Alec Brinker - Justin Milliman
CS4710: Model-Driven Software Development, Fall 2022
Alloy Final Project - Definitely Not MTU IT (DFMTUIT) - Part 1

Abstractions

---

1. Person - Abstract sig that contains both Customers and Specialists. Person is abstract since we do not want any individuals who are neither customers nor specialists.
2. Customer - Represents customers who create tickets when they find issues
3. Specialist - Represents specialists who work on tickets when they are created
4. Ticket - Represents tickets that are created when an issue is noticed by a customer
5. Reply - Represents messages that are attached to tickets by either Customers or Specialists

Relations

---

1. Person:
    a. pReplies: The set of replies this person has created
2. Customer:
    a. cusTickets: The set of tickets that were created by this customer
3. Specialist:
    a. specTickets: The set of tickets that are being worked on by this specialist
4. Ticket:
    a. Customer: The customer that created this ticket
    b. Specialist: The specialist that is working on this ticket
    c. Replies: The set of replies to this ticket
5. Reply:
    a. Ticket: The ticket that this reply is attached to
    b. Creator: The person that created this reply
    c. Prev: The message that this message replied to

Multiplicities

---

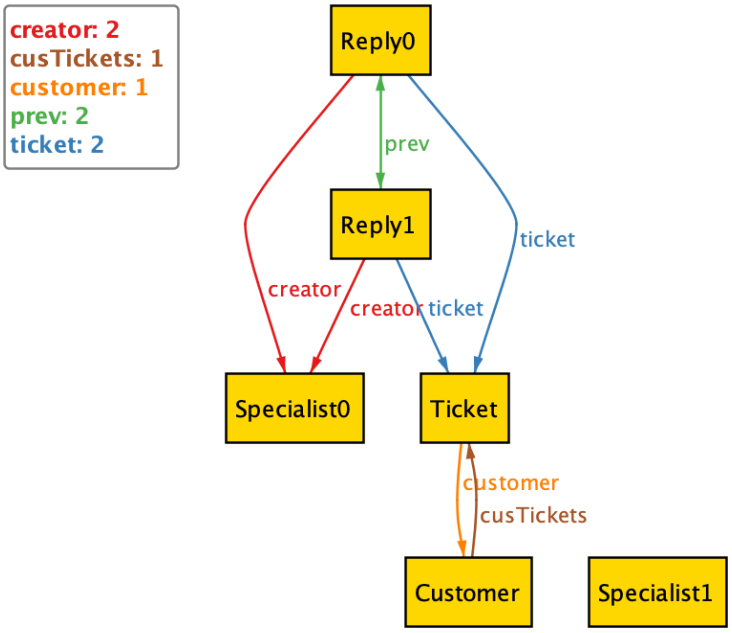
1. Person has no associated multiplicity.
    a. pReplies uses ‘set’ since a person can have sent any number of replies, including 0

2. Customer uses ‘some’ to guarantee at least one is created to help with modeling
    a. cusTickets uses ‘set’ since a customer can have any number of tickets, including 0
3. Specialist uses ‘some’ to guarantee at least one is created to help with modeling
    a. specTickets uses ‘set’ since a specialist can be working on any number of tickets at one time, including 0
4. Ticket uses ‘some’ to guarantee at least one is created to help with modeling
    a. Customer uses ‘one’ since a ticket must have been created by exactly one customer
    b. Specialist uses ‘lone’ because a ticket can be assigned to at most one specialist, but may not be assigned yet
    c. Replies uses ‘set’ because a ticket can have any number of replies, including 0
5. Reply uses ‘some’ to guarantee at least one is created to help with modeling
    a. Ticket uses ‘one’ because a reply must be associated with exactly one ticket
    b. Creator uses ‘one’ because a reply must have been sent by exactly one person
    c. Prev uses ‘lone’ because a message can be a reply to the ticket directly, or it can be a reply to another message associated with the same ticket

Versions

Version 1:

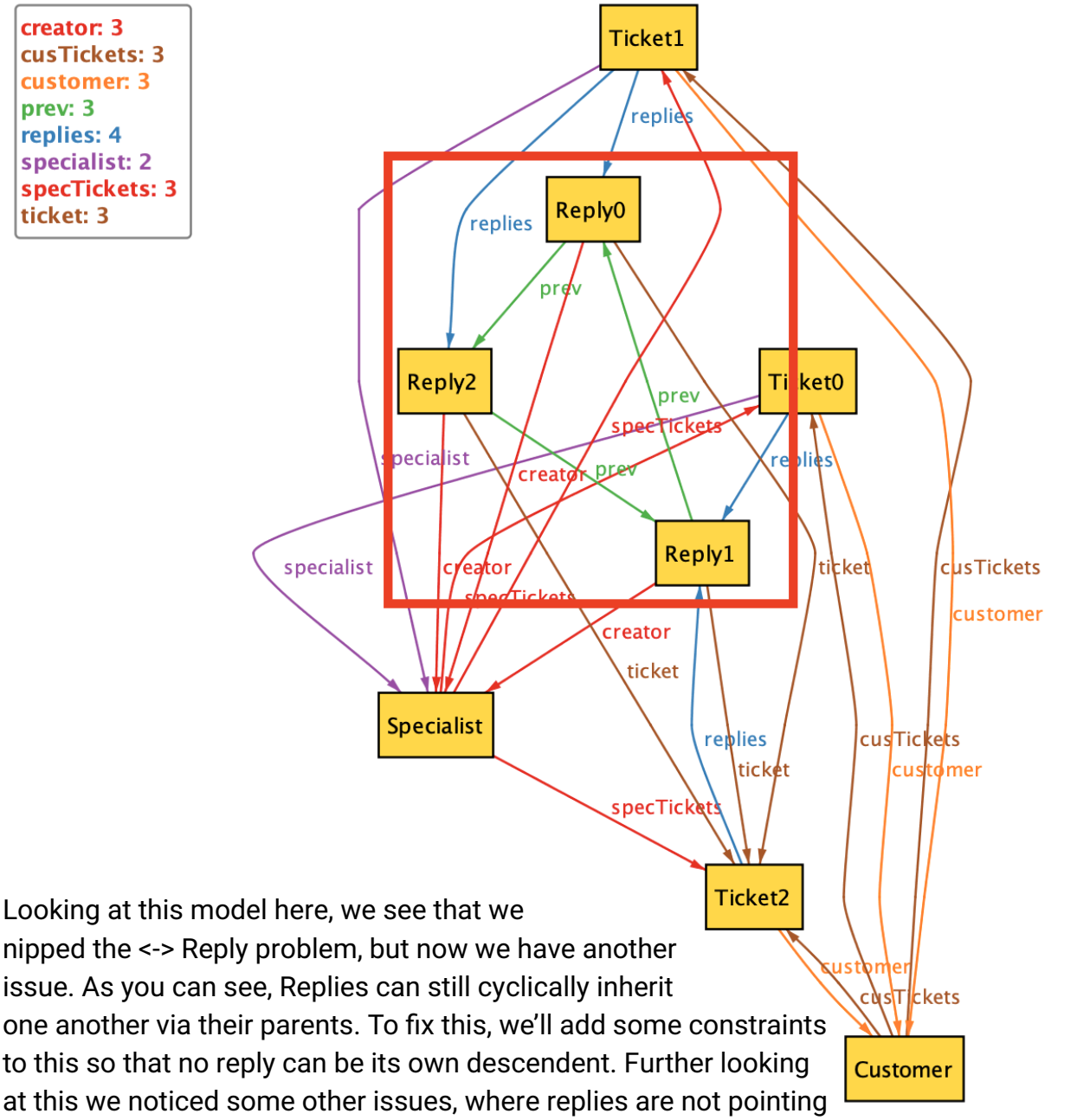
```
module DFMTUIT

abstract sig Person {}
some sig Ticket {
        customer: one Customer,
        specialist: lone Specialist,
        replies: set Reply
}
some sig Customer extends Perso
        cusTickets: set Ticket
}
some sig Specialist extends Perso
        specTickets: set Ticket
}
some sig Reply {
        ticket: one Ticket,
        creator: one Person,
        prev: lone Reply
}

fact CustomerTicketReflexive { customer in ~cusTickets }
fact SpecialistTicketReflexive { specialist in ~specTickets }
```



Looking at the above model: A customer can have an open ticket, and one and only one specialist can reply. A specialist can reply more than once, but as you'll notice, the replies are replies to each other. To fix this, we'll add additional facts that a reply cannot be a reply to a reply that replies that reply (super easy).

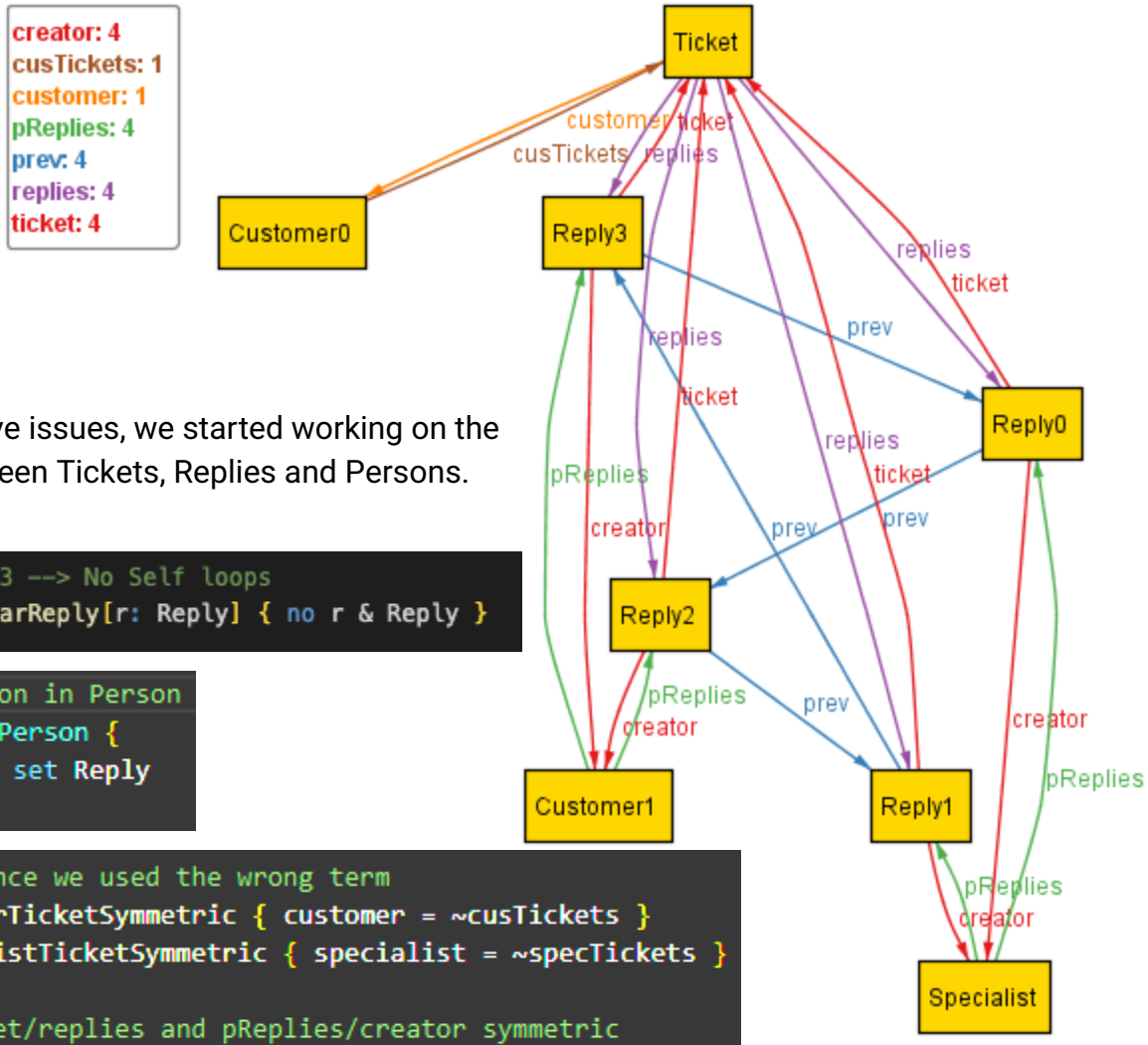
Version 2:

```
// Iteration 2 --> Replies cannot be replies to themselves, chained replies should be on the same ticket
fact ReplySameTicket { all r: Reply { r.ticket = r.prev.ticket } }
fact NoSelfReply { all r: Reply { r.prev != r } }
```



Looking at this model here, we see that we nipped the <-> Reply problem, but now we have another issue. As you can see, Replies can still cyclically inherit one another via their parents. To fix this, we'll add some constraints to this so that no reply can be its own descendent. Further looking at this we noticed some other issues, where replies are not pointing to the appropriate tickets when nested under other replies.

Version 3:



To fix the above issues, we started working on the relations between Tickets, Replies and Persons.

```
// Iteration 3 --> No Self loops
pred NoCircularReply[r: Reply] { no r & Reply }
```

```
// New relation in Person
abstract sig Person {
    pReplies: set Reply
}
```

```
// Rename since we used the wrong term
fact CustomerTicketSymmetric { customer = ~cusTickets }
fact SpecialistTicketSymmetric { specialist = ~specTickets }

// Make ticket/replies and pReplies/creator symmetric
fact ReplyTicketSymmetric { ticket = ~replies }
fact PersonReplySymmetric { pReplies = ~creator}
```

Now replies still won't inherit themselves as a parent node. Note that we still have to come back to fix the cyclical inheritance, but that the issue with Replies being the child of a reply on another ticket has been resolved. We did so by adding facts specifying that the new pReplies relation is symmetric with creator, and that the ticket relation is symmetric with replies.

Facts

---

1. fact CustomerTicketSymmetric { customer = ~cusTickets }
   This guarantees that for any ticket associated with a customer, that customer is equally associated with that ticket.

2. fact SpecialistTicketSymmetric { specialist = ~specTickets }
   This guarantees that for any ticket associated with a specialist, that specialist is equally associated with that ticket.

3. Fact ReplyTicketSymmetric { ticket = ~replies }
   This guarantees that for any ticket associated with a reply, that reply is equally associated with that ticket.

4. Fact PersonReplySymmetric { pReplies = ~creator }
   This guarantees that for any person associated with a reply, that reply is equally associated with that person.

5. fact ReplySameTicket { all r: Reply { r.ticket = r.prev.ticket } }
   This guarantees that all replies in a chain must be associated with the same ticket

6. fact NoSelfReply { all r: Reply { r.prev != r } }
   This guarantees that no message can be a reply to itself