Chapter 1

Background on Iris

- Concept of separation logic
- Any references
- Iris is a framework that supports different separations logics for different languages
- We will show Iris in the context of the language HeapLang
- Notation for thesis: `This` represents elements in Coq
- Start with introducing the grammar of Iris and HeapLang
- Then we will show the proof rules of Iris and how they are used in with HeapLang
- Then we will show the context of Iris and how it is used in the proof mode
- Then we will show how Iris tactics are constructed using Lemma's about the context
- Then we will show how we can use Elpi to extend the tactics of Iris
- Lastly talk about induction and fix-points in Iris

Question: Should I introduce HeapLang, as I don't need to actually verify any program in my thesis? But it would be more clear what Iris is if I introduce it.

Iris is a separation logic [Jun+15; Jun+16; Kre+17; Jun+18]. Propositions can be seen as predicates over resources, *e.g.,* heaps. Thus, there are a number of extra logical connectives such as `P * Q`, which represents that `P` and `Q` split up the resources into two disjoints in which they respectively hold. Moreover, hypotheses in our logic can often be used only once when proving something, they represent resources that we consume when used. To be able to reason in this logic in Coq a tactics' language has been added to Coq called the Iris Proof Mode (IPM) [KTB17; Kre+18].

1.1 Grammar

- Iris grammar is an extension on predicate logic grammar

$$\tau ::= T \mid 0 \mid 1 \mid \tau \to \tau$$
$$t, P ::= x \mid \mathsf{False} \mid \mathsf{True} \mid t =_\tau t \mid P \wedge P \mid P \vee P \mid P * P \mid P \mathrel{-\!\!*} P \mid$$
$$\exists x : \tau.\, P \mid \forall x : \tau.\, P \mid \Box P \mid \triangleright P \mid \{P\}\, t\, \{v.\, P\}$$

- Iris is expanded with a few extra connectives
- $*$ is separating conjunction, explain intuition, refer to rule
- $\mathrel{-\!\!*}$ is magic wand, explain intuition, refer to rule
- $\Box$ is always, explain intuition, refer to rule
- $\triangleright$ is later, explain intuition, refer to rule
- $\{P\}\, t\, \{v.\, P\}$ is a hoare triple, explain intuition, refer to rule

1.2 Proof rules

- We omit any standard proof rules
- We will show the proof rules for the separation logic connectives

$$\begin{array}{c} \mathsf{True} * P \dashv\vdash P \\ P * Q \;\vdash\; Q * P \\ (P * Q) * R \;\vdash\; P * (Q * R) \end{array} \qquad \frac{P_1 \vdash Q_1 \qquad P_2 \vdash Q_2}{P_1 * P_2 \vdash Q_1 * Q_2}\ {*\text{-MONO}} \qquad \frac{P * Q \vdash R}{P \vdash Q \mathrel{-\!\!*} R}\ {\mathrel{-\!\!*}\text{I-E}}$$

- These are the rules about the separating conjunction is commutative and associative.
- Magic wand is implication with disjunct resources.
- Resources can't be duplicated.

$$\frac{P \vdash Q}{\Box P \vdash \Box Q}\ {\Box\text{-MONO}} \qquad \frac{}{\Box P \vdash P}\ {\Box\text{-E}} \qquad \begin{array}{c} \Box P \wedge Q \vdash \Box P * Q \\ \Box P \vdash \Box \Box P \\ \forall x.\ \Box P \vdash \Box \forall x.\, P \\ \Box \exists x.\, P \vdash \exists x.\ \Box P \end{array}$$

- Persistence can only be introduced using the $\Box$-mono rule.
- Represents resources that can be duplicated and thus can be used multiple times.

- Persistence can always be eliminated.

$$\triangleright\text{-MONO}\ \frac{P \vdash Q}{\triangleright P \vdash \triangleright Q} \qquad \triangleright\text{-I}\ \ P \vdash \triangleright P \qquad \forall x.\ \triangleright P \vdash \triangleright \forall x.\ P \qquad \triangleright \exists x.\ P \vdash \triangleright \mathsf{False} \vee \exists x.\ \triangleright P \qquad \triangleright P \vdash \triangleright \mathsf{False} \vee (\triangleright \mathsf{False} \Rightarrow P)$$

$$\triangleright (P * Q) \dashv\vdash \triangleright P * \triangleright Q$$
$$\Box \triangleright P \dashv\vdash \triangleright \Box P$$

- Later is used to represent resources that can be used in the future.

1.3 Contexts

- Iris uses a named context instead of the entailment
- `env A` is a list of pairs from identifiers to `A`.

```
Inductive ident :=
  | IAnon : positive → ident
  | INamed :> string → ident.

Record envs (PROP : bi) := Envs {
  env_persistent : env PROP;
  env_spatial : env PROP;
  env_counter : positive;
}.
```

TODO: Explain *Prop*somewhere or just use *iProp*

TODO: I am simplifying the environments here, should I do that?

- Identifiers are either anonymous or named
- Environments are maps from identifiers to values
- The final context is two environments of propositions and a counter
- The first environment is the persistent context
- The second environment is the spatial context
- The two environments can't have overlapping identifiers
- The counter is used to always be able to generate a fresh anonymous identifier
- Semantics of the contexts is

```
Definition of_envs {PROP : bi}
    (Γp Γs : env PROP) : PROP :=
  (□ [∧] Γp ∧ [*] Γs)%I.
```

Question: Should I differentiate between □ and <pers>

- The persistent environment is combined with ∧ and surrounded by a □.
- The spatial environment is combined with ∗
- We can now write our entailment as

```
Definition envs_entails {PROP : bi}
    (Δ : envs PROP) (Q : PROP) : Prop :=
  of_envs (env_intuitionistic Δ) (env_spatial Δ) ⊢ Q.
```

- This is represented in the proof state as

```
P, Q, R: iProp
============
"HP" : P
------------□
"HR" : Q
------------*
R
```

- P is a persistent proposition
- Q is a spatial proposition
- We need to proof R

1.4 Tactics

- The proof rules are hard to use with the context
- Define Lemma's that work with the context
- These allow us to define our tactics easily

```
Lemma tac_wand_intro Δ i P Q :
  match envs_app false (Esnoc Enil i P) Δ with
  | None ⇒ False
  | Some Δ' ⇒ envs_entails Δ' Q
  end →
  envs_entails Δ (P -* Q).
```

- Introduces a magic wand
- Add introduced proposition to the spatial context
- The condition we need to satisfy is a Coq function that resolves to Q with P added to the context
- If `i` already exists in the context, we have to proof False
- Tactics now just process the arguments and call necessary Lemma's

The tactics the IPM adds are build to replicate many of the behaviors of the Coq tactics while manipulating the Iris contexts. In the next section we will show how the Iris variant of the **`intros`** tactic works.

1.5 `iIntros` example

`iIntros` is based on the Coq **`intros`** tactic. The Coq **`intros`** tactic makes use of a domain specific language (DSL) for quickly introducing different logical connective. In Iris this concept was adopted for the `iIntros` tactic, but adopted to the Iris contexts. Also, a few expansions, as inspired by ssreflect [HKP97; GMT16], were added to perform other common initial proof steps such as **`simpl`**, **`done`** and others. We will show a few examples of how `iIntros` can be used to help prove lemmas.

We begin with a lemma about the magic wand. The magic wand can be seen as the implication of separation logic which also takes into account the separation of resources.

$$\frac{P * Q \vdash R}{P \vdash Q \mathrel{-\!\!*} R} \mathrel{-\!\!*}\text{-Intro} \qquad \frac{P \wedge Q \vdash R}{P \vdash Q \to R} \to\text{-Intro}$$

Thus, where a normal implication introduction adds the left-hand side to the Coq context, the magic wand adds the left-hand side to the spatial resource context.

TODO: Rewrite when I have a solid explanation of the Iris contexts

```
P, R: iProp
============
------------*
P -* R -* P
```

When using `iIntros "HP HR"`, the proof state is transformed into the following state.

```
P, R: iProp
============
"HP" : P
"HR" : R
------------*
P
```

We have introduced the two separation logic propositions into the spatial context. This does not only work on the magic wand, we can also use this to introduce more complicated statements. Take the following proof state,

```
P: nat → iProp
============================================
--------------------------------------------*
∀ x : nat, (∃ y : nat, P x * P y) ∨ P 0 -* P 1
```

It consists of a universal quantification, an existential quantification, a conjunction and a disjunction. We can again use one application of `iIntros` to introduce and eliminate the premise. `iIntros "%x [[%y [Hx Hy]] | H0]"` takes the proof to the following state of two goals

```
(1/2)
P: nat → iProp
x, y: nat
==================
"Hx" : P x
"Hy" : P y
------------------*
P 1

(2/2)
P: nat → iProp
x: nat
==================
"H0" : P 0
------------------*
P 1
```

The intro pattern consists of multiple sub intro patterns. Each sub intro pattern starts with a forall introduction or wand introduction. We then interpret the intro pattern for the introduced hypothesis. They can have the following interpretations:

- `"H"` represents renaming a hypothesis. The name given is used as the name of the hypothesis in the spatial context.
- `"%H"` represents pure elimination. The introduced hypothesis is interpreted as a Coq hypothesis, and added to the Coq context.
- `"[IPL | IPR]"` represents disjunction elimination. We perform a disjunction elimination on the introduced hypothesis. Then, we apply the two included intro patterns two the two cases created by the disjunction elimination.
- `"[IPL IPR]"` represents separating conjunction elimination. We perform a separating conjunction elimination. Then, we apply the two included intro patterns two the two hypotheses by the separating conjunction elimination.
- `"[\%x IP]"` represents existential elimination. If first element of a separating conjunction pattern is a pure elimination we first try to eliminate an exists in the hypothesis and apply the included intro pattern on the resulting hypothesis. If that does not succeed we do a conjunction elimination.

Thus, we can break down `iIntros "\%x [[\%y [Hx Hy]] | H0]"` into its components. We first forall introduce or first sub intro pattern `"\%x"` and then perform the second case, introduce a pure Coq variable for the `∀ x : nat`. Next we wand introduce for the second sub intro pattern, `"[[\%y [Hx Hy]] | H0]"` and interpret the outer pattern. it is the third case and eliminates the disjunction, resulting in two goals. The left patterns of the seperating conjunction pattern eliminates the exists and adds the `y` to the Coq context. Lastly, `"[Hx Hy]"` is the fourth case and eliminates the seperating conjunction in the Iris context by splitting it into two assumptions `"Hx"` and `"Hy"`.

There are more patterns available to introduce more complicated goals, these can be found in a paper written by Krebbers, Timany, and Birkedal [KTB17].

1.6 Induction or something

- Different fix-points in use
- Coq fix-points
- Only works on terms where recursive call is under a Coq constructor
- Banach fix-points
- Only work on terms where recursive call is under a later

- Least and Greatest fix-points
- More flexible
- More work to prove induction principle?