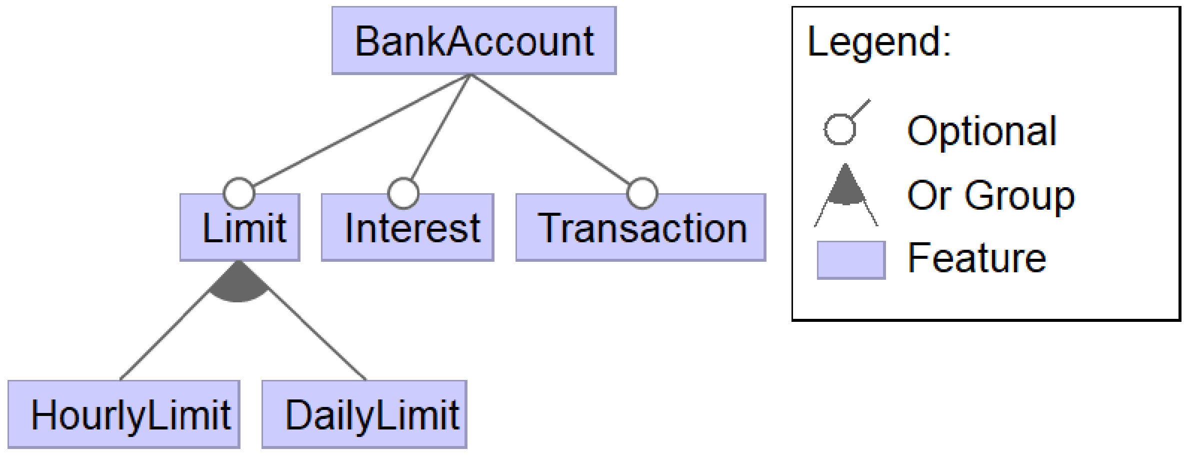
Partial Proofs to Optimize Software Product Line Verification

KeY-Symposium 2023, Bergen

Maximilian Kodetzki, Tabea Bordis, Tobias Runge, and Ina Schaefer

Motivation

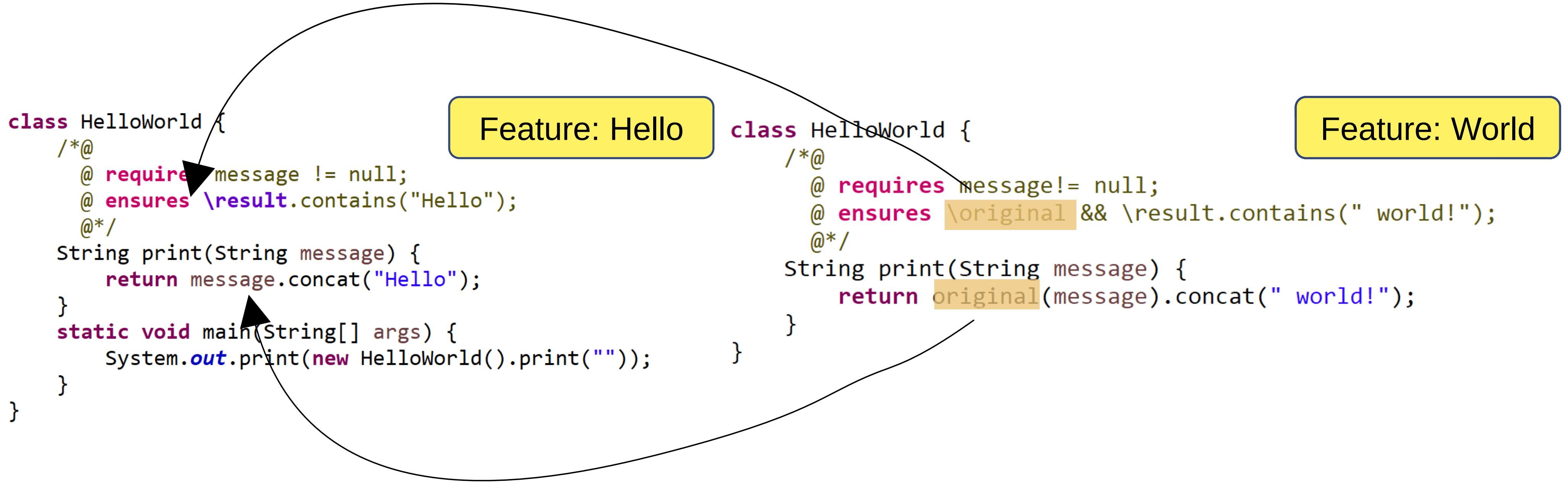
- Recurring problems
- Development is expensive: time & costs
- Development of software families



Software Product Lines

Motivation: Correctness of Software Product Lines

- Safety-critical systems require correctness
- Testing cannot guarantee correctness
- Deductive verification using feature-based specifications



Feature: Hello

```
class HelloWorld {
    /*@
      @ requires message != null;
      @ ensures \result.contains("Hello");
      @*/
    String print(String message) {
        return message.concat("Hello");
    }
    static void main(String[] args) {
        System.out.print(new HelloWorld().print(""));
    }
}
```

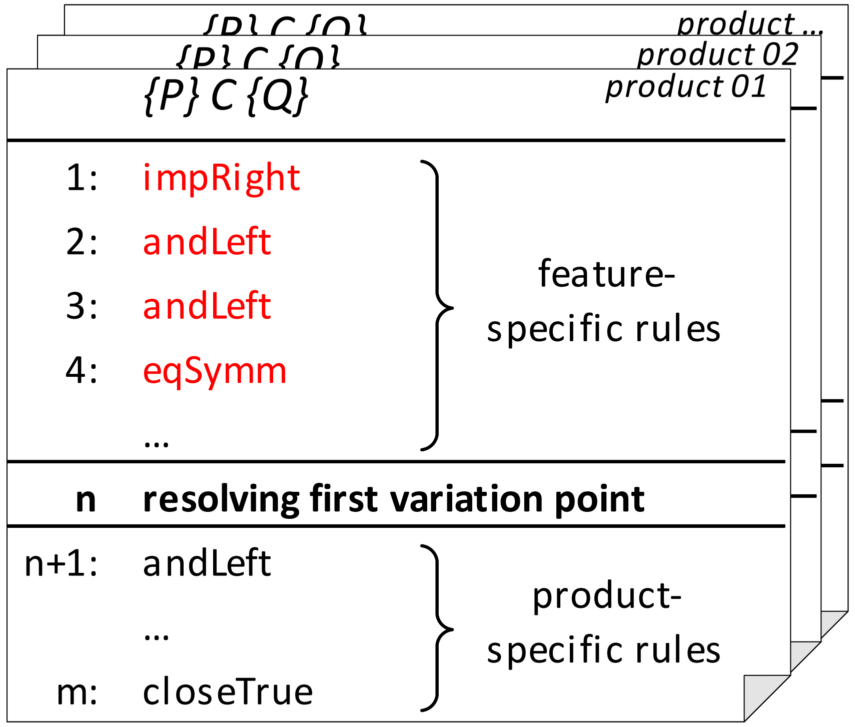
Feature: World

```
class HelloWorld {
    /*@
      @ requires message!= null;
      @ ensures \original && \result.contains(" world!");
      @*/
    String print(String message) {
        return original(message).concat(" world!");
    }
}
```

Motivation: Verification of Software Product Lines

- Product-based verification verifies every product individually

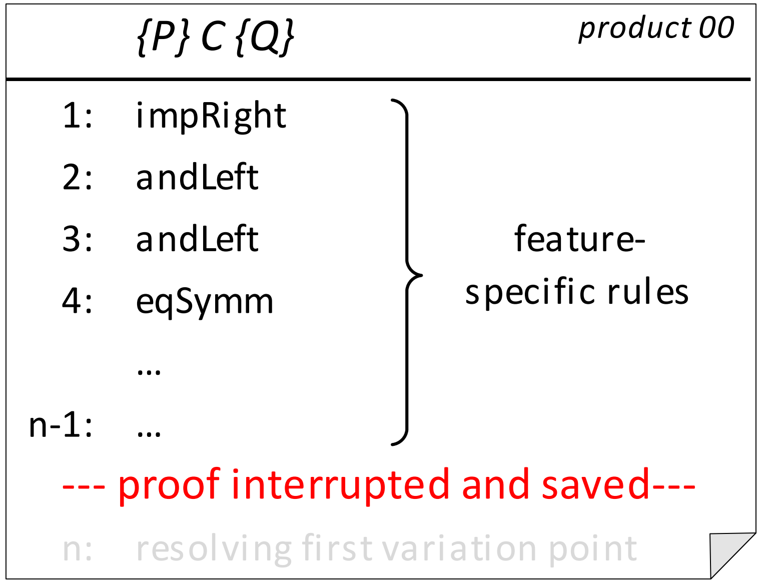
```
\original && \result.contains(" world!");
```



- Application of the same rules for each product
- Unnecessary proof parts

Partial Proofs

- Splitting up proofs in *proof start* and *proof completion*
- Based on abstract constraints[1] and partial proofs[2,3]
- Splitting index: variation points of specification



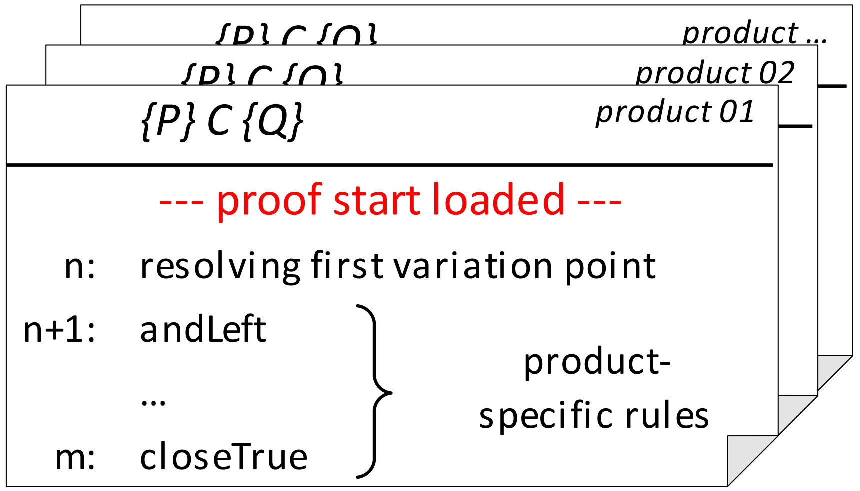
{P} C {Q} *product 00*

1: impRight
2: andLeft
3: andLeft
4: eqSymm
…
n-1: …

$\}$ feature-specific rules

--- proof interrupted and saved---

n: resolving first variation point



{P} C {Q} *product …*
{P} C {Q} *product 02*
{P} C {Q} *product 01*

--- proof start loaded ---

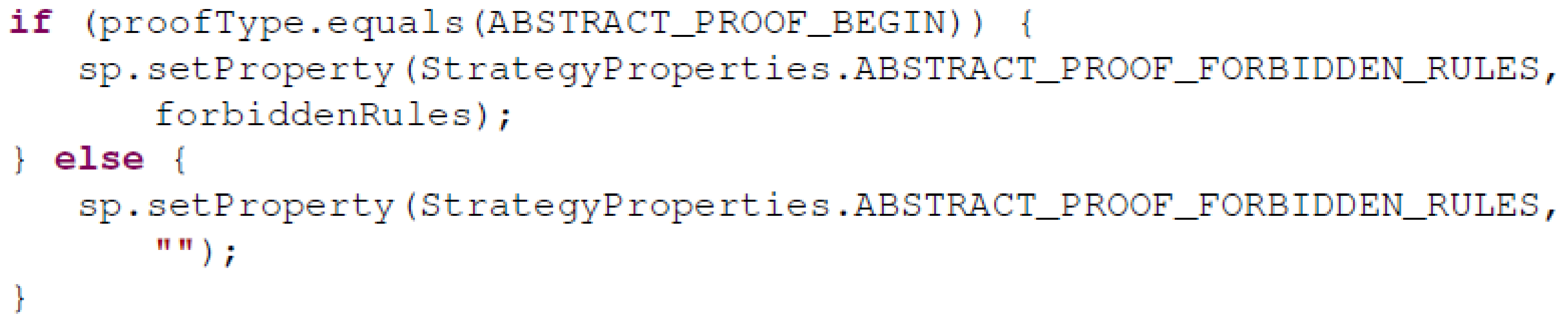
n: resolving first variation point
n+1: andLeft
…
m: closeTrue

$\}$ product-specific rules

[1] Knüppel et al. (2020): Using Abstract Contracts for Verifying Evolving Features and Their Interactions
[2] Kuiter (2020): Proof Repositories for Correct-by-Construction Software Product Lines
[3] Kuiter et al. (2022): Verification Strategies for Feature-Oriented Software Product Lines

Evaluation

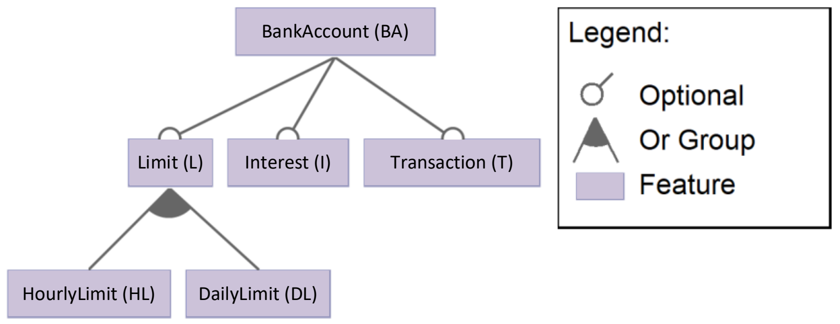
- Implementation in VarCorC
- Verification using modified version of KeY[1,2]
- Three case studies
- Parameter: Proof nodes & verification time

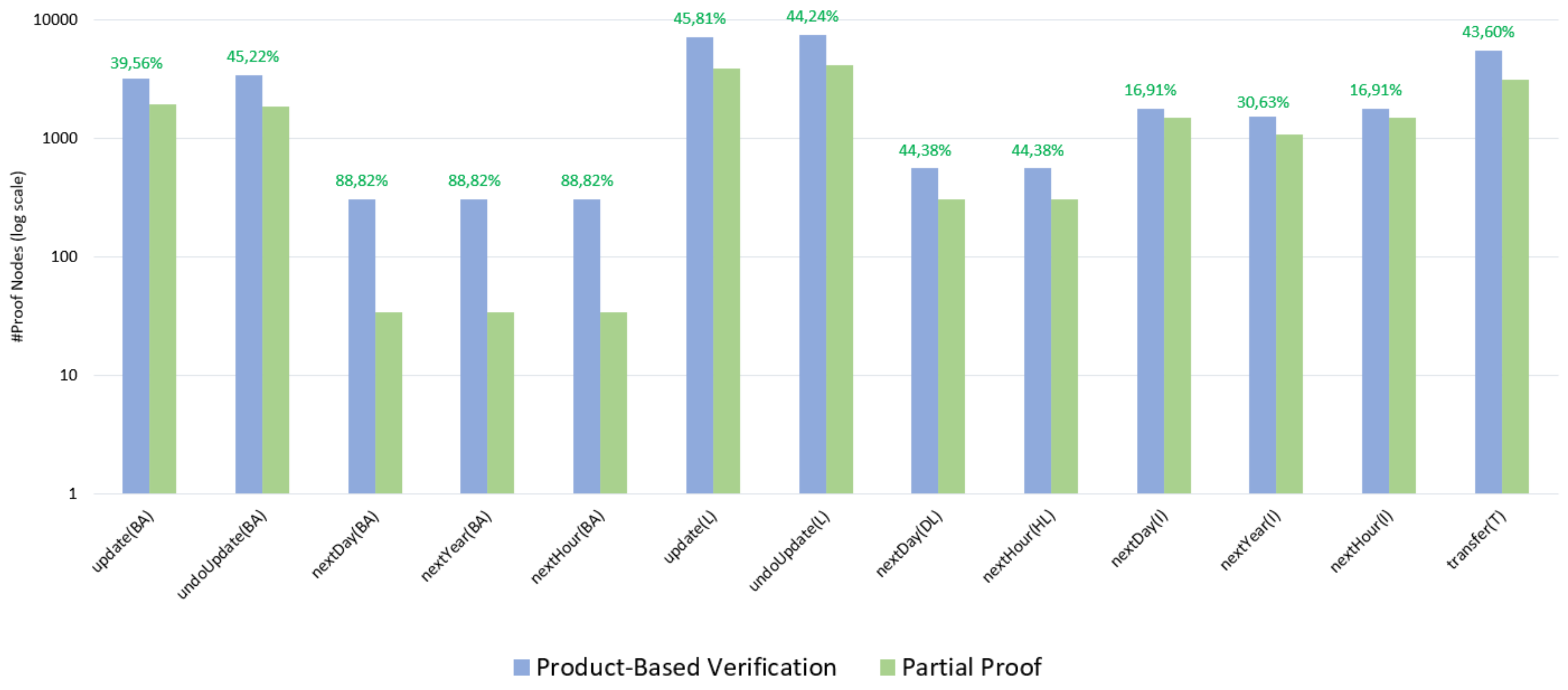


```
if (proofType.equals(ABSTRACT_PROOF_BEGIN)) {
    sp.setProperty(StrategyProperties.ABSTRACT_PROOF_FORBIDDEN_RULES,
        forbiddenRules);
} else {
    sp.setProperty(StrategyProperties.ABSTRACT_PROOF_FORBIDDEN_RULES,
        "");
}
```

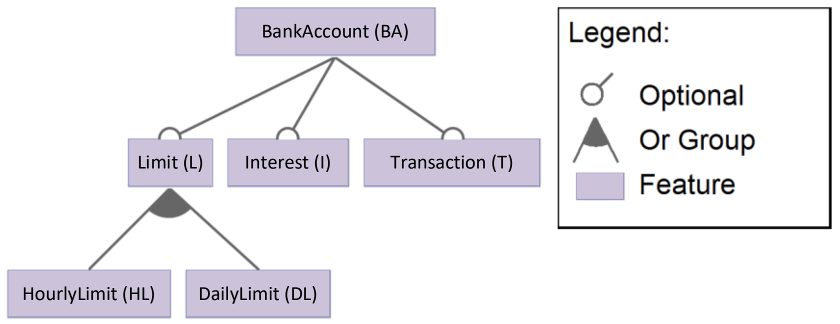
[1] Pelevina (2015): Realization and Extension of Abstract Operation Contracts for Program Logic
[2] Kuiter (2020): Proof Repositories for Correct-by-Construction Software Product Lines

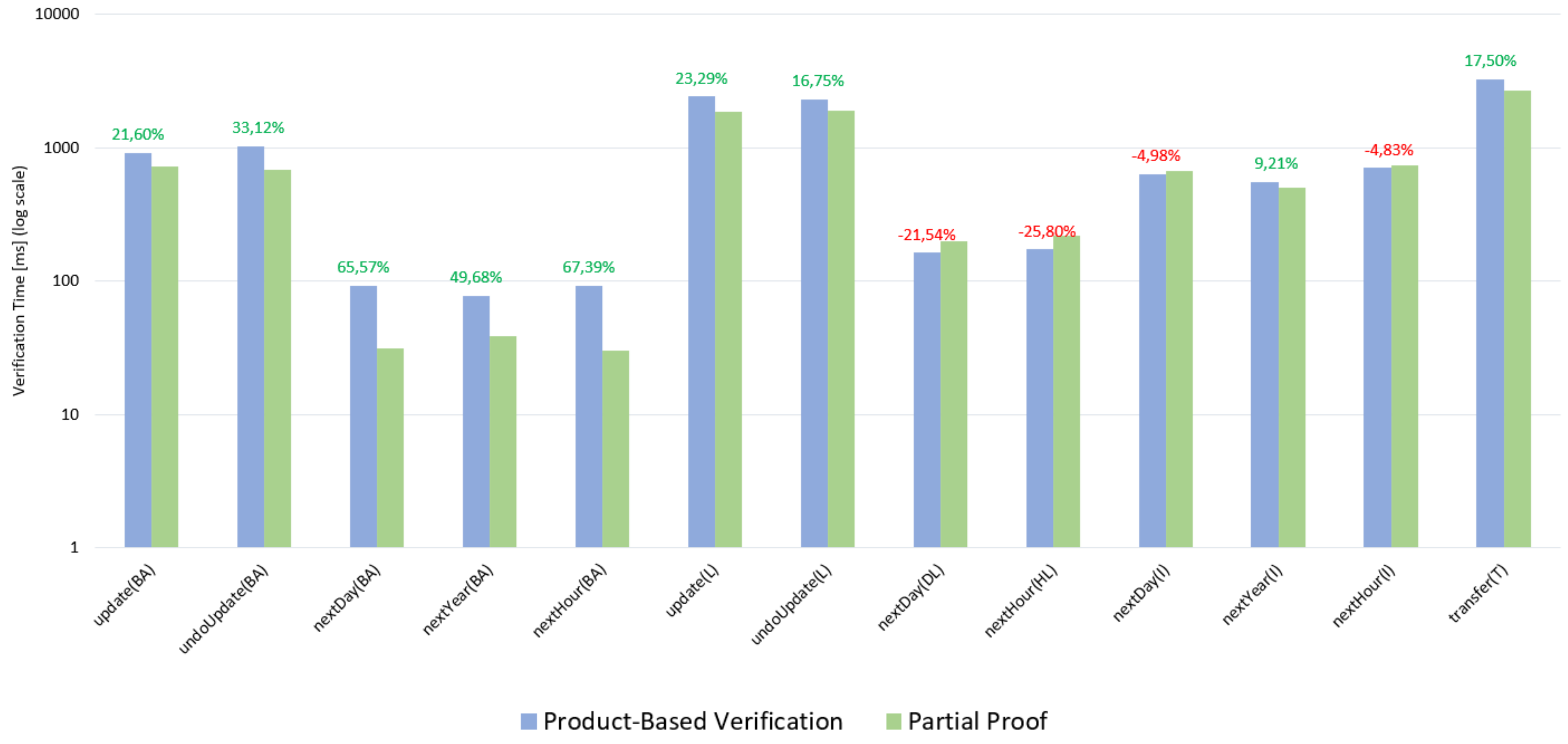
Evaluation: Proof Nodes





Evaluation: Verification Time

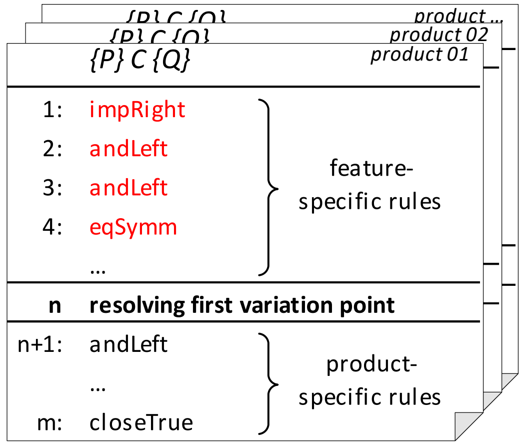




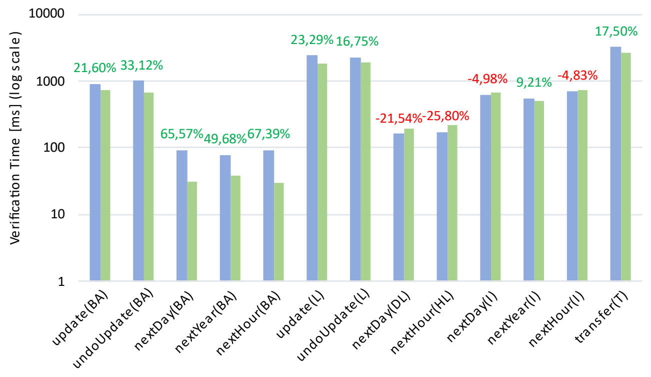
Conclusion

- Product-based verification of software product lines
- Introduction of partial proofs:
  - Proof start and proof completion
- Evaluation: Trend of improvement for large-scale SPLs

```
class HelloWorld {
    /*@
      @ requires message != null;
      @ ensures \result.contains("Hello");
      @*/
    String print(String message) {
        return message.concat("Hello");
    }
    static void main(String[] args) {
        System.out.print(new HelloWorld().print(""));
    }
}
```



{P} C {Q}
product ...
product 02
product 01
1: impRight
2: andLeft
3: andLeft
4: eqSymm
...
feature-specific rules
n resolving first variation point
n+1: andLeft
...
m: closeTrue
product-specific rules


Verification Time [ms] (log scale)
update(BA)
undoUpdate(BA)
nextDay(BA)
nextYear(BA)
nextHour(BA)
update(L)
undoUpdate(L)
nextDay(DL)
nextHour(HL)
nextDay(I)
nextYear(I)
nextHour(I)
transfer(T)