CS:5810
Formal Methods in Software Engineering

Dynamic Models in Alloy

Overview

- Basics of dynamic models
  - Modeling a system's states and state transitions
  - Modeling operations causing transitions
- Simple example of operations

Static Models

- So far we’ve used Alloy to define the allowable values of state components
  - values of sets
  - values of relations
- A model instance is a set of state component values that
  - Satisfies the constraints defined by multiplicities, fact, “realism” conditions, …

Static Models

```
Person = {Matt, Sue}
Man = {Matt}
Woman = {Sue}
Married = {}
spouse = {}
children = {}
siblings = {}
```

```
Person = {Matt, Sue}
Man = {Matt}
Woman = {Sue}
Married = {Matt, Sue}
spouse = {(Matt,Sue), (Sue,Matt)}
children = {}
siblings = {}
```

```
Person = {Matt, Sue, Sean}
Man = {Matt, Sean}
Woman = {Sue}
Married = {Matt, Sue}
spouse = {(Matt,Sue), (Sue,Matt)}
children = {(Matt,Sean), (Sue,Sean)}
siblings = {}
```

Dynamic Models

- Static models allow us to describe the legal states of a dynamic system

- We also want to be able to describe the legal transitions between states

  E.g.

  - To get married one must be alive and not currently married
  - One must be alive to be able to die
  - A person becomes someone's child after birth

Example

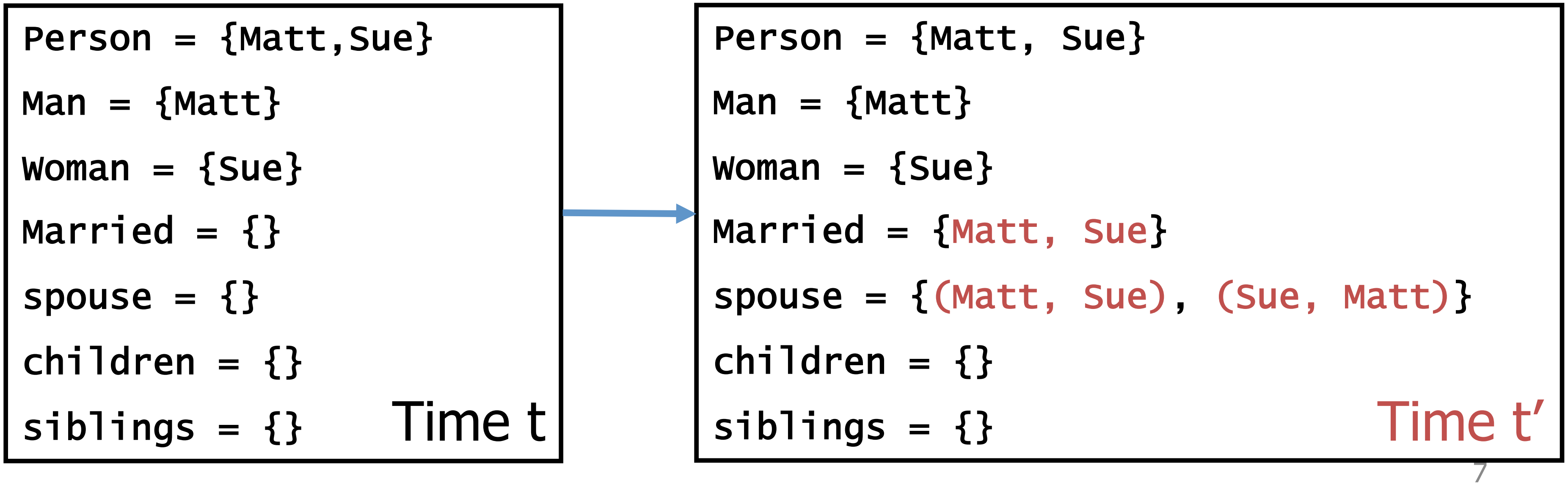
Family Model

```
abstract sig Person {
      children: set Person,
      siblings: set Person
}

sig Man, Woman extends Person {}

sig Married in Person {
      spouse: one Married
}
```

Transitions

- **Two people get married**
  - At time t, spouse = {}
  - At time t’, spouse = {(Matt, Sue), (Sue,Matt)}

⇒ We add the notion of time in the relation spouse



```
Person = {Matt,Sue}
Man = {Matt}
Woman = {Sue}
Married = {}
spouse = {}
children = {}
siblings = {}
```

Time t

```
Person = {Matt, Sue}
Man = {Matt}
Woman = {Sue}
Married = {Matt, Sue}
spouse = {(Matt, Sue), (Sue, Matt)}
children = {}
siblings = {}
```

Time t’

Modeling State Transitions

- Alloy does not have an embedded notions of state transition
- However, there are several ways to model dynamic aspects of a system
- A general and relative simple one is to:
  - introduce a `Time` signature expressing time and
  - add a time component to each relation that changes over time

Summarizing

Family Model

```
abstract sig Person {
     children: set Person,
     siblings: set Person
}
sig Man, Woman extends Person {}

sig Married in Person {
     spouse: one Married
}
```

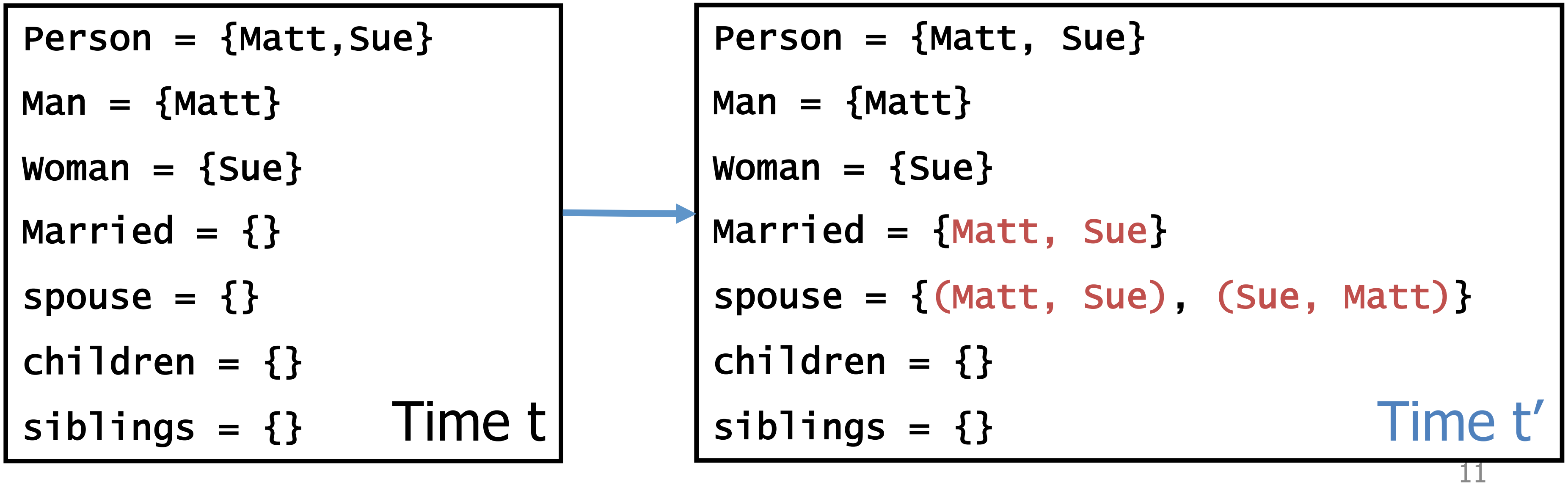
Example

Family Model

```
sig Time {}

abstract sig Person {
     children: Person set -> Time,
     siblings: Person set -> Time
}
sig Man, Woman extends Person {}

sig Married in Person {
     spouse: Married one -> Time
}
```

Transitions

- **Two people get married**
  - At time t, Married = {}
  - At time t’, Married = {Matt, Sue}
  - Actually, we can’t have a time-dependent signature such as Married because signatures are not time dependent.



```
Person = {Matt,Sue}
Man = {Matt}
Woman = {Sue}
Married = {}
spouse = {}
children = {}
siblings = {}
```

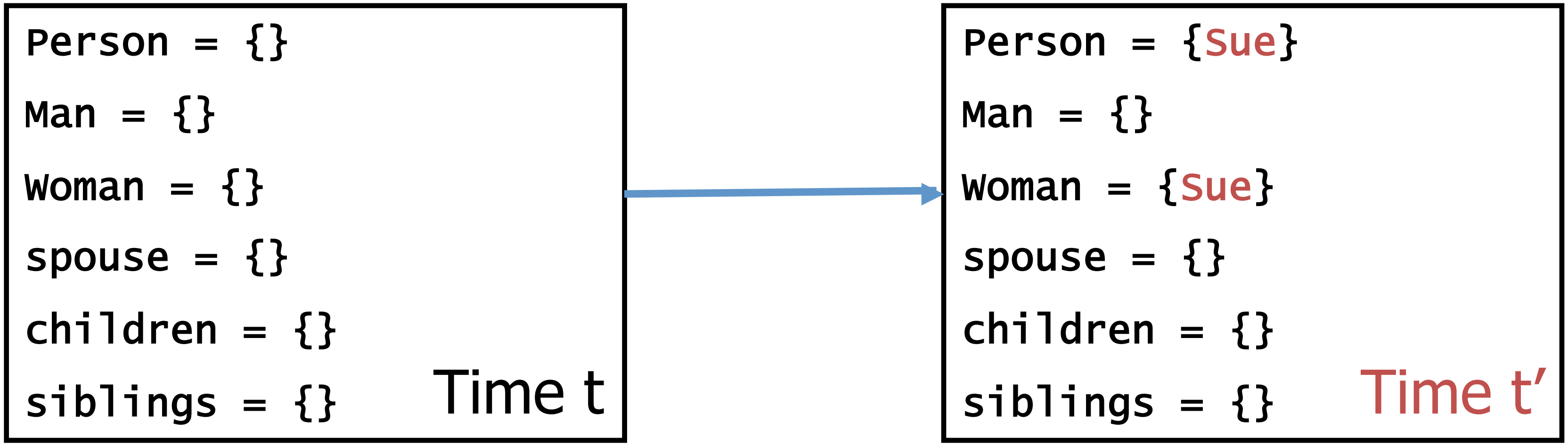
Time t

```
Person = {Matt, Sue}
Man = {Matt}
Woman = {Sue}
Married = {Matt, Sue}
spouse = {(Matt, Sue), (Sue, Matt)}
children = {}
siblings = {}
```

Time t’

Transitions

- **A person is born**
  - At time t, Person = {}
  - At time t’, Person = {Sue}
  - We cannot add the notion being born to the signature Person because signatures are not time dependent.



```
Person = {}
Man = {}
Woman = {}
spouse = {}
children = {}
siblings = {}
```

Time t

```
Person = {Sue}
Man = {}
Woman = {Sue}
spouse = {}
children = {}
siblings = {}
```

Time t’

Signatures are static

Family Model

```
abstract sig Person {
     children: Person set -> Time,
     siblings: Person set -> Time,
     spouse: Person lone -> Time
}
sig Man, Woman extends Person {}
```

```
~~sig Married in Person {~~
     ~~spouse: Married one -> Time~~
~~}~~
```

Signatures are static

Family Model

```
abstract sig Person {
      children: Person set -> Time,
      siblings: Person set -> Time,
      spouse: Person lone -> Time
}
sig Man, Woman extends Person {}
```

We want to add this relation, but where?

```
          alive: Person set -> Time
```

Signatures are static

Family Model

```
abstract sig Person {
     children: Person set -> Time,
     siblings: Person set -> Time,
     spouse: Person lone -> Time
     alive: set Time

}

sig Man, Woman extends Person {}
```

Revising constraints

Family Model

```
abstract sig Person {
     children: Person set -> Time,
     siblings: Person set -> Time,
     spouse: Person lone -> Time,
     alive: set Time
     parents: Person set -> Time
}

sig Man, Woman extends Person {}

~~fun parents[] : Person->Person {~children}~~
fact parentsDef {
   all t: Time | parents.t = ~(children.t)
}
```

Revising constraints

```
-- Time-dependent parents relation

fact parentsDef {
  all t: Time | parents.t = ~(children.t)
}

-- Two persons are blood relatives iff
-- they have a common ancestor
pred BloodRelatives [p, q: Person, t: Time]
{
  some p.*(parents.t) & q.*(parents.t)
}
```

Revising static constraints

```
fact static {
  -- People cannot be their own ancestors
  all t: Time | no p: Person |
    p in p.^(parents.t)

  -- No one can have more than one father
  -- or mother
  all t: Time | all p: Person |
   lone (p.parents.t & Man)
     and
   lone (p.parents.t & Woman)

  ...
```

Revising static constraints

```
...

-- A person p's siblings are those people, other
-- than p, with the same parents as p
all t: Time | all p: Person |
  some p.parents.t implies
   p.siblings.t =
        ({q: Person | p.parents.t = q.parents.t} -
  p )
  else no p.siblings.t

-- Each married man (woman) has a wife (husband)
all t: Time | all p: Person |
  let s = p.spouse.t |
    (p in Man implies s in Woman) and
    (p in Woman implies s in Man)
```

Revising static constraints

```
...
-- A spouse can't be a sibling
all t: Time | no p: Person |
  some p.spouse.t and
  p.spouse.t in p.siblings.t

-- People can't be married to a blood
-- relative
  all t: Time | no p: Person |
    let s = p.spouse.t |
      some s and
    BloodRelatives [p, s, t]

  ...
```

Revising static constraints

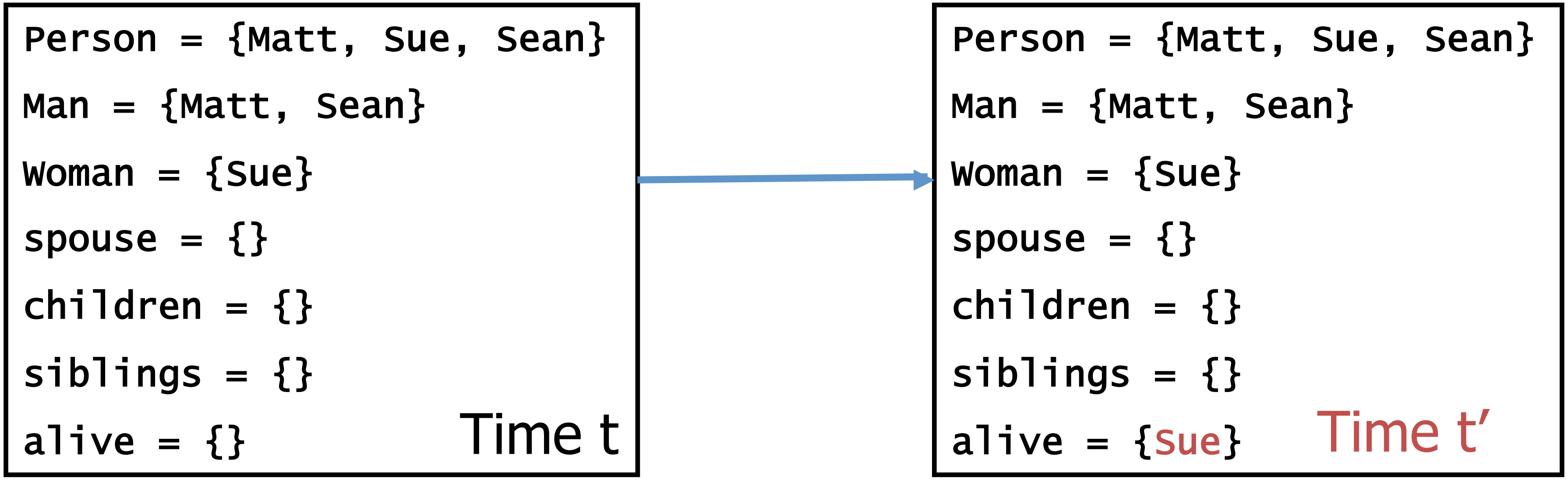
```
…
-- a person can't have children with
-- a blood relative
all t: Time | all p, q: Person |
  (some (p.children.t & q.children.t) and
   p != q)
  implies
  not BloodRelatives [p, q, t]

-- the spouse relation is symmetric
all t: Time |
  spouse.t = ~(spouse.t)
}
```

Exercises

- Load `family-6.als`
- Execute it
- Analyze the model
- Look at the generated instance
- Does it look correct?
- What, if anything, would you change about it?

Transitions

- **A person is born**
  - Add to alive relation
  - NB: No requirement that a person have parents



```
Person = {Matt, Sue, Sean}
Man = {Matt, Sean}
Woman = {Sue}
spouse = {}
children = {}
siblings = {}
alive = {}
```

Time t

```
Person = {Matt, Sue, Sean}
Man = {Matt, Sean}
Woman = {Sue}
spouse = {}
children = {}
siblings = {}
alive = {Sue}
```

Time t’

Transitions

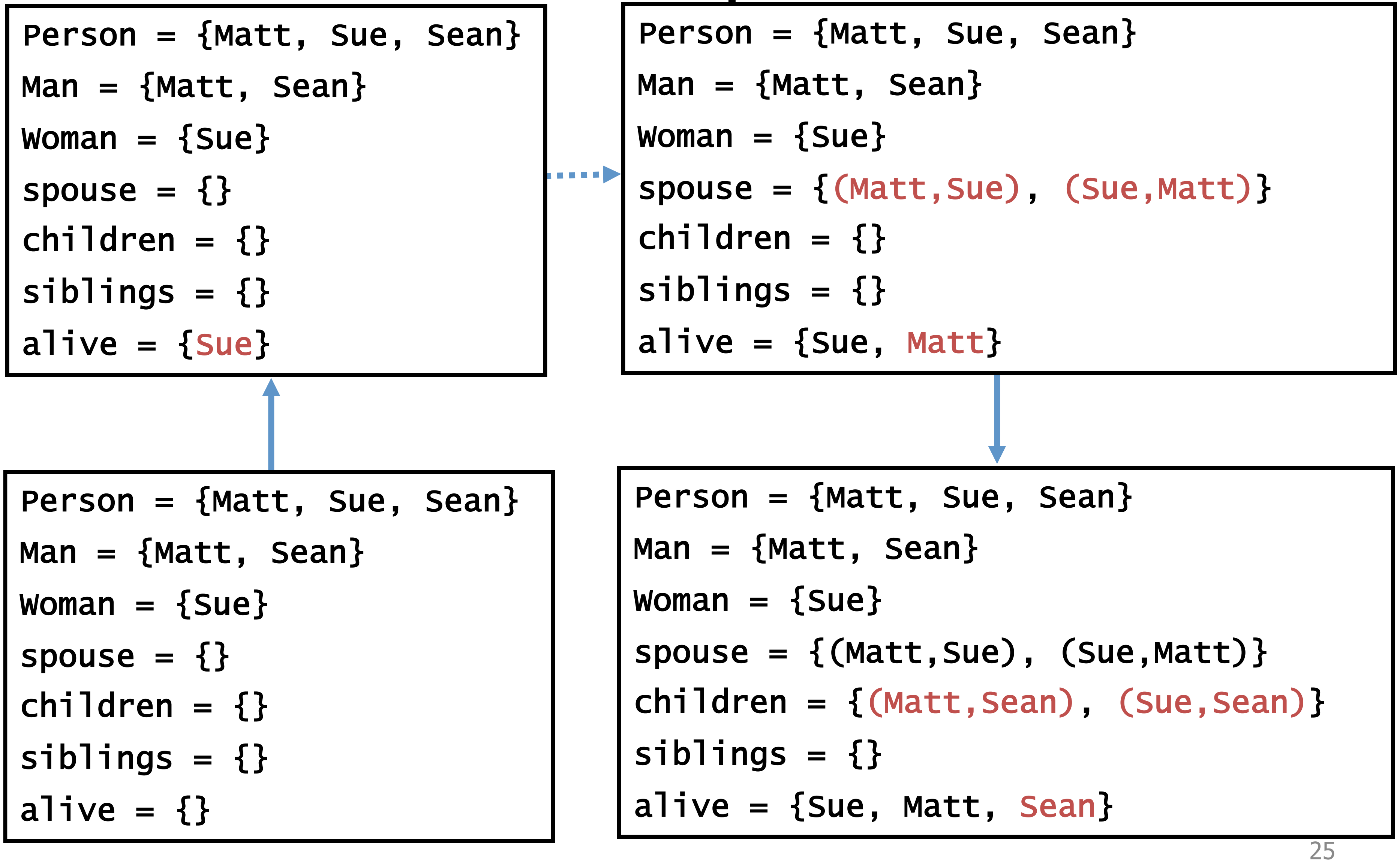
- **A person is born to parents**
  - Add to alive relation
  - Modify children/ parents relations

```
Person = {Matt, Sue, Sean}
Man = {Matt, Sean}
Woman = {Sue}
spouse = {(Matt,Sue), (Sue,Matt)}
children = {}
siblings = {}
alive = {Matt, Sue}
```

↓

```
Person = {Matt, Sue, Sean}
Man = {Matt, Sean}
Woman = {Sue}
spouse = {(Matt,Sue), (Sue,Matt)}
children = {(Matt,Sean), (Sue,Sean)}
siblings = {}
alive = {Matt, Sue, Sean}
```

State Sequences



```
Person = {Matt, Sue, Sean}
Man = {Matt, Sean}
Woman = {Sue}
spouse = {}
children = {}
siblings = {}
alive = {}
```

```
Person = {Matt, Sue, Sean}
Man = {Matt, Sean}
Woman = {Sue}
spouse = {}
children = {}
siblings = {}
alive = {Sue}
```

```
Person = {Matt, Sue, Sean}
Man = {Matt, Sean}
Woman = {Sue}
spouse = {(Matt,Sue), (Sue,Matt)}
children = {}
siblings = {}
alive = {Sue, Matt}
```

```
Person = {Matt, Sue, Sean}
Man = {Matt, Sean}
Woman = {Sue}
spouse = {(Matt,Sue), (Sue,Matt)}
children = {(Matt,Sean), (Sue,Sean)}
siblings = {}
alive = {Sue, Matt, Sean}
```

Express a transition in Alloy

- A transition can be modeled as a predicate between two states:
  - the state right before the transition and
  - the state right after it

- We define it as predicate with (at least) two formal parameters: `t, t’: Time`

- Constraints over time `t` (resp., `t’`) model the state right before (resp., after) the transition

Express a transition in Alloy

- Pre condition constraints
  - Describe the states to which the transition applies
- Post condition constraints
  - Describes the effects of the transition in generating the next state
- Frame condition constraints
  - Describes what does not change between pre-state and post-state of a transition

Distinguishing the pre, post and frame conditions in comments provides useful documentation

Example: Marriage

```
pred marriage [m: Man, w: Woman, t,t': Time] {
-- preconditions
   -- m and w must be alive
   m+w in alive.t
   -- neither one is married
   no (m+w).spouse.t
   -- they are not be blood relatives
   not BloodRelatives[m, w, t]
-- post-conditions
   -- w is m’s wife
   m.spouse.t' = w
   -- m is w’s husband
   -- (redundant)
-- frame conditions    ??
}
```

Frame condition

How is each relation touched by marriage?

- 5 relations :
  - `children, parents, siblings`
  - `spouse`
  - `alive`
- `parents` and `siblings` relations are defined in terms of the `children` relation
- Thus, the frame condition has only to consider `children`, `spouse` and `alive` relations

Frame condition predicates

```
pred noChildrenChangeExcept [ps: set Person
                             t,t': Time] {
  all p: Person - ps |
    p.children.t' = p.children.t
}

pred noSpouseChangeExcept [ps: set Person
                           t,t': Time] {
  all p: Person - ps |
    p.spouse.t' = p.spouse.t
}

pred noAliveChange [t,t': Time] {
  alive.t’ = alive.t
}
```

Example: Marriage

```
pred marriage [m: Man, w: Woman, t,t': Time]
{
-- preconditions
  m+w in alive.t
  no (m+w).spouse.t
  not BloodRelatives[m, w, t]
-- post-conditions
  m.spouse.t' = w
-- frame conditions
  noChildrenChangeExcept[none, t, t’]
  noSpouseChangeExcept[m+w, t, t’]
  noAliveChange[t, t’]
}
```

Instance of marriage

```
open ordering [Time] as T
…

pred marriageInstance {
  some t: Time |
  some m: Man | some w: Woman |
    let t' = T/next[t] |
      marriage[m, w, t, t’]
}
run { marriageInstance }
```

Example: Birth

```
pred birth[t, t': Time] {
  -- precondition and post-condition
  one p: Person |
    p !in alive.t and
    alive.t’ = alive.t + p
  -- frame condition
  noChildrenChangeExcept[none, t, t']
  noSpouseChangeExcept[none, t, t']
}
```

Example: Birth from parents

```
pred birthFromParents [m, w: Person, t,t': Time] {
-- precondition
   m+w in alive.t
   m.spouse.t = w
-- precondition and post-condition
    one p: Person | {
        -- precondition
        p !in alive.t
        -- postcondition
        alive.t’ = alive.t + p
        m.children.t’ = m.children.t + p
        w.children.t’ = w.children.t + p
  }
-- frame condition
    noChildrenChangeExcept[m+w, t, t’]
    noSpouseChangeExcept[none, t, t’]
}
```

Instance of birth

```
pred birthInstance {
  some t: Time |
    let t' = T/next[t] |
      birth[t, t']
}

pred birthFromParentsInstance {
  some t: Time |
  some m, w: Person |
    let t' = T/next[t] |
      birthFromParents[m, w, t, t’]
}
```

Specifying a transition system

- A transition system can be defined as a set of traces:
  - sequences of time steps generated by the operators
- In our example, for every trace:
  - The first time step satisfies some initialization condition
  - Each pair of consecutive steps are related by
    - a birth operation, or
    - a marriage operation, or
    - a birthFromParents operation

Initial State Specification

```
pred init [t: Time] {
  no children.t
  no spouse.t
  no alive.t
}
```

Trace Specification

```
pred Trace {
  init[T/first]
  all t: Time - T/last | let t’ = T/next[t] |
      birth[t, t’] or
      (one m: Man | one w: Woman |
         marriage[m, w, t, t’]) or
      (one m: Man | one w: Woman |
         birthFromParents[m, w, t, t’])
}
run {Trace and some Man and some Woman}
```

Realism Constraints

```
run {
  marriageInstance
  birthInstance
  birthFromParentsInstance
} for 5
```

Constraint about `alive` relation

```
-- only living people can have or be
-- children or have spouses
fact staticAlive {
  all t: Time | all p: Person |
  let mainRels = (children + spouse).t |
     p !in alive.t implies (
        no p.mainRels
        and
        no mainRels.p
     )
}
```

Exercises

- Load `family-7.als`
- Execute it
- Look at the generated instance
- Does it look correct?
- What if anything would you change about it?