

CSE4213 Lecture Notes

Preconditions and Guards

Schneider, chapter 2

Computer Science and Software Engineering
Monash University

20050412 / Lecture 11

- 1 Objectives
- 2 Preconditions and Guards
- 3 Experiments
- 4 Examples
- 5 Summary

Objectives of this lecture

The two concepts of **preconditions** and **guards** are frequently confused, but in this lecture we will attempt to show that they are very different.

In part the confusion arises because, in general, only the concept of guard is implemented in commonly used programming languages. Curiously, in a strategy known as **defensive** programming guards are used where preconditions are needed, and this is most inappropriate.

In this lecture we will explore and contrast the concepts of **preconditions** and **guards**, in order to understand their intended purpose, and the implications of their use.

Preconditions and Guards

In the **Simple Library** case study we have instances of preconditions and guards.

Preconditions appeared in **preconditioned** substitutions.

Guards appeared in **if-then-else** constructs.

Preconditions and guards appear to be similar, but in fact they **diametrically opposed concepts**.

Experiment 1

- Animate the `CoffeeClub0` machine developed earlier in these lectures.
- Select the *FeedBank* operation a few times, and observe what happens to the state.
- Select the *RobBank* operation a few times.
- What happens when *RobBank* is selected when *piggybank* is 0?
- What happens for the following sequence: select *RobBank* when *piggybank* is 0; then select *FeedBank*?
- What conclusion do you draw from this?

Experiment 2

- Modify the *FeedBank* operation by adding a precondition $42 \leq \textit{piggybank}$, and remake the machine.
- Run the animator, and select *FeedBank* with *piggybank* equal to 0.
- What happens?
- What conclusions do you draw from this?

What are preconditions?

- Preconditions are assumptions.
- They are not conditions that are going to be checked.
- They do not prevent access to the substitutions to which they are attached.
- On the contrary, a preconditioned substitution assumes the precondition is true. If it is not then the consequence is not defined.
- Preconditions are not necessarily **correct** or **incorrect**; they may be appropriate or inappropriate.

Experiment 3

- Remove the preconditions from the FeedBank and RobBank operations.
- Remake the machine.
- Enter the **Provers** environment and run the **AutoProver**.
- Look at the undischarged proof obligations, using the **BToolProver**.
- **What conclusions do you draw?**

Experiment 4

- Modify the RobBank operation to have a guard.

```
RobBank =  
  IF amount <= piggybank  
  THEN piggybank := piggybank - amount  
  END
```

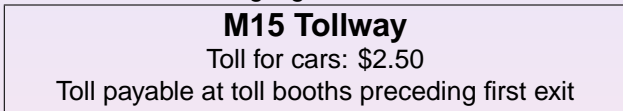
- Remake.
- Examine any undischarged proof obligations.
- What conclusions do you draw?

Conclusions from experiments

- We could conclude from the preceding experiments that it doesn't matter for the *RobBank* operation whether we use a guard or a precondition. Both lead to proof obligations that are discharged.
- But that might lead us into a false sense of security.
- For example, we might want to explore when *RobBank(1)* followed by *FeedBank(1)* is equivalent to doing nothing.
- This is true for all values of the initial value of *piggybank*, except 0.
- The exception is due to the behaviour of *RobBank(amount)*, and hence this operation should have a precondition $amount \leq piggybank$.
- It is pointless having a precondition and a guard with the same condition.

Preconditions in real life

- Consider the following sign



- If you enter the tollway you have been warned that you will need to have \$2.50 to pay the toll.
- Being able to pay the toll is a precondition to entry.
- What is the consequence of entering the tollway and not having the means to pay the toll?

Another example

- Near the entrance to a supermarket there may be a sign

Condition of entry to this store

All bags may be searched before exit from this store

- On entry to the store you have to be prepared to have any bags searched.
- This is a precondition.
- What happens if you enter the store and are not prepared to have your bag searched?

Guards in real life

- On entrance to a cinema there may be the following sign

Cinema admission prices

Adults \$13

Children \$7

Please pay at box office

- You have to buy a ticket before entering the particular theatre.
- There will be a person (guard) at each theatre to collect your ticket.
- This is a guard and you cannot enter the theatre without purchasing a ticket.

B Hive

Patrons of this club must be 18 years of age or older

If there is a guard (bouncer) at the door, then the entrance is guarded (-); otherwise it is preconditioned entrance.

Conclusions on guards and preconditions

In general, preconditions are assumptions that are necessary for the successful completion of behaviour. They are the part of the contract, between the user of an operation and the implementor of an operation, that must be met by the user of the operation.

Clearly, there is a weakest precondition – **necessary** and **sufficient** – but it might be difficult in some cases to determine that weakest precondition. In such cases a **sufficient** precondition will be used.

It is desirable that a precondition is not strengthened capriciously as was done earlier for the FeedBank operation.

Guards are used to select between different states and inputs. A guard ensures that some condition is satisfied; there is no assumption.

Summary

- **Preconditions** are assumptions
- **Guards** are gatekeepers
- Preconditions are used to define specifications, when we look for the **weakest** precondition
- Guards are used to build programs, when we look for the **strongest** behaviour enforcers