1. **Title** *(30 seconds)*
   Hello! My name is Jan Gorzny, and I am here today to present the paper 'Towards the compression of first-order resolution proofs by lowering unit clauses'
   The work was done with my co-author Bruno, and was initiated during (and supported by) the Google Summer of Code program in 2014.

2. **Proof Compression Motivation** *(75 seconds)*
   As computer generated proofs become larger and larger, they get harder to use. Proofs generated by modern SAT and SMT solvers can contain thousands of clauses which make them impossible to understand by humans, and difficult to verify by computers. Worse, the best, most advanced provers still do not generate the best, least redundant proofs. Thus, post-generation methods for proof compression are helpful. When we consider propositional proofs, there are several algorithms that can be used; e.g., recycle and reconstruct and recycle pivots to name two. However, for first-order proofs, there are fewer choices. Most of these incur a cost overhead by transforming the proof first, or introducing external abbreviations to shorten the proof—these, of course, cannot be checked by a pure first-order resolution proof checker.

3. **A Propositional Proof** *(30 seconds)*
   Before we go any further, we should clarify what we consider a proof. A typical propositional proof is on the screen. Our proofs will be directed acyclic graphs, where clauses are represented using sequents. In the propositional case, the clauses will be proper sets, and contraction (the deletion of repeated literals) will be implicit. For first-order proofs, the clauses will be multi-sets and contraction will be explicit.

4. **LowerUnits** *(45 seconds)*
   The primary result of this work is the lifting of the *LowerUnits* algorithm from propositional logic to first-order logic. Lower units identifies units: subproofs with a conclusion that has exactly one literal, and then postpones resolution with these subproofs until lower in the proof. It can be shown that this is always possible. As a result of the lowering, we often require fewer resolutions.
   The algorithm traverses the graph twice: once, bottom up, to collect the units, and a second time, to delete them, and then reintroduce them to the bottom of the proof.

5. **Propositional Example** *(60 seconds)*
   To ensure that the idea is clear, we present an example. On the first pass, we identify the units that have multiple children in the proof, namely not-a and b, and then we start the second pass. We delete these nodes from the proof, and fix the proof: nodes which have a deleted parent are replaced by their non-deleted parent in a top-down manner.
   Now the proof is shorter, but incomplete, so we reintroduce our units. Note that the unit b was replaced with a non-unit, but this is okay. We also re-introduce the units in the order they were identified (from the top down). Now we have the following much shorter proof (the original proof had 6 resolutions, the compressed proof has 3).

6. **First-Order Change: Helpful Contractions** *(30 seconds)*
   Since in the first-order case, we have proper sets for clauses, and further, literals that may be unified but are not syntactically equal, we rely on contractions to reduce the number of resolutions. Here, we delete n1, and then the result proof has a node n5' that can be contracted, which reduces the number of resolutions by 1.

7. **First-Order Challenge: Pre-Deletion Check** *(90 seconds)*
   Given that contractions appear beneficial, it is natural to see if this is how the compression algorithm should be lifted. Thus, we ask, "does lowering a first-order unit always allow compression?"
   The answer to this is no, and is shown in this example.
   Thus, in order to lower a unit in the first-order case, it must satisfy the pre-deletion property: the property that all of the literals that are resolved against the candidate literal are unifiable together.

8. **First-Order Challenge: Post-Deletion Check** *(90 seconds)*
   The last property is not sufficient. For example, take a look at this proof where the marked node is to be deleted. After deletion, we find that contraction is not possible. However, it does satisfy the pre-deletion check. In the original proof, unification changes the literals resolved against n2 so that it can be re-used. Without these changes, these literals cannot be contracted. Thus n2 should not be marked for deletion.
   This is summarized as the post-deletion check: if a unit is to be lowered, the literals it is resolved against, must, after the unit is deleted from the proof, still be unifiable.

9. **First-Order Lower Units Challenges** *(30 seconds)*
   Thus, there is some difficulty with lifting this algorithm. The first, deletion changing literals, is overcome by book keeping. The implementation can track a literal, and how it changes as the new proof being constructed changes it.
   The second is harder: we only really know if a literal is safe to be lowered, after attempting to lower it. This can be overcome by a quadratic approach: delete each potential unit and attempt to lower it; but there may be a linear number of potential units.
   This makes such a solution harder to implement as well.

10. **Greedy First-Order Lower units – A Quicker Alternative** *(30 seconds)*
    Instead, we developed a fast, easy-to-implement algorithm. We focus only on the first property, and completely ignore the second property. And during proof construction, we always attempt to contract as much as possible.
    This allows us to achieve compression using a single traversal of the proof; thus the algorithm is linear in the size of the proof.
    But, since the first property is not always sufficient, we don't always get proof compression, and we instead return the original proof in these cases.

11. **First-Order Example** *(60 seconds)*
    An example execution is now presented.
    We are attempting to delete n1, which does satisfy the pre-deletion check.

12. **Experiment Setup** *(60 seconds)*
    To evaluate the algorithm's usefulness, we generated some proofs and observed its performance. The algorithm implemented as part of the Skeptik proof compression system.
    Evaluation was performed using 308 proofs. These proofs were generated on a cluster at the University of Victoria, using SPASS set to use only contraction and resolution inference rules, and with a 300 second timeout.
    SPASS was given 2280 problems from the TPTP problem library. Given its run-time constraints, it was able to generate the 308 proofs.
    Compression, on the other hand, was performed on this simple laptop.
    Compression took about 5 seconds, whereas generating the proofs required 40 minutes.

13. **Results I** *(60 seconds)*

    The first plot on the left shows the initial proof length along the horizontal axis plotted against compressed proof length on the vertical axis. The algorithm left many small proofs uncompressed, but larger proofs were more likely to be compressed, with several proofs resulting in significant compression. This is expected: the larger a proof is, the more likely it contains a unit that satisfies the pre-deletion property.

    The second plot shows the total number of proof nodes for the first n proofs, for the largest n. Thus the first value at 208 shows that there is approximately 1900 proof nodes in all of the first 208 proofs initially, and a bit less than that after compression. The proofs were sorted by size, so as the horizontal axis increases in value, we start to see significant savings gained by compression. This trend appears to increase superlinearly, so we expect that the performance will only improve on larger proofs.

14. **Results II** *(60 seconds)*

    In longer proofs, we saw more compression. Only 5 proofs with length larger than 30 were not compressed.

    The compression ratio, overall, was just over 11%. This is consistent with a 15% compression that was observed for the propositional case, especially when consider that when we consider only the largest proofs we had, it jumps up to just over 18%.

    Lastly, we note that the pre-deletion check is often good enough: only 14 proofs, in total, were returned without compression.

15. **Conclusion** *(90 seconds + 60 seconds for questions)*

    To conclude, we've developed a fast and easy to implement compression algorithm that appears to work very well. 5 seconds for 11% fewer nodes is not a large price to pay considering the initial generating time of 40 minutes. Of course, it should be evaluated on larger proofs as well.

    There is still work to be done: can we lift other compression algorithms? We think so. Also, can we overcome the post-deletion property efficiently to compress those additional 14 proofs?

    Thanks! Any questions?