Finding Applications for Partial Classical Logic

Mathematics is the queen of sciences. Where other sciences, like physics, or
astronomy have to rely on observation and experiment to obtain a reasonable level
of certainty, mathematicians are able to obtain absolute certainty
by proving their theorems.
Mathematical certainty is the highest level of certainty that
mankind has ever obtained. The Pythagorean theorem is
approximately 2500 years old, and its correctness has never
been doubted.

Mathematical proofs are logical arguments, consisting of small steps,
which can be checked by everyone. They are published in articles
or books, and everyone can read them and check the steps in
the proof. This is different from the other sciences, where one
usually needs expensive equipment in order to repeat experiments.

Unfortunately, in modern times, the situation is changing.
The reason for this is that mathematics has become very
complex. Proofs are sometimes so big and so difficult that there
is no single person who can check them completely.
Some proofs rely on other proofs that have been written by somebody else.
Different parts of the proof may use areas of mathematics.
Sometimes mathematicians take results from books as starting point of
their own work. In all cases we have proofs that are so complex,
that there is no person that can understand the complete proof.

The solution for this problem may come from computer science.
One can write down the proof in a computer language,
and let a computer check the proof.
There aleady exist languages in which this can be done, because
in the beginning of the 20th century, there was a foundational crisis
in mathematics. This crisis was caused by the fact that mathematicians
started considering more and more abstract and bigger structures,
like for example infinite sets, infinite sets of infinite sets,
the set of all infinite sets, etc.
These structures were so abstract that many mathematicians believed
that they cannot exist. When something does not exist, assuming
its existence is an absurdity. Since in mathematics, everything can be proven
from absurdity, the opponents argued that all this abstract mathematics
was possibly proven from absurdity, instead of being real mathematics.

This problem was eventually solved by introducing axiomatic set theory
in the time 1930-1940.
The definitions of the big objects were written down in a logical language,
and mathematicians went on to do what they like: Proving properties about them.
This means that mathematicians already know languages
and logical systems in which mathematical proofs can be checked
for more than 70 years. So what is stopping us from using this language? The
problem is that mathematicians, even when they write
very precisely, they are not still precise enough for a computer,
not even close.
They write things like `It can be immediately seen that ...', or
`Rewriting equation 123 in form X gives ...'.
An expert who reads the proof will indeed see it immediately,
and will manage to rewrite the equation in form X after some trying.
A computer will probably not manage. This means
that a mathematician who wants his proof checked by a computer
has to insert another 20 proofs steps everywhere in his proof,
which is too much work.

The current situation is: We know in principle how mathematical
proofs can be written down in a logical language in which they can
be checked by computers, but it is so much work that mathematicians
have no time to do this. It has been done for some big
proofs, for example the classification of finite groups,
or the Kepler conjecture, but it took many mathematicians many years

to do this.

Our project project wants to contribute to developing computer languages and
systems that make checking of mathematical proofs easier.
The first thing that we will do is develop a computer
program with reasoning abilities. In that case, when
the mathematician writes 'It can be immediately seen that ...',
maybe the computer will indeed see it after some calculation.

Another aspect that we want to study is automatic detection
of undefined values. Everyone learns in high
school that `one should not divide by zero'.
It is easy to recognize the number zero,
but what if one divides by a more complicated expression, for example
x squared - 10 times x + 26? Can it be zero for some x?
We want to develop a computer language in which conditions
like `one should not divide by zero' can be stated,
and checked automatically. This means that the mathematician
can concentrate on his calculations, and will be warned by the computer
if he accidentily tries to divide by zero, which will save time.
Division by zero is the only simplest example. There exist much more
complicated conditions that are much harder to check.

The second part of our project consists of checking a complete
mathematical proof. We chose an application from the mathematics of physics,
namely the laws of rigid body movement. Mechanics in high school
only looks at point masses. A point mass is a simplification
of reality because it does not rotate and it has no orientation.
In reality, physical objects do have orientation.
A car cannot move in every direction, it can only drive forward or
backward. In order to know which direction is forward, one must
know the orientation of the car.

We chose the rigid body laws because it is an important theory that is used in
many applications. Secondly, not much computerized proof checking has been
done in the domain of physics, and it is nice to try out a new area.
Thirdly, we suspect that there are engineering text books in which
the theory of rigid objects is used incorrectly.
These errors are not errors in mathematical calculations,
but errors in the conditions under which the equations can be applied.
These conditions are similar
to the division by zero problem mentioned above, but more complicated:
They are related to the coordinate systems in which the movements
of the object are expressed. Engineers use many coordinate systems
at the same time, and sometimes confuse them. By making the coordinate
system part of the definedness conditions, and checking them
with our system, confusion of coordinate systems becomes impossible.