Circular lists in Iris $*$ deduction rules of $\triangleright$

Herman Bergwerf

March 11th 2022

Overview

- The $\triangleright$ modality and standard deduction rules.
- Step-indexed propositions and linear arithmetic.
- The comparison rule and formula reduction.
- Proving completeness and decidability.
- *(if there is time)* Verification of a circular list in Coq/Iris.

Deduction rules of $\triangleright$

Circular lists in Iris

Program safety

- **Goal** Show that a program does not crash, and satisfies a post-condition *whenever* it terminates.
- **Application** Progams where termination is not guaranteed, or where strict totality is not important.
- **Method** Given that program x is *safe* for n steps, show that it is *safe* for $n + 1$ steps (*i.e.* induction).

Why $\triangleright$?

- **Syntactic tool** We can completely hide step counters using $\triangleright$.
- **Meaning** $\triangleright P$ means P is true *after* one step.
- **More steps** Once P is true, it remains true: $P \vdash \triangleright P$.
- **Löb induction** We can recover inductions: $\triangleright P \Rightarrow P \vdash P$.

Question

Are we still missing deduction rules for $\triangleright$?

Formulas

We study *propositional logic* with $\triangleright$.

Let $\Sigma = \{P_0, P_1, ...\}$ be proposition letters.

$$\varphi_0, \varphi_1 \in \mathcal{L}_\triangleright ::= \top \mid \bot \mid P_i \in \Sigma \mid \varphi_0 \wedge \varphi_1 \mid \varphi_0 \vee \varphi_1 \mid \varphi_0 \Rightarrow \varphi_1 \mid \triangleright\varphi_0$$

Deduction rules I

$$\frac{}{\varphi \vdash \varphi}\ \text{refl} \qquad \frac{\varphi_0 \vdash \varphi_1 \qquad \varphi_1 \vdash \varphi_2}{\varphi_0 \vdash \varphi_2}\ \text{trans}$$

$$\frac{}{\varphi \vdash \top}\ \top\text{-intro} \qquad \frac{}{\bot \vdash \varphi}\ \bot\text{-elim}$$

$$\frac{\sigma \vdash \varphi_0 \qquad \sigma \vdash \varphi_1}{\sigma \vdash \varphi_0 \wedge \varphi_1}\ \wedge\text{-intro} \qquad \frac{\varphi_0 \vdash \varphi_2 \qquad \varphi_1 \vdash \varphi_2}{\varphi_0 \vee \varphi_1 \vdash \varphi_2}\ \vee\text{-elim}$$

$$\frac{}{\varphi_0 \wedge \varphi_1 \vdash \varphi_0}\ \wedge\text{-elim-l} \qquad \frac{}{\varphi_0 \vdash \varphi_0 \vee \varphi_1}\ \vee\text{-intro-l}$$

$$\frac{}{\varphi_0 \wedge \varphi_1 \vdash \varphi_1}\ \wedge\text{-elim-r} \qquad \frac{}{\varphi_1 \vdash \varphi_0 \vee \varphi_1}\ \vee\text{-intro-r}$$

$$\frac{\sigma \wedge \varphi_0 \vdash \varphi_1}{\sigma \vdash \varphi_0 \Rightarrow \varphi_1}\ \Rightarrow\text{-intro} \qquad \frac{\sigma \vdash \varphi_0 \Rightarrow \varphi_1 \qquad \sigma \vdash \varphi_0}{\sigma \vdash \varphi_1}\ \Rightarrow\text{-elim}$$

Deduction rules II

$$\frac{}{\varphi_0 \vdash \triangleright\varphi_0}\ \triangleright\text{-intro} \qquad \frac{}{\triangleright\varphi_0 \wedge \triangleright\varphi_1 \vdash \triangleright(\varphi_0 \wedge \varphi_1)}\ \triangleright\text{-conj}$$

$$\frac{\top \vdash \triangleright\varphi}{\top \vdash \varphi}\ \triangleright\text{-elim} \qquad \frac{\varphi_0 \vdash \varphi_1}{\triangleright\varphi_0 \vdash \triangleright\varphi_1}\ \triangleright\text{-mono} \qquad \frac{\triangleright\varphi \vdash \varphi}{\top \vdash \varphi}\ \triangleright\text{-Löb}$$

A deduction tree

$$
\dfrac{
  \dfrac{
    \dfrac{
      \dfrac{}{\ldots \vdash P \Rightarrow Q}
      \qquad
      \dfrac{
        \dfrac{}{\ldots \vdash \triangleright Q \Rightarrow P}
        \qquad
        \dfrac{}{\ldots \vdash \triangleright Q}
      }{\ldots \vdash P}\ \Rightarrow\text{-elim}
    }{\triangleright Q \Rightarrow P \wedge P \Rightarrow Q \wedge \triangleright Q \vdash Q}\ \Rightarrow\text{-elim}
  }{\triangleright Q \Rightarrow P \wedge P \Rightarrow Q \vdash \triangleright Q \Rightarrow Q}\ \Rightarrow\text{-intro}
  \qquad
  \dfrac{}{\triangleright Q \Rightarrow Q \vdash Q}\ \text{strong-Löb}
}{
  \dfrac{\triangleright Q \Rightarrow P \wedge P \Rightarrow Q \vdash Q}{\triangleright Q \Rightarrow P \vdash (P \Rightarrow Q) \Rightarrow Q}\ \Rightarrow\text{-intro}
}\ \text{trans}
$$

Models

We want to interpret entailments $\varphi_0 \vdash \varphi_1$ in a model $\mathfrak{A}$.

- A domain $\dot{\mathfrak{A}}$ (denoted with a dot)
- A binary relation $\sqsubseteq_{\mathfrak{A}} \subseteq \dot{\mathfrak{A}} \times \dot{\mathfrak{A}}$ to interpret $\vdash$
- A denotation $\mathfrak{A}[\![\varphi]\!](\Gamma) \in \dot{\mathfrak{A}}$, where $\varphi \in \mathcal{L}_{\triangleright}$ and $\Gamma : \Sigma \to \dot{\mathfrak{A}}$

Entailment realization: $\varphi_0 \models_{\mathfrak{A}} \varphi_1 := \forall \Gamma.\ \mathfrak{A}[\![\varphi_0]\!](\Gamma) \sqsubseteq_{\mathfrak{A}} \mathfrak{A}[\![\varphi_1]\!](\Gamma)$

Step-indexed propositions

Downwards closed binary sequences.

$$
\begin{aligned}
\dot{\mathfrak{B}} &:= \{\alpha : \mathbb{N} \to \{0,1\} \mid \forall i \forall j \leq i.\ \alpha(i) \to \alpha(j)\} \\
\alpha \sqsubseteq_{\mathfrak{B}} \beta &:= \forall i.\ \alpha(i) \to \beta(i) \\
\mathfrak{B}[\![P_i]\!](\Gamma) &:= \Gamma(P_i) \\
\mathfrak{B}[\![\bot]\!](\Gamma) &:= \lambda i.\ 0 \\
\mathfrak{B}[\![\top]\!](\Gamma) &:= \lambda i.\ 1 \\
\mathfrak{B}[\![\varphi_0 \wedge \varphi_1]\!](\Gamma) &:= \lambda i.\ \mathfrak{B}[\![\varphi_0]\!](\Gamma)(i) \wedge \mathfrak{B}[\![\varphi_1]\!](\Gamma)(i) \\
\mathfrak{B}[\![\varphi_0 \vee \varphi_1]\!](\Gamma) &:= \lambda i.\ \mathfrak{B}[\![\varphi_0]\!](\Gamma)(i) \vee \mathfrak{B}[\![\varphi_1]\!](\Gamma)(i) \\
\mathfrak{B}[\![\varphi_0 \Rightarrow \varphi_1]\!](\Gamma) &:= \lambda i.\ \forall j \leq i.\ \mathfrak{B}[\![\varphi_0]\!](\Gamma)(j) \to \mathfrak{B}[\![\varphi_1]\!](\Gamma)(j) \\
\mathfrak{B}[\![\triangleright\varphi]\!](\Gamma) &:= \lambda i.\ \textbf{if } i = 0 \textbf{ then } 1 \textbf{ else } \mathfrak{B}[\![\varphi]\!](\Gamma)(i-1)
\end{aligned}
$$

Linear integer arithmetic

Number of 1's in a downwards closed sequence.

$$\begin{aligned}
\dot{\mathfrak{N}} &:= \mathbb{N} \cup \{\omega\} \\
p \sqsubseteq_{\mathfrak{N}} q &:= p \leq q \\
\mathfrak{N}[\![P_i]\!](\Gamma) &:= \Gamma(P_i) \\
\mathfrak{N}[\![\bot]\!](\Gamma) &:= 0 \\
\mathfrak{N}[\![\top]\!](\Gamma) &:= \omega \\
\mathfrak{N}[\![\varphi_0 \wedge \varphi_1]\!](\Gamma) &:= \min\{\mathfrak{N}[\![\varphi_0]\!](\Gamma),\ \mathfrak{N}[\![\varphi_1]\!](\Gamma)\} \\
\mathfrak{N}[\![\varphi_0 \vee \varphi_1]\!](\Gamma) &:= \max\{\mathfrak{N}[\![\varphi_0]\!](\Gamma),\ \mathfrak{N}[\![\varphi_1]\!](\Gamma)\} \\
\mathfrak{N}[\![\varphi_0 \Rightarrow \varphi_1]\!](\Gamma) &:= \textbf{if } \mathfrak{N}[\![\varphi_0]\!](\Gamma) \leq \mathfrak{N}[\![\varphi_1]\!](\Gamma) \textbf{ then } \omega \textbf{ else } \mathfrak{N}[\![\varphi_1]\!](\Gamma) \\
\mathfrak{N}[\![\triangleright\varphi]\!](\Gamma) &:= \textbf{if } \mathfrak{N}[\![\varphi]\!](\Gamma) = n \in \mathbb{N} \textbf{ then } n+1 \textbf{ else } \omega
\end{aligned}$$

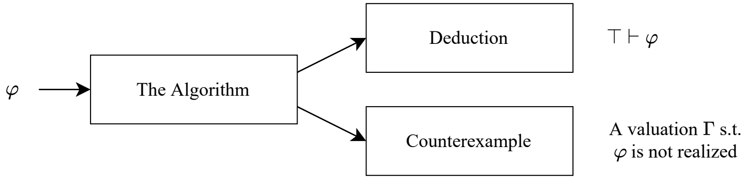
The comparison rule

$$\forall\, m\, n.\; m \leq n \vee m > n$$

$$\top \vdash \varphi_0 \Rightarrow \varphi_1 \vee \triangleright \varphi_1 \Rightarrow \varphi_0$$

Completeness and decidability

- **Completeness** Every entailment that is realized can be derived.
- **Decidability** This derivation is constructed using an algorithm.
- **Formalization** This is implemented and verified in Coq.



'$\mathbb{N} \cup \{\omega\}$'-truth-tables

P	Q	$(\triangleright P \Rightarrow Q \wedge Q \Rightarrow P) \Rightarrow P$
0	0	$\min\{(0+1) \Rightarrow 0, 0 \Rightarrow 0\} \Rightarrow 0 \equiv \min\{0, \omega\} \Rightarrow 0 \equiv \omega$
$\vdots$	$\vdots$	$\vdots$
10	50	$\min\{(10+1) \Rightarrow 50, 50 \Rightarrow 10\} \Rightarrow 10 \equiv \min\{\omega, 10\} \Rightarrow 10 \equiv \omega$
$\vdots$	$\vdots$	$\vdots$
ω	ω	$\min\{\omega \Rightarrow \omega, \omega \Rightarrow \omega\} \Rightarrow \omega \equiv \min\{\omega, \omega\} \Rightarrow \omega \equiv \omega$

A finite number of cases

$$P \leq Q \vee P > Q$$

$$P < Q \vee P = Q \vee P > Q$$

$$P + 2 \leq Q \vee P + 1 = Q \vee P = Q \vee P = Q + 1 \vee P \geq Q + 2$$

Formula reductions

$$\varphi_0 \Rightarrow \varphi_1 \vdash (\varphi_0 \wedge \varphi_1) \Leftrightarrow \varphi_0 \quad (1)$$

$$\varphi_1 \Rightarrow \varphi_0 \vdash (\varphi_0 \wedge \varphi_1) \Leftrightarrow \varphi_1 \quad (2)$$

$$\varphi_0 \Rightarrow \varphi_1 \vdash (\varphi_0 \vee \varphi_1) \Leftrightarrow \varphi_1 \quad (3)$$

$$\varphi_1 \Rightarrow \varphi_0 \vdash (\varphi_0 \vee \varphi_1) \Leftrightarrow \varphi_0 \quad (4)$$

$$\varphi_0 \Rightarrow \varphi_1 \vdash (\varphi_0 \Rightarrow \varphi_1) \Leftrightarrow \top \quad (5)$$

$$\triangleright\ \varphi_1 \Rightarrow \varphi_0 \vdash (\varphi_0 \Rightarrow \varphi_1) \Leftrightarrow \varphi_1 \quad (6)$$

Variables and modal depth

$$
\begin{aligned}
\mathrm{FV}(P_i) &:= \{P_i\} \\
\mathrm{FV}(\triangleright\varphi) &:= \mathrm{FV}(\varphi) \\
\mathrm{FV}(\varphi_0 \mathbin{\square} \varphi_1) &:= \mathrm{FV}(\varphi_0) \cup \mathrm{FV}(\varphi_1)
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{MD}(P_i) &:= 0 \\
\mathrm{MD}(\triangleright\varphi) &:= 1 + \mathrm{MD}(\varphi) \\
\mathrm{MD}(\varphi_0 \mathbin{\square} \varphi_1) &:= \max\{\mathrm{MD}(\varphi_0), \mathrm{MD}(\varphi_1)\}
\end{aligned}
$$

$$\square \in \{\wedge, \vee, \Rightarrow\}$$

The reduction theorem

Definition

A formula τ is a φ-atomic formula if there exists an $n \leq \mathrm{MD}(\varphi)$ and an $x \in \{\top, \bot\} \cup \mathrm{FV}(\varphi)$ such that $\tau = \triangleright^n x$.

The reduction theorem

Definition
A formula τ is a φ-atomic formula if there exists an $n \leq \mathrm{MD}(\varphi)$ and an $x \in \{\top, \bot\} \cup \mathrm{FV}(\varphi)$ such that $\tau = \triangleright^n x$.

Definition
A formula σ is exhaustive for φ if for every two φ-atomic formulas τ_0 and τ_1 either $\sigma \vdash \tau_0 \Rightarrow \tau_1$ or $\sigma \vdash \triangleright\tau_1 \Rightarrow \tau_0$.

The reduction theorem

Definition
A formula τ is a φ-atomic formula if there exists an $n \leq \mathrm{MD}(\varphi)$ and an $x \in \{\top, \bot\} \cup \mathrm{FV}(\varphi)$ such that $\tau = \triangleright^n x$.

Definition
A formula σ is exhaustive for φ if for every two φ-atomic formulas τ_0 and τ_1 either $\sigma \vdash \tau_0 \Rightarrow \tau_1$ or $\sigma \vdash \triangleright\tau_1 \Rightarrow \tau_0$.

Theorem
If σ is exhaustive for φ then there exists a φ-atomic formula τ such that $\sigma \vdash \varphi \Leftrightarrow \tau$.

Coq formalization (definitions)

```
Inductive deduction : form term → form term → Prop :=
  | d_refl p            : p ⊢ p
  | d_trans p q r       : p ⊢ q → q ⊢ r → p ⊢ r
  | d_true_intro p      : p ⊢ ⊤
  | d_false_elim p      : ⊥ ⊢ p
  | d_conj_intro c p q : c ⊢ p → c ⊢ q → c ⊢ p `∧` q
  | d_conj_elim_l p q  : p `∧` q ⊢ p
  | d_conj_elim_r p q  : p `∧` q ⊢ q
  | d_disj_intro_l p q : p ⊢ p `∨` q
  | d_disj_intro_r p q : q ⊢ p `∨` q
  | d_disj_elim p q r  : p ⊢ r → q ⊢ r → p `∨` q ⊢ r
  | d_impl_intro c p q : c `∧` p ⊢ q → c ⊢ p ⇒ q
  | d_impl_elim c p q  : c ⊢ p ⇒ q → c ⊢ p → c ⊢ q
  | d_later_intro p    : p ⊢ ▷p
  | d_later_elim p     : ⊢ ▷p → ⊢ p
  | d_later_fix p      : ▷p ⊢ p → ⊢ p
  | d_later_mono p q   : p ⊢ q → ▷p ⊢ ▷q
  | d_later_conj p q   : ▷p `∧` ▷q ⊢ ▷(p `∧` q)
  | d_compare p q      : ⊢ p ⇒ q `∨` ▷q ⇒ p
  where "p ⊢ q" := (deduction p q) and "⊢ q" := (⊤ ⊢ q).
```

Coq formalization (results)

- Soundness

```
Theorem deduction_sound Γ p q :
  p ⊢ q → realizes Γ p q.
```

- Decidability

```
Theorem deduction_decidable p q :
  { p ⊢ q } + { ∃ Γ, ¬ realizes Γ p q }.
```

- Completeness

```
Corollary deduction_complete p q :
  (∀ Γ, realizes Γ p q) → p ⊢ q.
```

Deduction rules of ▷

Circular lists in Iris

Separation logic

- **Pre- and postconditions** Hoare triples: $\{P\}C\{Q\}$
- **Mutable resources** The *points to* predicate: $l \mapsto v$
- **Heap modularity** The separating conjunction: $P * Q$

Examples:

- $\{l \mapsto 1\}\ l := {}^*l + {}^*l\ \{l \mapsto 2\}$
- $\{list(l_0, \vec{v}) * list(l_1, \vec{w})\}\ merge(l_0, l_1)\ \{list(l_0, v\vec{w}) * list(l_1, \vec{w})\}$

Coq/Iris

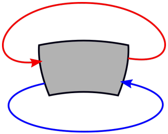
- **HeapLang** Functional, mutable references, concurrency
- **Hoare triples** `[[{ P }]] program [[{ x, RET x; Q x }]]`
- **Coq tactics** `iIntros`, `iExists`, `wp_pures`, …

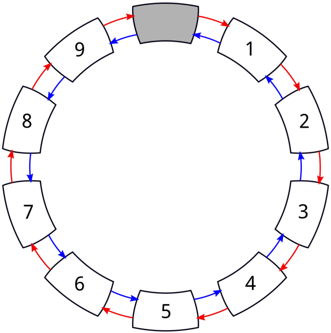
Goal

Total correctness of *insert* and *delete* for a doubly linked circular list.

Circular lists

- **Doubly linked** A node is a tuple (*prev*, *next*, *value*)
- **Dummy nodes** Values are `NONE` or `SOME v`





Operations

- **Utilities** `get_prev`, `get_next`, `set_prev`, `set_next`
- **List interface** `make`, `insert`, `delete`
- **Deque interface** `push_front`, `push_back`, `pop_front`, `pop_back`

Implementation

```
Definition make : val :=
  λ: <>,
    let: "node" := ref NONE in
    "node" ← (("node", "node"), NONE);;
    "node".
```

Implementation

```
Definition make : val :=
  λ: <>,
    let: "node" := ref NONE in
    "node" ← (("node", "node"), NONE);;
    "node".
```

```
Definition insert : val :=
  λ: "prev" "v",
    let: "next" := get_next "prev" in
    let: "node" := ref ("prev", "next", SOME "v") in
    set_next "prev" "node";;
    set_prev "next" "node";;
    "node".
```

List predicates

- **List segment** The nodes are linked properly.

  ```
  Fixpoint dseg (prev after : loc)
    (nodes : list (loc * option val)) : iProp
  ```

- **Circular list** The first and last node are connected.

  ```
  Definition dlist
    (nodes : list (loc * option val)) : iProp
  ```

- **Value list** A list of values is stored in a circular list.

  ```
  Definition deque (l : loc)
    (vs : list val) : iProp
  ```

Specifications (dseg)

We show that nodes in `dlist` can be rotated.

```
Lemma dseg_split lA lB vB lC vC lD ns1 ns2 :
  dseg lA lD (ns1 ++ (lB, vB) :: (lC, vC) :: ns2) -∗
    dseg lA lC (ns1 ++ [(lB, vB)]) ∗
    dseg lB lD ((lC, vC) :: ns2).
```

```
Lemma dseg_glue lA lB vB lC vC lD ns1 ns2 :
  dseg lA lC (ns1 ++ [(lB, vB)]) ∗
  dseg lB lD ((lC, vC) :: ns2) -∗
    dseg lA lD (ns1 ++ (lB, vB) :: (lC, vC) :: ns2).
```

Specifications (dseg)

We show that nodes in `dlist` can be rotated.

```
Lemma dseg_split lA lB vB lC vC lD ns1 ns2 :
  dseg lA lD (ns1 ++ (lB, vB) :: (lC, vC) :: ns2) -∗
    dseg lA lC (ns1 ++ [(lB, vB)]) ∗
    dseg lB lD ((lC, vC) :: ns2).
```

```
Lemma dseg_glue lA lB vB lC vC lD ns1 ns2 :
  dseg lA lC (ns1 ++ [(lB, vB)]) ∗
  dseg lB lD ((lC, vC) :: ns2) -∗
    dseg lA lD (ns1 ++ (lB, vB) :: (lC, vC) :: ns2).
```

```
Lemma dlist_step n ns :
  dlist (n :: ns) ⊣⊢ dlist (ns ++ [n]).
```

Specifications (dlist)

```
Lemma make_spec :
  [[{ True }]]
    make #()
  [[{ l, RET #l; dlist [(l, None)] }]].
```

Specifications (dlist)

```
Lemma make_spec :
  [[{ True }]]
    make #()
  [[{ l, RET #l; dlist [(l, None)] }]].
```

```
Lemma insert_spec l0 v0 v ns :
  [[{ dlist ((l0, v0) :: ns) }]]
    insert #l0 v
  [[{ l, RET #l; dlist ((l0, v0) :: (l, Some v) :: ns) }]].
```

Specifications (deque)

```
Definition push_back : val :=
  λ: "dq" "v", insert (get_prev "dq") "v";; #().
Definition pop_back : val :=
  λ: "dq", delete (get_prev "dq").
```

Specifications (deque)

```
Definition push_back : val :=
  λ: "dq" "v", insert (get_prev "dq") "v";; #().
Definition pop_back : val :=
  λ: "dq", delete (get_prev "dq").
```

```
Lemma push_back_spec l v vs :
  [[{ deque l vs }]]
    push_back #l v
  [[{ RET #(); deque l (vs ++ [v]) }]].
```

Specifications (deque)

```
Definition push_back : val :=
  λ: "dq" "v", insert (get_prev "dq") "v";; #().
Definition pop_back : val :=
  λ: "dq", delete (get_prev "dq").
```

```
Lemma push_back_spec l v vs :
  [[{ deque l vs }]]
    push_back #l v
  [[{ RET #(); deque l (vs ++ [v]) }]].
```

```
Lemma pop_back_spec l v vs :
  [[{ deque l (vs ++ [v]) }]]
    pop_back #l
  [[{ RET (SOMEV v); deque l vs }]].
```

DOI: 10.5281/zenodo.6340500