Behavioral Clafer

Paulius Juodišius
Bogdan Petrutescu
Supervised by: Andrzej Wąsowski

July 25, 2013

Current Clafer

- Clafer is a great concise modeling language for expressing models with some structure.
- Clafer models encapsulates all possible configuration.
- With tool support we can generate and analyze instances.

Let's start with an example.

Structural Clafer example

```
1  enum RecipientLocation  = AtHome | Away
2  enum ParcelStatus = Dropped | InTransit | Delivered
4
4  abstract Parcel
5    PStatus -> ParcelStatus
6    M -> Messenger ?
7    Rec -> Recipient
9
9  abstract Recipient
10   RLoc -> RecipientLocation
11   Addr -> Address
13
13 abstract Messenger
14   P -> Parcel ?
15   Location -> Address
16   [ P => P.M=this ]
```

Structural Clafer example continued..

```
18
18  abstract Address
20
20  AlloyBook : Parcel
21    [Rec = Bob]
23
23  Mark : Messenger
25
25  Bob : Recipient
26    [Addr = Main1]
28
28  Main1 : Address
29  Main2 : Address
30  Main3 : Address
```

Let's generate some instances...

Sample instances generated using ClaferIG (1)

```
 1  Main2
 2  Main3
 3  AtHome
 4  Away
 5  Dropped
 6  InTransit
 7  Delivered
 8  AlloyBook
 9    PStatus = Delivered
10    Rec = Bob
11  Mark
12    Location = Main3
13  Bob
14    RLoc = AtHome
15    Addr = Main1
16  Main1
```

Sample instances generated using ClaferIG (2)

```
 1  Main2
 2  Main3
 3  AtHome
 4  Away
 5  Dropped
 6  InTransit
 7  Delivered
 8  AlloyBook
 9    PStatus = Delivered
10    M = Mark
11    Rec = Bob
12  Mark
13    P = AlloyBook
14    Location = Main3
15  Bob
16    RLoc = AtHome
17    Addr = Main1
18  Main1
```

Sample instances generated using ClaferIG (3)

```
 1  Main2
 2  Main3
 3  AtHome
 4  Away
 5  Dropped
 6  InTransit
 7  Delivered
 8  AlloyBook
 9    PStatus = Dropped
10    M = Mark
11    Rec = Bob
12  Mark
13    P = AlloyBook
14    Location = Main2
15  Bob
16    RLoc = Away
17    Addr = Main1
18  Main1
```

Behavioral Clafer

But that is only structure.
Maybe we can add a time dimension and create traces instead of single instances?
Then we could create tools that do something along the lines of:

1. Generate sample traces to support iterative modeling process
2. Runtime verification
3. Generate testing data for TDD.
4. ...

Let's try.

Expressing temporal properties

So we added two syntactic ways to express temporal properties:

- LTL
- Pattern Expressions based on M.Dwyer's Patterns[1].

Patterns Expressions are just sugar that is translated to LTL during preprocessing.
By default all clafers are mutable. Clafer can be made immutable using `immutable keyword`.

[1] Matthew B Dwyer, George S Avrunin, and James C Corbett. "Patterns in property specifications for finite-state verification". In: *Software Engineering, 1999. Proceedings of the 1999 International Conference on*. IEEE. 1999, pp. 411–420.

Samples of termporal properties

LTL

```
1  [ F A ]
2  [ A U B ]
3  [ G (A => F B) ]
4  [ C W D ]
```

Pattern Expressions

```
1  [ A precedes B]
2  [ C respondsTo D before E]
3  [ eventually A before E ]
4  [ eventually C after B and D ]
```

Parcel service example

Let's add temporal properties to the parcel service example.

```
4  abstract Parcel
5     PStatus -> ParcelStatus
6     M -> Messenger ?
7     Rec -> Recipient
8        [immutable]
9     [ PStatus = Dropped]
```

Parcel clafer behavioral properties (1)

Eventually parcel is delivered:

```
10    [ F (PStatus=Delivered) ]
```

Once delivered, parcel status can not change:

```
11    [ G (PStatus=Delivered => G(PStatus=Delivered)) ]
```

Parcel Dropped status has to precede InTransit status. Such event sequence is applicable before parcel reaches Delivered status:

```
12    [ PStatus=Dropped precedes PStatus=InTransit
          before (PStatus=Delivered) ]
13    // is translated to F (PStatus=Delivered) =>
          (!PStatus=Status U (PStatus=Dropped or
          PStatus=Delivered))
```

Parcel clafer behavioral properties (2)

Once in transit parcel status can not become Dropped:

```
14    [ G (PStatus=InTransit => X !(PStatus=Dropped)
          ) ]
```

Before delivery parcel has to be InTransit status:

```
15    [ PStatus=InTransit precedes PStatus=Delivered
          ]
```

During delivery parcel has to be linked with messenger:

```
16    [ G ((PStatus=InTransit && X PStatus=Delivered
          ) => M) ]
```

If Parcel is linked with messenger, its status can not be Dropped:

```
17    [ G (M => (PStatus!=Dropped)) ]
```

Parcel clafer behavioral properties (3)

Now let's add some temporal properties to the Recipient and Messenger clafers.

```
20  abstract Recipient
21    RLoc -> RecipientLocation
22    Addr -> Address
23      [immutable]
25
25  abstract Messenger
26    P -> Parcel ?
27    Location -> Address
```

Parcel clafer behavioral properties (4)

Messenger can deliver package only and only if Parcel is still not delivered, messenger is at recipient's address and recipient is at home:

```
28    [ G (P => ((P.PStatus=InTransit && Location=P.Rec
          .Addr && P.Rec.RLoc=AtHome ) <=> (P.PStatus=
          InTransit) &&  X (P.PStatus=Delivered)))]
```

This is structural property, that ensures bidirectional reference link between Parcel and Messenger clafers:

```
30    [ G (P => P.M=this) ]
```

During delivery parcel has to be linked with messenger:

```
31    [ G (P => ((some P) U P.PStatus=Delivered)) ]
```

Behavioral Clafer tool support

- We extended Clafer compiler to support new grammar and semantics.
- Using the tool behavioral Clafer model is translated to Alloy. LTL properties are encoded using embeddings presented in A.Cunha paper[2].
- Alloy analyzer tool can be used to generate traces.

[2] Alcino Cunha. "Bounded Model Checking of Temporal Formulas with Alloy". In: *CoRR* abs/1207.2746 (2012).

Behavioral Clafer Alloy output

Let's look at the Alloy output:

```
 9  open util/ordering[Time]
10  pred show {}
11  run show for 10
13
13  sig Time {loop: lone Time}
14  fact Loop {loop in last->Time}
```

A. Cunha's LTL embeddings use bounded model checking with local state idiom. Therefore we define `Time signature that will be attached to all mutable clafer relations. Time is ordered using ordering module. Bounded model checking requires a loop (lasso) relation between last Time atom and any arbitrary Time atom.`

Behavioral Clafer Alloy output

Now let's look at the Parcel clafer signature:

```
37 abstract sig c8_Parcel
38 { r_c9_PStatus : c9_PStatus -> Time
39 , r_c19_M : c19_M -> Time
40 , r_c29_Rec : one c29_Rec }
```

Each mutable subclafer is a product of field relation and Time set.

Note, that Parcel's `Rec is immutable, so its relation is not a product with Time.`

Liveness

Liveness property embedding in A. Cunha paper:

$$[F\varphi]_t \equiv \text{ some } t' : t. * (\text{next} + \text{loop}) \mid [\varphi]_{t'}$$

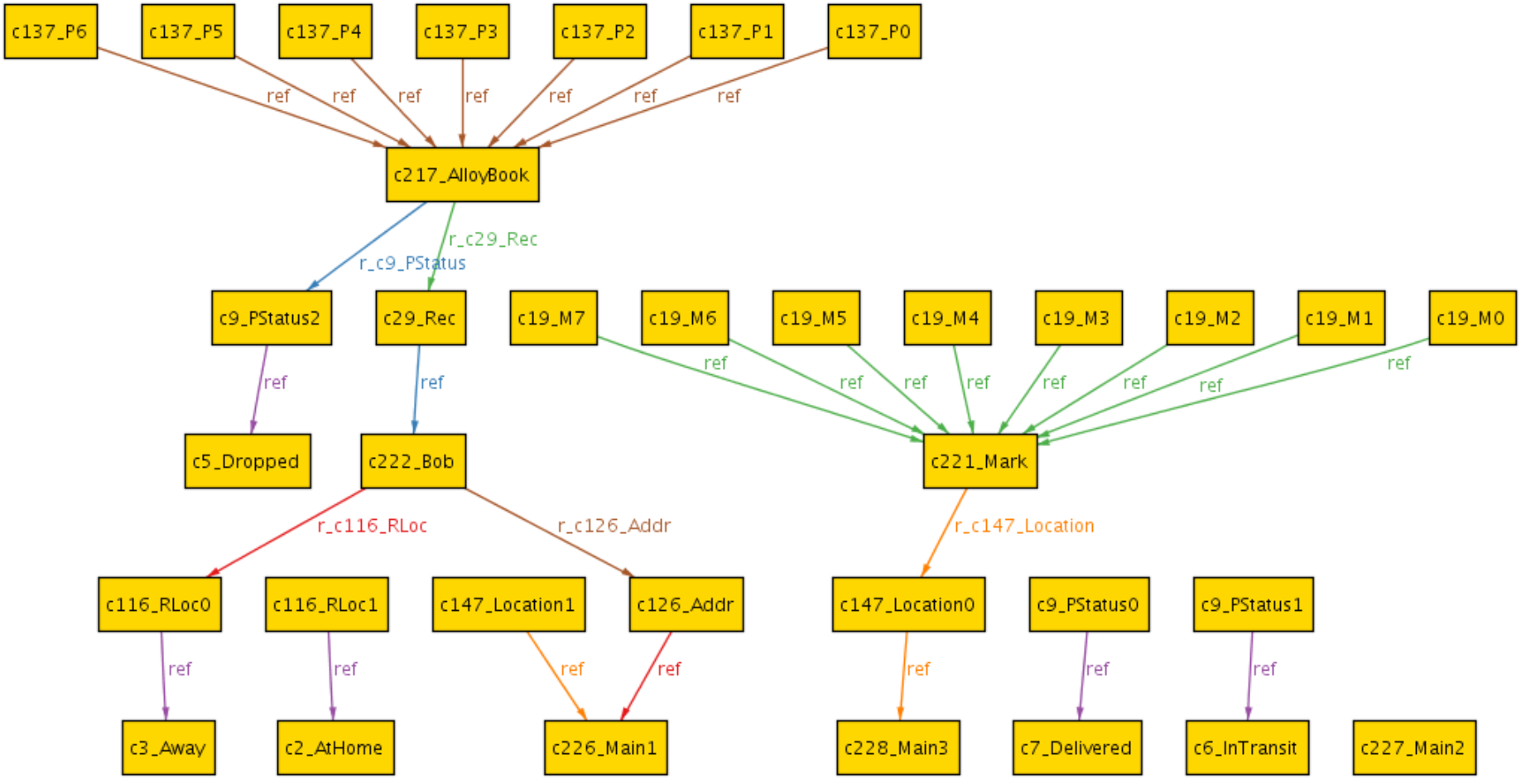
Liveness property in Clafer:

```
10    [ F (PStatus=Delivered) ]
```

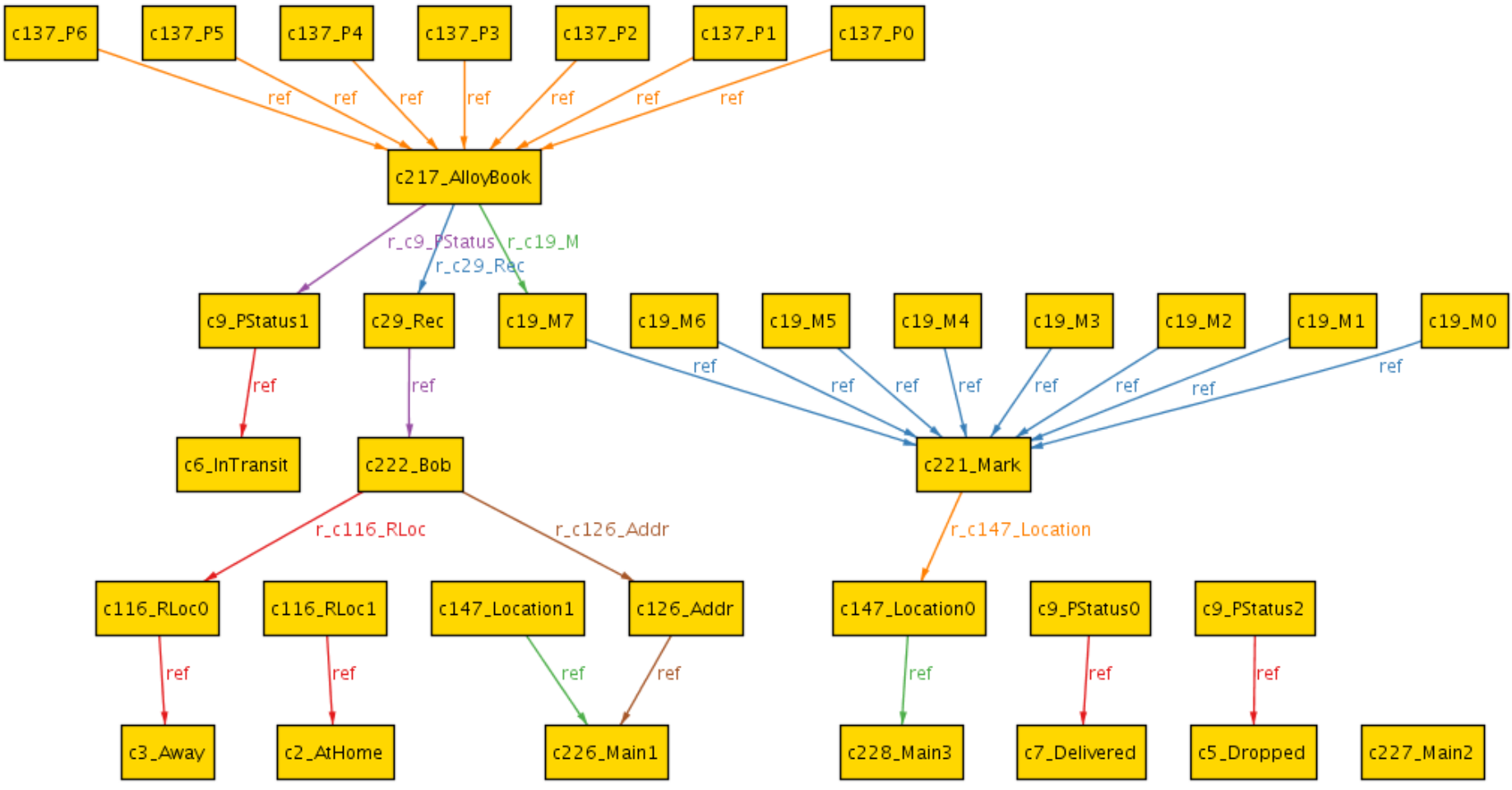
Liveness property in Alloy:

```
46    all t : (Time <: first) | (some t':t.*(Time <:
          next + loop) | (this.(@r_c9_PStatus.t'.@ref))
           = c7_Delivered)
```

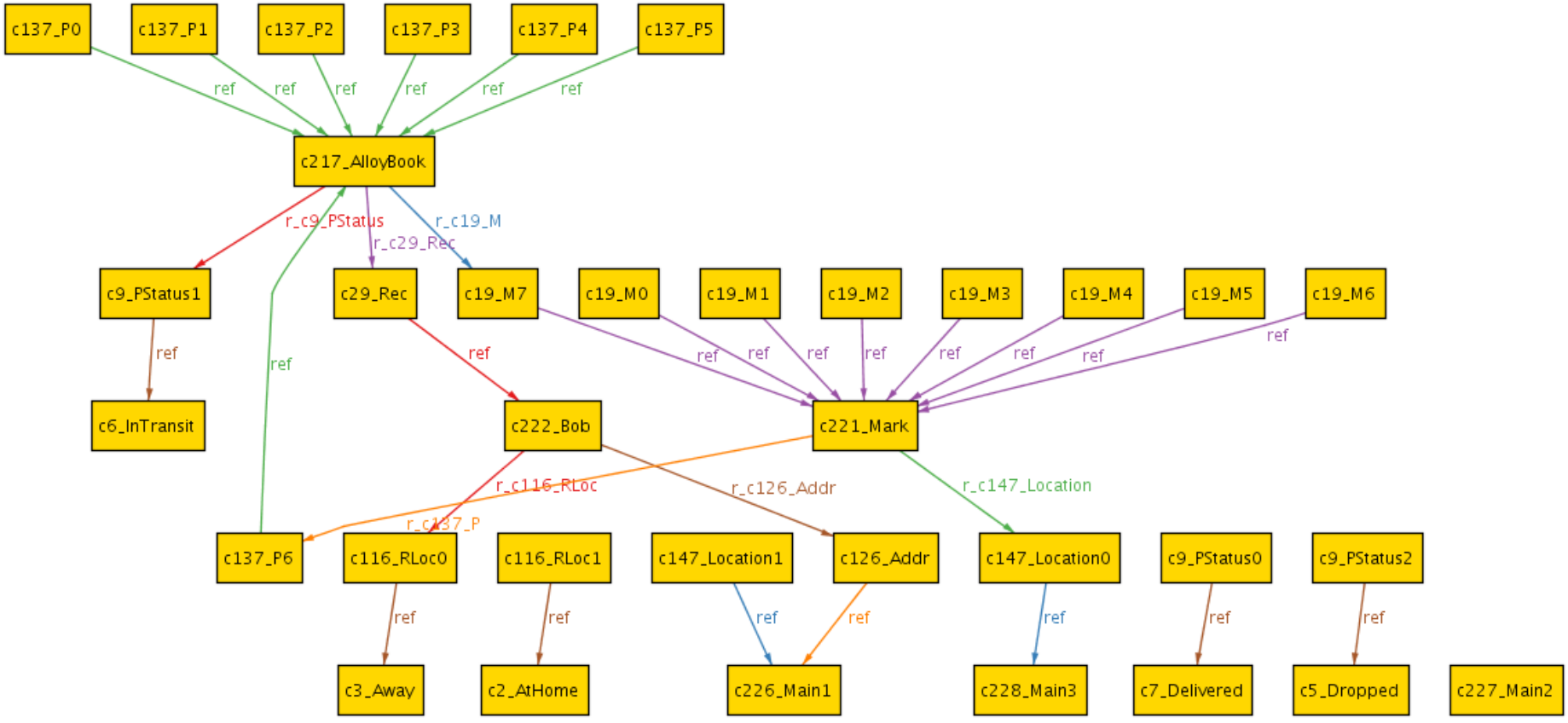
Parcel service trace (Time=0)



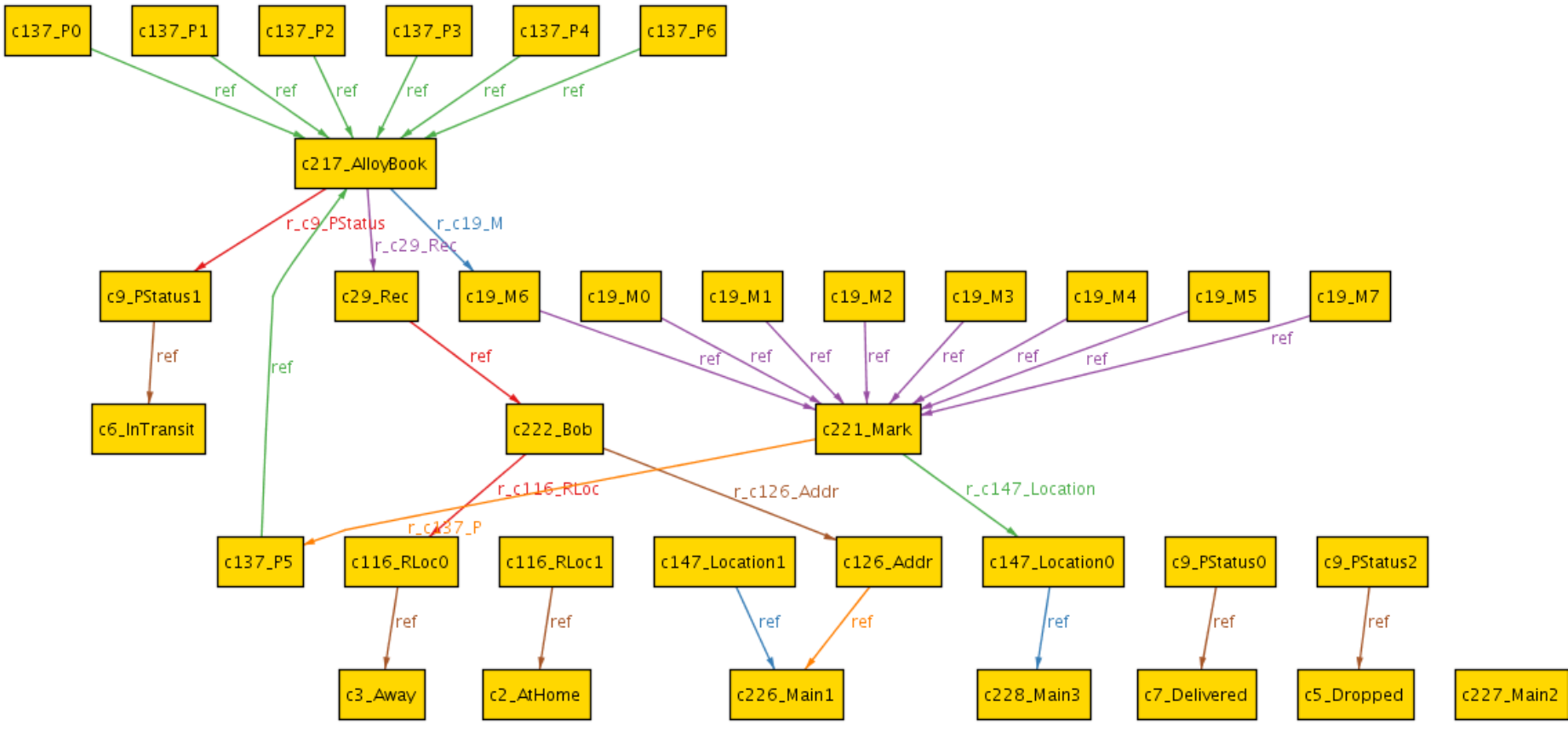
Parcel service trace (Time=1)



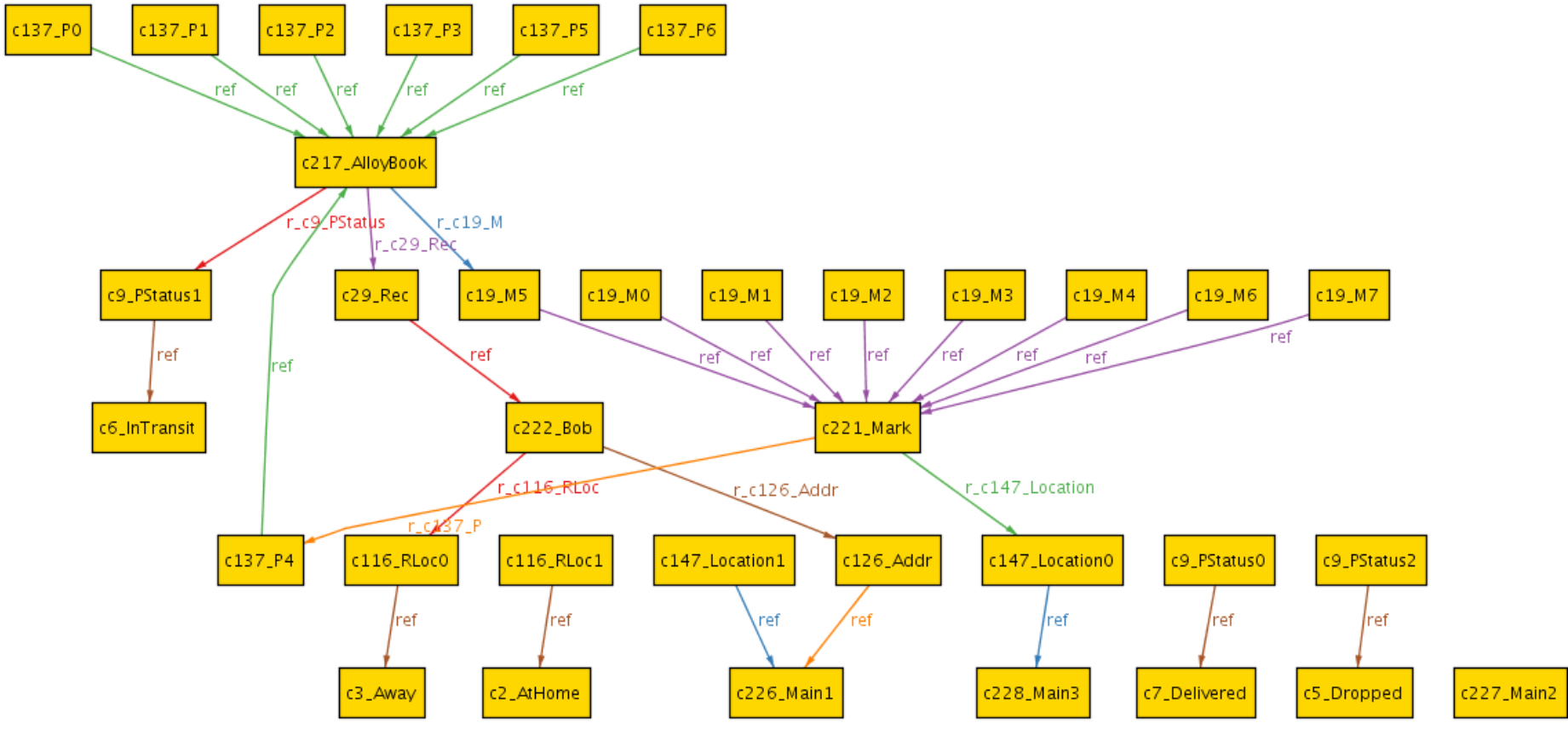
Parcel service trace (Time=2)



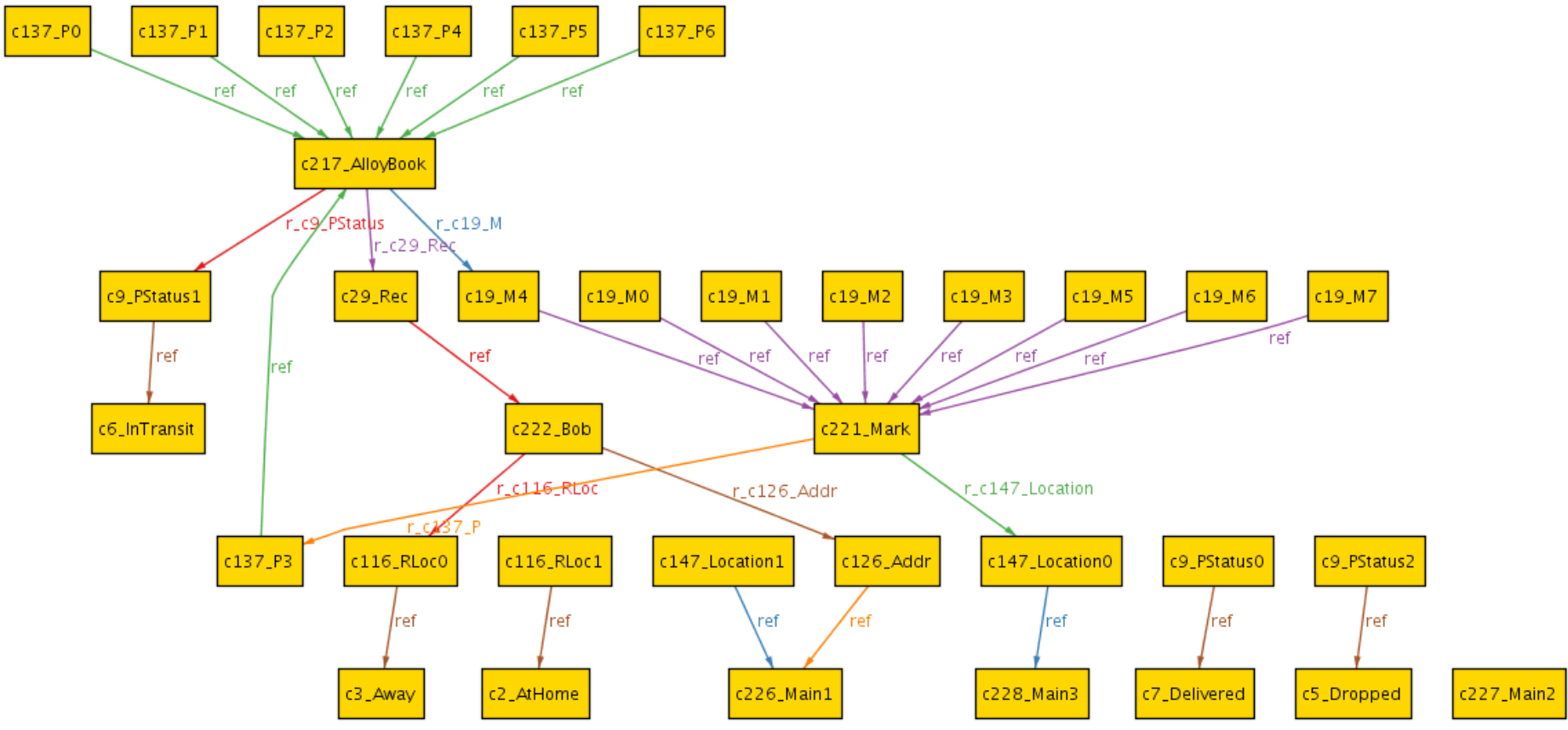
Parcel service trace (Time=3)



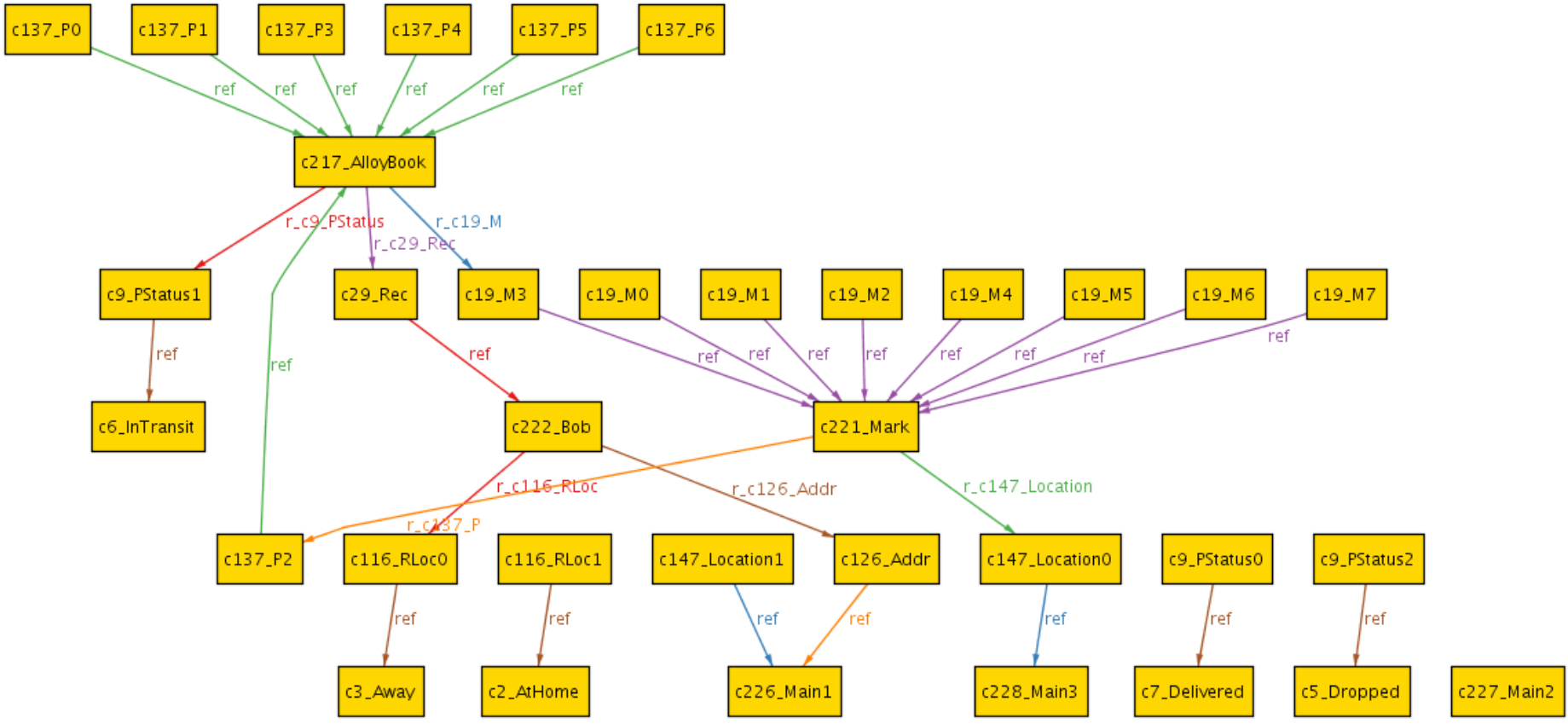
Parcel service trace (Time=4)



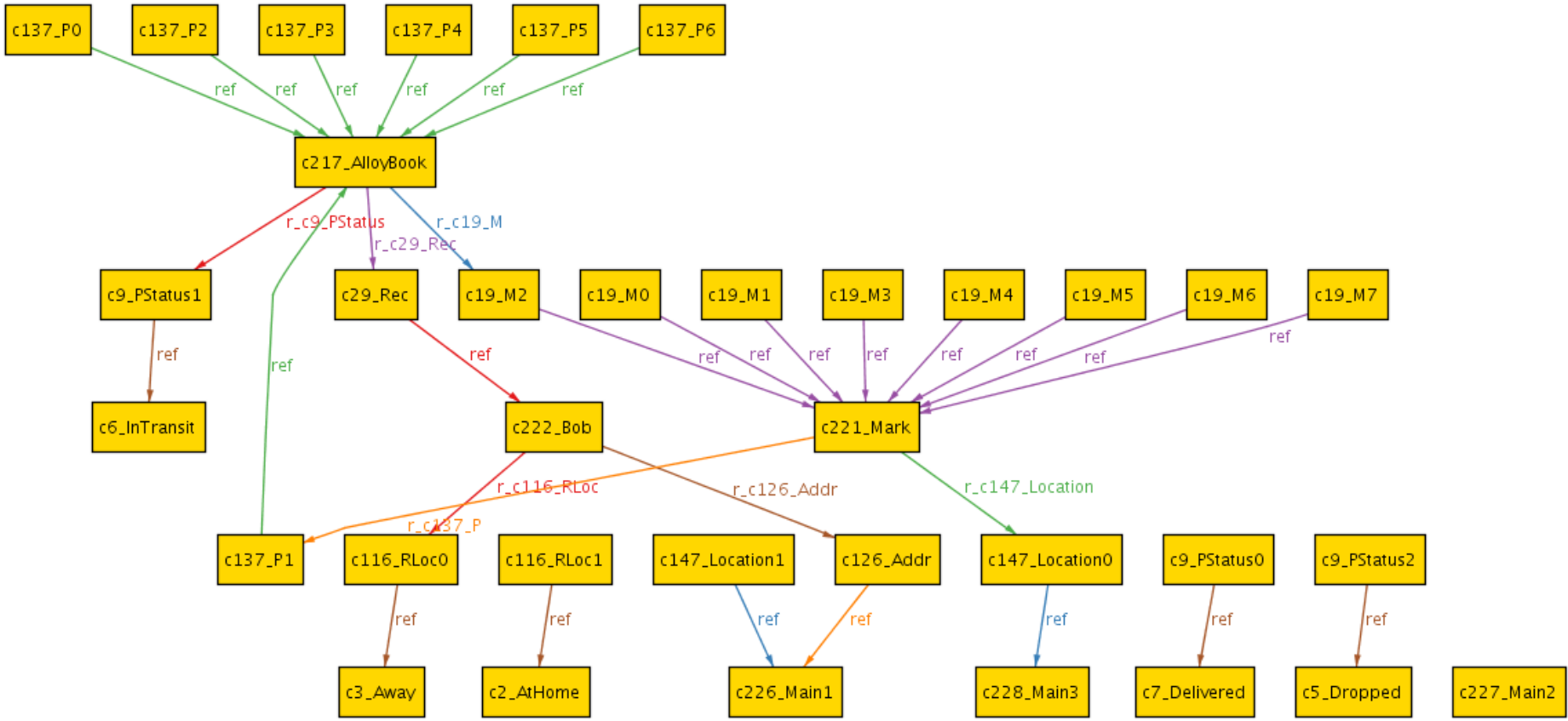
Parcel service trace (Time=5)



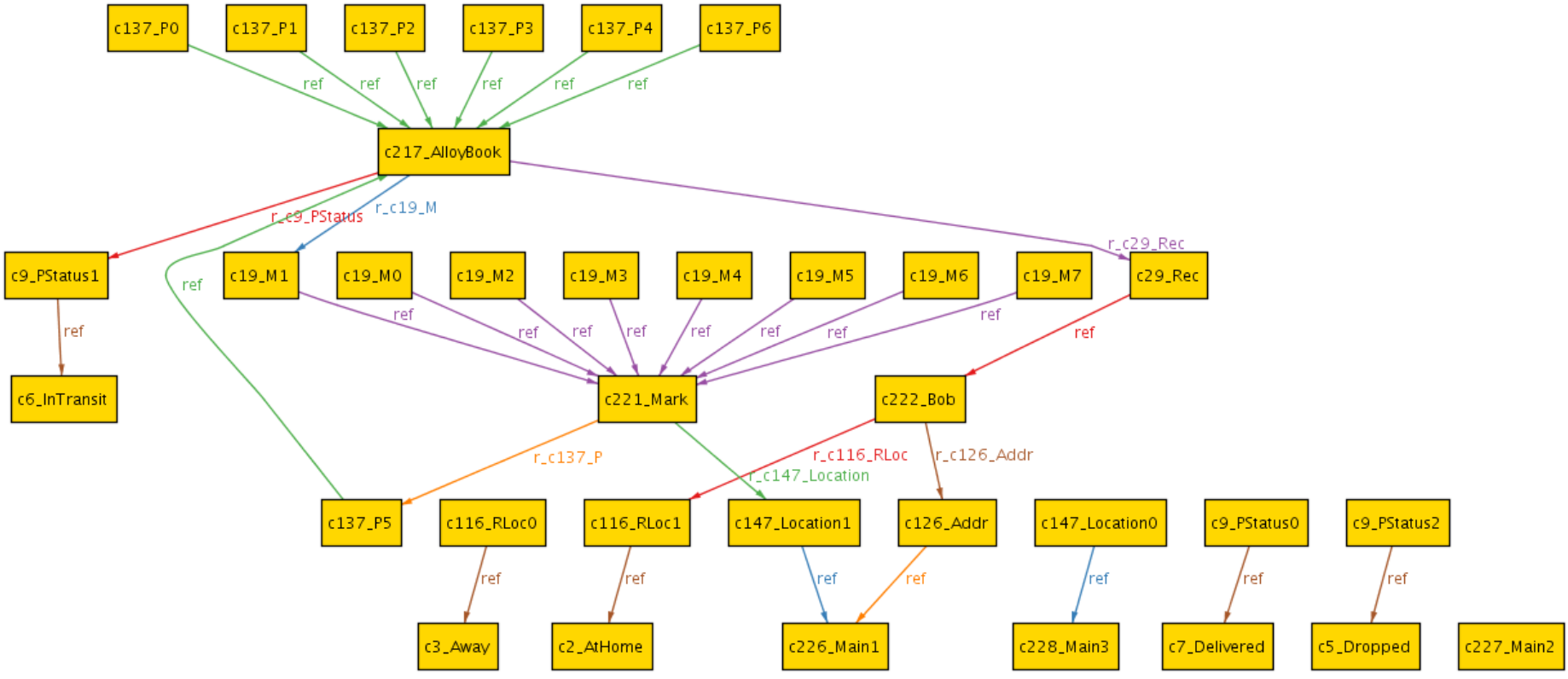
Parcel service trace (Time=6)



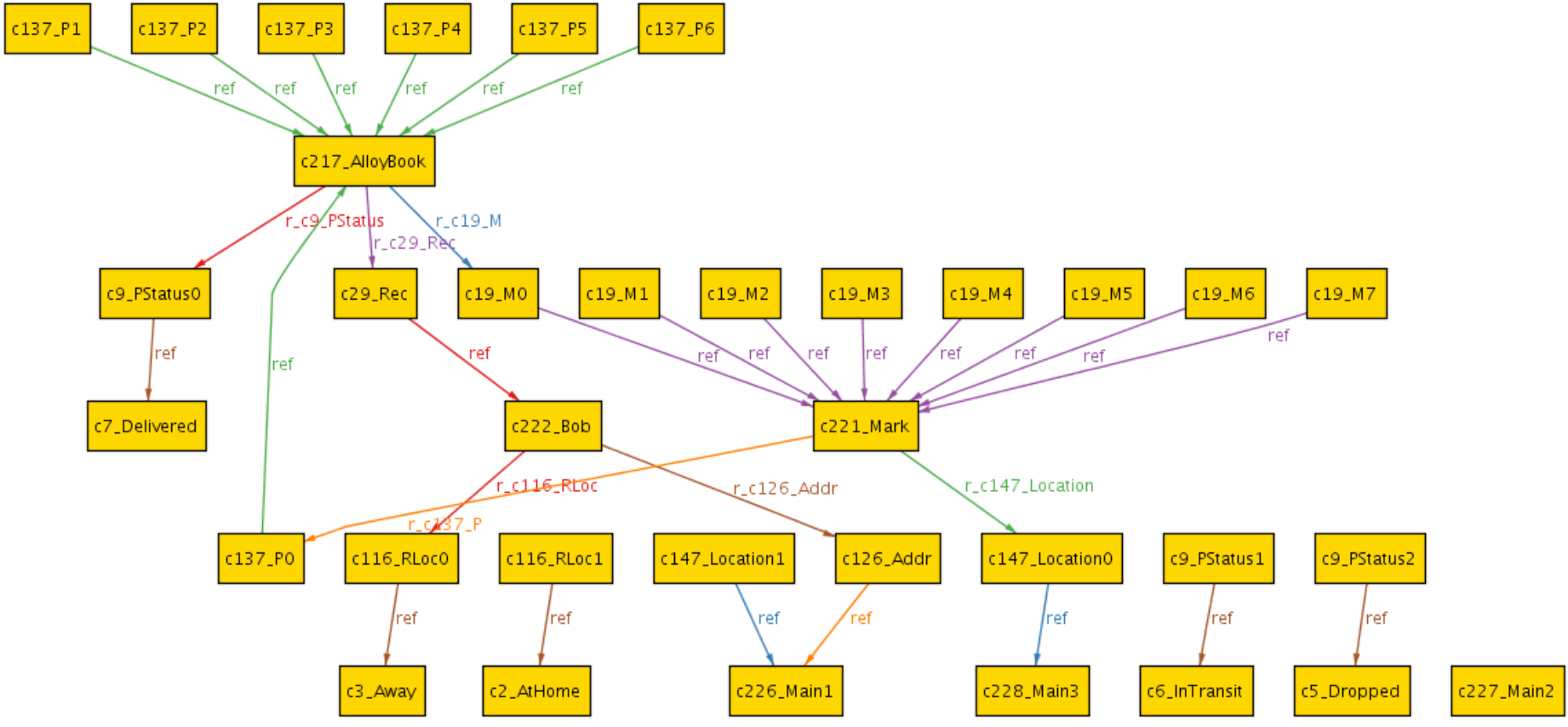
Parcel service trace (Time=7)



Parcel service trace (Time=8)



Parcel service trace (Time=9)



Open issues/Future work

- Calculate scope - right now it is just a parameter (–fs=scope).
- For reference clafers add Time product to `rel field instead of appending to field declarations in parent signature. It will prevent generation of unnecessary instances.`
- `Create more models.`