes described in paragraph 2 below, I was able to prove that the theory available in https://github.com/gruninger/colore/blob/master/ontologies/dolce/dolce_colore.clif is consistent using the Darwin prover within HETS system http://hets.eu.

Now [9] contains a list of modules for TUpper – however a number of links there seem to be outdated, e.g., colore.oor.net/occupy/occupy_mereology.clif. That's why instead of guessing how to update these links I decided to use http://colore.oor.net/tupper/tupper.clif as a version of TUpper to be tested. After the appropriate corrections related to the aforementioned issues, eprover run within the HETS application found this ontology inconsistent.[6] The same result was produced by Vampire run within the https://www.tptp.org/cgi-bin/SystemOnTPTP hub for a TPTP translation of `tupper.clif` produced by Macleod https://github.com/thahmann/macleod. The proof can be found in the appendix below. Looking at the proof I came to the conclusion that a source[7] of TUpper's inconsistency are the following two axioms:

```
285. ! [X0] : (timepoint(X0) => ? [X1] : (before(X0,X1) & timepoint(X1)))
291. ! [X50] : (('INF+' != X50 & timepoint(X50)) => before(X50,'INF+'))
```

Axiom (labelled by Vampire as) `285` says that every time point is `before` some time point. Axiom `291` refers to the least upper bound of `before` relation, implicitly affirming the existence of a time point such that all (other) time points are `before` it. Obviously, to show that these two axioms cannot belong to a consistent theory you need other axioms indicated in the proof, but these two indicate the conflicting properties of `before` relation postulated by TUpper.

Interestingly enough, the repository https://github.com/gruninger/colore contains also an OWL version of TUpper, which again can be shown to be inconsistent – I refer here to the ontology available at https://github.com/gruninger/colore/blob/master/ontologies/tupper/owl/tupper.all.owl. The Hermit explains this inconsistency as shown in Fig. 1.


Explanation 1 — Display laconic explanation

Explanation for: owl:Thing SubClassOf owl:Nothing
1) DisjointClasses: box, edge, point, poly, surface
2) Reflexive: part
3) inverse (part) some owl:Thing SubClassOf surface
4) inverse (in) some owl:Thing SubClassOf box
5) surface SubClassOf in some owl:Thing

Explanation 2 — Display laconic explanation

Explanation for: owl:Thing SubClassOf owl:Nothing
1) DisjointClasses: box, edge, point, poly, surface
2) Reflexive: part
3) inverse (part) some owl:Thing SubClassOf surface
4) surface SubClassOf in some owl:Thing
5) inverse (in) some owl:Thing SubClassOf poly

Figure 1: tupper.all.owl is inconsistent

[6]Let me note in passing that using a different prover, i.e., darwin, within the same application produced a time out error after 1 hour.

[7]I said *a* source, and not *the* source, because there might be other sources besides the one found by Vampire – see, for instance, the inconsistency result for the OWL version of TUpper.

UFO UFO is meant to provide foundations for domain analysis in conceptual modelling, for designing concrete models and modelling grammars. It may be called a "calculus of content" aimed at supporting the ontological analysis, conceptual clarification, and semantic explicitation of the content embedded in representation artifacts – see [10].

The TPTP formalisation of UFO is provided via the https://github.com/unibz-core/ufo-formalization.git repository. The commit `2da0df3` there, which I made use of in this paper, contains a typo in one of the axioms.

Instead of

```
  fof(ax_endurantTypeCharacterizationByMomentTypes_a83, axiom, (
 ![ET,MT]: (characterizes(MT,ET) => (
  endurantType(ET)
  & momentType(M)
  & (![E,W]: (iof(E,ET,W) => (?[M]: (iof(M,MT,W) & inheresIn(M,E)))))
  & (![M2,W2]: (iof(M2,MT,W2) => (?[E2]: (iof(E2,ET,W2) & inheresIn(M2,E2)))))
 ))
)).
```

one should read

```
  fof(ax_endurantTypeCharacterizationByMomentTypes_a83, axiom, (
 ![ET,MT]: (characterizes(MT,ET) => (
  endurantType(ET)
  & momentType(MT)
  & (![E,W]: (iof(E,ET,W) => (?[M]: (iof(M,MT,W) & inheresIn(M,E)))))
  & (![M2,W2]: (iof(M2,MT,W2) => (?[E2]: (iof(E2,ET,W2) & inheresIn(M2,E2)))))
 ))
)).
```

After this axiom is fixed the Paradox prover finds a one-element model of this theory.

YAMATO YAMATO is promoted as an alternative to other upper-level ontologies - it separates itself from them in the following aspects ([11]):

- account of roles and functions
- differences between events and processes
- conception of artifact
- theory of informational entities.

Although the canonical version of YAMATO is a first-order theory, for now its only computer-readable version is rendered in OWL – following [11] I used the ontology version from http://www.hozo.jp/onto_library/download.php?filename=YAMATO20210808_owl.zip.[8] Unfortunately, this OWL ontology is not OWL 2 DL compliant, so the usual OWL reasoners cannot reason over it – below I copied the error thrown by the Hermit reasoner:

[8]More precisely, speaking YAMATO is only partially axiomatised, but even the current formalisation goes beyond the limits of OWL. For instance, the axioms like axiom A7 in [11], which states that processes are additive with respect to parthood, cannot be fully expressed in a description logic language.

```
An error occurred during reasoning: Non-simple property
'has_actor_achievement_action.doer.initial_state' or its inverse
appears in the cardinality restriction
'has_actor_achievement_action.doer.initial_state max 1 Thing'.
```

3. Lessons learned

A sufficiently rich foundational ontology may turn out to be practically undecidable given a certain reasoning context. The results reported in this paper show that whether a set of first-order formulas can be automatically “reasoned over” depends, at least, on the following factors:

1. prover (application)
2. and its settings
3. hardware infrastructure available
4. time limitations.

Thus, even if one cannot check whether a given ontology is consistent because of a timeout or out-of-memory error, a different prover or a more powerful computer or a longer, but still reasonable, time may conclusively show whether a given yet undecided theory is consistent. Nevertheless the aforementioned tests performed on the BFO ontology indicate that the context in which such a task is accomplishable may be rather demanding. Incidentally, let me observe that the relatively small set of theories revealed all possible outcomes from an automatic reasoning system:

1. a theory is consistent
2. a theory is inconsistent
3. the system’s resource limits were exceeded
4. a theory is out of the system’s scope.

Using currently available computer tools for automatic inference requires a specific level of precision. As of now even a simple typo may make it impossible to automatically verify whether a given piece of theory is consistent or not. Nevertheless, each such issue, if identified by the reasoner and fixed, will help the ontologists to improve the quality of their artefacts. It goes without saying that the bigger and the older an ontology gets, the more likely it is that such blunders may arise.

Since an applied formal ontology is after all an informational artifact that evolves over time, some kind of version control environment seems to be a must if one wants to document its development in an auditable manner. As we saw above, some (but yet not all!) foundational ontologies already meet this requirement using git. But simple, one-size-fits-all versioning may not be enough, and on top of it one may want to run a set of automatic checks *before* a change is made to the ontology, e.g., to verify that the change will not make the ontology inconsistent or practically undecidable. [12] gives an example of such infrastructure.

References

[1] F. Loebe, P. Burek, H. Herre, GFO: The General Formal Ontology, Applied Ontology 17 (2022) 71–106.

[2] B. Fitelson, E. N. Zalta, Steps toward a computational metaphysics, Journal of Philosophical Logic 36 (2007) 227–247.

[3] J. N. Otte, J. Beverley, A. Ruttenberg, BFO: Basic Formal Ontology, Applied Ontology 17 (2022) 17–43.

[4] R. Arp, B. Smith, A. D. Spear, Building Ontologies with Basic Formal Ontology, MIT Press, 2015.

[5] J. A. Bateman, GUM: The generalized upper model, Applied Ontology 17 (2022) 45–69.

[6] A. Gangemi, N. Guarino, C. Masolo, A. Oltramari, L. Schneider, Sweetening ontologies with dolce, in: A. Gómez-Pérez, V. R. Benjamins (Eds.), Knowledge Engineering and Knowledge Management: Ontologies and the Semantic Web, Springer Berlin Heidelberg, Berlin, Heidelberg, 2002, pp. 166–181.

[7] O. Kutz, T. Mossakowski, A modular consistency proof for DOLCE, in: W. Burgard, D. Roth (Eds.), Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, August 7-11, 2011, AAAI Press, 2011, pp. 227–234. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI11/paper/view/3754.

[8] S. Borgo, R. Ferrario, A. Gangemi, N. Guarino, C. Masolo, D. Porello, E. M. Sanfilippo, L. Vieu, DOLCE: A descriptive ontology for linguistic and cognitive engineering, Applied Ontology 107-141 (2022) 45–69.

[9] M. Grüninger, Y. Ru, J. Thai, TUpper: A top level ontology within standards, Applied Ontology 17 (2022) 143–165.

[10] G. Guizzardi, A. B. Benevides, C. M. Fonseca, D. Porello, J. P. A. Almeida, T. P. Sales, UFO: Unified Foundational Ontology, Applied Ontology 107-141 (2022) 167–210.

[11] R. Mizoguchi, S. Borgo, YAMATO: Yet-another more advanced top-level ontology, Applied Ontology 17 (2022) 211–232.

[12] D. Allemang, P. Garbacz, P. Grądzki, E. Kendall, R. Trypuz, An infrastructure for collaborative ontology development, in: Formal Ontology in Information Systems, IOS Press, 2021, pp. 112–126.

Appendix

The Vampire proof of TUpper's inconsistency:

```
259. ! [X17] : (timeduration(X17) =>
     ? [X46] : ? [X47] : (duration(X46,X47) = X17 & timepoint(X47) & timepoint(X46)))
     [input axiom1240]
265. timeduration('ZERO_DURATION') [input axiom1360]
282. ! [X0] : (timepoint(X0) => ~before(X0,X0)) [input axiom1320]
285. ! [X0] : (timepoint(X0) => ? [X1] : (before(X0,X1) & timepoint(X1))) [input axiom1300]
286. ! [X46] : ! [X47] : (before(X46,X47) => (timepoint(X47) & timepoint(X46))) [input axiom280]
288. ! [X46] : ~before(X46,X46) [input axiom300]
```

```
289. ! [X46] : ! [X47] : ! [X48] : ((before(X47,X48) & before(X46,X47)) => before(X46,X48))
     [input axiom310]
291. ! [X50] : (('INF+' != X50 & timepoint(X50)) => before(X50,'INF+')) [input axiom330]
728. ! [X0] : (timeduration(X0) =>
     ? [X1] : ? [X2] : (duration(X1,X2) = X0 & timepoint(X2) & timepoint(X1))) [rectify 259]
729. ! [X0] : (timeduration(X0) =>
     ? [X1,X2] : (duration(X1,X2) = X0 & timepoint(X2) & timepoint(X1))) [flattening 728]
769. ! [X0] : ! [X1] : (before(X0,X1) => (timepoint(X1) & timepoint(X0))) [rectify 286]
770. ! [X0,X1] : (before(X0,X1) => (timepoint(X1) & timepoint(X0))) [flattening 769]
773. ! [X0] : ~before(X0,X0) [rectify 288]
774. ! [X0] : ! [X1] : ! [X2] : ((before(X1,X2) & before(X0,X1)) => before(X0,X2)) [rectify 289]
775. ! [X0,X1,X2] : ((before(X1,X2) & before(X0,X1)) => before(X0,X2)) [flattening 774]
777. ! [X0] : (('INF+' != X0 & timepoint(X0)) => before(X0,'INF+')) [rectify 291]
1273. ! [X0] : (? [X1,X2] : (duration(X1,X2) = X0 & timepoint(X2) & timepoint(X1)) | ~timeduration(X0))
      [ennf transformation 729]
1300. ! [X0] : (~before(X0,X0) | ~timepoint(X0))
      [ennf transformation 282]
1303. ! [X0] : (? [X1] : (before(X0,X1) & timepoint(X1)) | ~timepoint(X0)) [ennf transformation 285]
1304. ! [X0,X1] : ((timepoint(X1) & timepoint(X0)) | ~before(X0,X1)) [ennf transformation 770]
1307. ! [X0,X1,X2] : (before(X0,X2) | (~before(X1,X2) | ~before(X0,X1))) [ennf transformation 775]
1308. ! [X0,X1,X2] : (before(X0,X2) | ~before(X1,X2) | ~before(X0,X1)) [flattening 1307]
1311. ! [X0] : (before(X0,'INF+') | ('INF+' = X0 | ~timepoint(X0))) [ennf transformation 777]
1312. ! [X0] : (before(X0,'INF+') | 'INF+' = X0 | ~timepoint(X0)) [flattening 1311]
1604. ! [X0] : (? [X1,X2] : (duration(X1,X2) = X0 & timepoint(X2) & timepoint(X1)) =>
      (duration(sK67(X0),sK68(X0)) = X0 & timepoint(sK68(X0)) & timepoint(sK67(X0)))) [choice axiom]
1605. ! [X0] : ((duration(sK67(X0),sK68(X0)) = X0 & timepoint(sK68(X0)) & timepoint(sK67(X0))) |
      ~timeduration(X0)) [skolemisation 1273,1604]
1614. ! [X0] : (? [X1] : (before(X0,X1) & timepoint(X1)) => (before(X0,sK72(X0)) & timepoint(sK72(X0))))
      [choice axiom]
1615. ! [X0] : ((before(X0,sK72(X0)) & timepoint(sK72(X0))) | ~timepoint(X0)) [skolemisation 1303,1614]
2071. timepoint(sK67(X0)) | ~timeduration(X0) [cnf transformation 1605]
2082. timeduration('ZERO_DURATION') [cnf transformation 265]
2095. ~before(X0,X0) | ~timepoint(X0) [cnf transformation 1300]
2097. timepoint(sK72(X0)) | ~timepoint(X0) [cnf transformation 1615]
2098. before(X0,sK72(X0)) | ~timepoint(X0) [cnf transformation 1615]
2099. ~before(X0,X1) | timepoint(X0) [cnf transformation 1304]
2100. ~before(X0,X1) | timepoint(X1) [cnf transformation 1304]
2102. ~before(X0,X0) [cnf transformation 773]
2103. ~before(X1,X2) | before(X0,X2) | ~before(X0,X1) [cnf transformation 1308]
2105. before(X0,'INF+') | 'INF+' = X0 | ~timepoint(X0) [cnf transformation 1312]
2823. 'INF+' = X0 | ~timepoint(X0) | timepoint('INF+') [resolution 2105,2100]
2827. 16 <=> timepoint('INF+') [avatar definition]
2829. timepoint('INF+') <- (16) [avatar component clause 2827]
2831. 17 <=> ! [X0] : ('INF+' = X0 | ~timepoint(X0)) [avatar definition]
2832. ~timepoint(X0) | 'INF+' = X0 <- (17) [avatar component clause 2831]
2833. 16 | 17 [avatar split clause 2823,2831,2827]
3056. before(X0,sK72(X1)) | ~before(X0,X1) | ~timepoint(X1) [resolution 2103,2098]
3061. before(X0,sK72(X1)) | ~before(X0,X1) [subsumption resolution 3056,2100]
3063. ~before(sK72(X0),X0) | ~timepoint(sK72(X0)) [resolution 3061,2095]
```

```
3068. ~before(sK72(X0),X0) [subsumption resolution 3063,2099]
3073. 'INF+' = sK72('INF+') | ~timepoint(sK72('INF+')) [resolution 3068,2105]
3075. 18 <=> timepoint(sK72('INF+')) [avatar definition]
3077. ~timepoint(sK72('INF+')) <- (~18) [avatar component clause 3075]
3079. 19 <=> 'INF+' = sK72('INF+') [avatar definition]
3081. 'INF+' = sK72('INF+') <- (19) [avatar component clause 3079]
3082. ~18 | 19 [avatar split clause 3073,3079,3075]
3085. ~timepoint('INF+') <- (~18) [resolution 3077,2097]
3088. ~16 | 18 [avatar split clause 3085,3075,2827]
3096. ~timeduration(X5) | 'INF+' = sK67(X5) <- (17) [resolution 2832,2071]
3182. 'INF+' = sK67('ZERO_DURATION') <- (17) [resolution 3096,2082]
3189. timepoint('INF+') | ~timeduration('ZERO_DURATION') <- (17) [superposition 2071,3182]
3193. timepoint('INF+') <- (17) [subsumption resolution 3189,2082]
3194. 16 | ~17 [avatar split clause 3193,2831,2827]
3286. before('INF+','INF+') | ~timepoint('INF+') <- (19) [superposition 2098,3081]
3288. ~timepoint('INF+') <- (19) [subsumption resolution 3286,2102]
3289. $false <- (16, 19) [subsumption resolution 3288,2829]
3290. ~16 | ~19 [avatar contradiction clause 3289]
3291. $false [avatar sat refutation 2833,3082,3088,3194,3290]
```