

FELLOWSHIP: A SHORT MANUAL

FLORENT KIRCHNER

ABSTRACT. For any assistance regarding Fellowship’s proof language, the `help` command of the toplevel is recommended and provides thorough documentation. This command however does not provide a detailed insight of the logical structure of the prover. This note intends to provide the curious users with a fine breakdown of the prover’s implementation.

1. INFERENCE RULES

1.1. **Classical logic sequent calculus.** Figure 1 gives the inference rules for classical logic sequent calculus.

$\text{axiom}_{\mathcal{L}} \frac{}{\Gamma; * : A \vdash \Delta, x : A, \Delta'}$	$\text{axiom}_{\mathcal{R}} \frac{}{\Gamma, x : A, \Gamma' \vdash * : A; \Delta}$
$\text{false}_{\mathcal{L}} \frac{}{\Gamma; * : \text{false} \vdash \Delta}$	$\text{true}_{\mathcal{R}} \frac{}{\Gamma \vdash * : \text{true}; \Delta}$
$\text{cut}_{\mathcal{L}} \frac{x : A, \Gamma \vdash * : B; \Delta}{\Gamma; * : A \vdash \Delta}$	$\text{cut}_{\mathcal{R}} \frac{x : A, \Gamma; * : B \vdash \Delta}{\Gamma \vdash * : A; \Delta}$
$\text{imply}_{\mathcal{L}} \frac{\Gamma \vdash * : A; \Delta \quad \Gamma; * : B \vdash \Delta}{\Gamma; * : A \Rightarrow B \vdash \Delta}$	$\text{imply}_{\mathcal{R}} \frac{\Gamma, x : A \vdash * : B; \Delta}{\Gamma \vdash * : A \Rightarrow B; \Delta}$
$\text{conj}_{\mathcal{L}} \frac{\Gamma, x : A, y : B \vdash * : C; \Delta}{\Gamma; * : A \wedge B \vdash \Delta}$	$\text{conj}_{\mathcal{R}} \frac{\Gamma \vdash * : A; \Delta \quad \Gamma \vdash * : B; \Delta}{\Gamma \vdash * : A \wedge B; \Delta}$
$\text{disj}_{\mathcal{L}} \frac{\Gamma; * : A \vdash \Delta \quad \Gamma; * : B \vdash \Delta}{\Gamma; * : A \vee B \vdash \Delta}$	$\text{disj}_{\mathcal{R}1} \frac{\Gamma \vdash * : A; \Delta}{\Gamma \vdash * : A \vee B; \Delta}$
$\text{neg}_{\mathcal{L}} \frac{\Gamma \vdash * : A; \Delta}{\Gamma; * : \neg A \vdash \Delta}$	$\text{disj}_{\mathcal{R}2} \frac{\Gamma \vdash * : B; \Delta}{\Gamma \vdash * : A \vee B; \Delta}$
$\text{forall}_{\mathcal{L}} \frac{\Gamma; * : B[x \leftarrow t] \vdash \Delta}{\Gamma; * : \forall(x : A)B \vdash \Delta}$	$\text{neg}_{\mathcal{R}} \frac{\Gamma; * : A \vdash \Delta}{\Gamma \vdash * : \neg A; \Delta}$
$\text{exists}_{\mathcal{L}} \frac{\Gamma; * : B \vdash \Delta}{\Gamma; * : \exists(x : A)B \vdash \Delta}$	$\text{forall}_{\mathcal{R}} \frac{\Gamma \vdash * : B; \Delta}{\Gamma \vdash * : \forall(x : A)B; \Delta}$
	$\text{exists}_{\mathcal{R}} \frac{\Gamma \vdash * : B[x \leftarrow t]; \Delta}{\Gamma \vdash * : \exists(x : A)B; \Delta}$

FIGURE 1. Classical logic sequent calculus.

The usual contraction and weakening rules are implemented:

$$\text{contr}_{\mathcal{R}} \frac{\Gamma \vdash * : C; x : C, \Delta}{\Gamma \vdash * : C; \Delta} \quad \text{contr}_{\mathcal{L}} \frac{\Gamma, x : C; * : C \vdash \Delta}{\Gamma; * : C \vdash \Delta}$$

$$\text{weak}_{1\mathcal{R}} \frac{\Gamma \vdash * : C; \Delta}{\Gamma, x : A \vdash * : C; \Delta} \quad \text{weak}_{2\mathcal{R}} \frac{\Gamma \vdash * : C; \Delta}{\Gamma \vdash * : C; x : A, \Delta}$$

Of course, the left (trivial) counterparts of the weakening rules are also valid.

In the case of subtractive logic, the elimination rules for the $-$ connective are added:

$$\text{minus}_{\mathcal{L}} \frac{\Gamma; * : A \vdash x : B, \Delta}{\Gamma; * : A - B \vdash \Delta} \quad \text{minus}_{\mathcal{R}} \frac{\Gamma; * : B \vdash \Delta \quad \Gamma \vdash * : A; \Delta}{\Gamma \vdash * : A - B; \Delta}$$

Finally in the case of minimal logic, the $\text{neg}_{\mathcal{L}}$ and $\text{neg}_{\mathcal{R}}$ rules are discarded, as well as the $\text{false}_{\mathcal{L}}$ rule. $\neg A$ is considered as syntactic sugar for $A \Rightarrow \text{false}$, and is eliminated using the rules for implication.

1.2. Syntax of proof terms. $\bar{\lambda}\mu\tilde{\mu}$ -terms [1] are used as proof terms. The syntax of the $\bar{\lambda}\mu\tilde{\mu}$ -calculus defines commands c , terms v and contexts e as follows:

$$\begin{aligned} c &::= \langle v|e \rangle \\ v &::= x \parallel \top \parallel \lambda x.v \parallel e \cdot v \parallel (v, v') \parallel \text{inj}_r v \parallel \text{inj}_l v \parallel \mu\alpha.c \\ e &::= \alpha \parallel \perp \parallel v \cdot e \parallel \alpha\lambda'.e \parallel \text{proj}[x, x', c] \parallel [e, e'] \parallel \tilde{\mu}x.c \end{aligned}$$

1.3. Typing rules. Figure 2 describes the type inference rules: these are simply the deduction rules of Figure 1, annotated by terms.

$\text{axiom}_{\mathcal{L}} \frac{}{\Gamma; \alpha : A \vdash \Delta, \alpha : A, \Delta'}$	$\text{axiom}_{\mathcal{R}} \frac{}{\Gamma, x : A, \Gamma' \vdash x : A; \Delta}$
$\text{false}_{\mathcal{L}} \frac{}{\Gamma; \perp : \text{false} \vdash \Delta}$	$\text{true}_{\mathcal{R}} \frac{}{\Gamma \vdash \top : \text{true}; \Delta}$
$\text{cut}_{\mathcal{L}} \frac{x : A, \Gamma \vdash v : B; \Delta}{\Gamma \vdash v : B; \alpha : A, \Delta}$	$\text{cut}_{\mathcal{R}} \frac{x : A, \Gamma; e : B \vdash \Delta}{\Gamma \vdash \mu\alpha.\langle v e \rangle : A; \Delta}$
$\text{imply}_{\mathcal{L}} \frac{\Gamma \vdash v : A; \Delta \quad \Gamma; e : B \vdash \Delta}{\Gamma; v \cdot e : A \Rightarrow B \vdash \Delta}$	$\text{imply}_{\mathcal{R}} \frac{\Gamma, x : A \vdash v : B; \Delta}{\Gamma \vdash \lambda x.v : A \Rightarrow B; \Delta}$
$\text{minus}_{\mathcal{L}} \frac{\Gamma; e : A \vdash \alpha : B, \Delta}{\Gamma; \alpha\lambda'.e : A - B \vdash \Delta}$	$\text{minus}_{\mathcal{R}} \frac{\Gamma; e : B \vdash \Delta \quad \Gamma \vdash v : A; \Delta}{\Gamma \vdash e \cdot v : A - B; \Delta}$
$\text{conj}_{\mathcal{L}} \frac{\Gamma, x : A, y : B \vdash v : C; \Delta}{\Gamma; \text{proj}[x, y, \langle v e \rangle] : A \wedge B \vdash \Delta}$	$\text{conj}_{\mathcal{R}} \frac{\Gamma \vdash v : A; \Delta \quad \Gamma \vdash v' : B; \Delta}{\Gamma \vdash (v, v') : A \wedge B; \Delta}$
$\text{disj}_{\mathcal{L}} \frac{\Gamma; e : A \vdash \Delta \quad \Gamma; e' : B \vdash \Delta}{\Gamma; [e, e'] : A \vee B \vdash \Delta}$	$\text{disj}_{1\mathcal{R}} \frac{\Gamma \vdash v : A; \Delta}{\Gamma \vdash \text{inj}_l v : A \vee B; \Delta}$
$\text{disj}_{2\mathcal{R}} \frac{\Gamma \vdash v : B; \Delta}{\Gamma \vdash \text{inj}_r v : A \vee B; \Delta}$	$\text{neg}_{\mathcal{L}} \frac{\Gamma \vdash v : A; \Delta}{\Gamma; \tilde{\mu}x.\langle x v \cdot \perp \rangle : \neg A \vdash \Delta}$
$\text{forall}_{\mathcal{L}} \frac{\Gamma; e : B[x \leftarrow t] \vdash \Delta}{\Gamma; t \cdot e : \forall(x : A)B \vdash \Delta}$	$\text{neg}_{\mathcal{R}} \frac{\Gamma; e : A \vdash \Delta}{\Gamma \vdash \mu\alpha.\langle \lambda y.\mu z\langle y e \rangle \alpha \rangle : \neg A; \Delta}$
$\text{exists}_{\mathcal{L}} \frac{\Gamma; e : B \vdash \Delta}{\Gamma; \text{proj}[\alpha, \beta, \langle \beta e \rangle] : \exists(\alpha : A)B \vdash \Delta}$	$\text{forall}_{\mathcal{R}} \frac{\Gamma \vdash v : B; \Delta}{\Gamma \vdash \lambda x.v : \forall(x : A)B; \Delta}$
	$\text{exists}_{\mathcal{R}} \frac{\Gamma \vdash v : B[x \leftarrow t]; \Delta}{\Gamma \vdash (t, v) : \exists(x : A)B; \Delta}$

FIGURE 2. Typing rules for $\bar{\lambda}\mu\tilde{\mu}$

The equivalent of the contraction and weakening rules are trivially derived. The interested user can refer to [3]

2. EXAMPLE: 2 IS EVEN!

```
% ./fsp -ascii
```

```
> Welcome! This is FSP version 0.0.1.  
  To get help type "help" followed by a dot.  
  Current logic: intuitionistic sequent calculus.
```

```
fsp < minimal. lj.
```

```
> Current logic: minimal intuitionistic sequent calculus.
```

```
fsp < declare N: type.
```

```
> N defined.
```

```
fsp < declare 0: N.
```

```
> 0 defined.
```

```
fsp < declare S: N -> N.
```

```
> S defined.
```

```
fsp < declare Even: N -> bool.
```

```
> Even defined.
```

```
fsp < declare Odd: N -> bool.
```

```
> Odd defined.
```

```
fsp < declare EO: (Even [0]).
```

```
> EO defined.
```

```
fsp < declare OS: (forall n:N, Even [n] -> Odd [S n]).
```

```

> OS defined.

fsp < declare ES: (forall n:N, Odd [n] -> Even [S n]).

> ES defined.

fsp < theorem even_2: (Even [S (S 0)]).

> Proof term:
  ;:thesis:Even (S (S 0)).<?1||thesis>
  Natural language:
  we need to prove Even (S (S 0))
.... (1) <=====
      done

      1 goal yet to prove!

      |----- 1
      *:Even (S (S 0))

fsp < focus ES th.

> Proof term:
  ;:thesis:Even (S (S 0)).<;:th:Even (S (S 0)).<ES||?1.2>||thesis>
  Natural language:
  we need to prove Even (S (S 0))
    we need to prove Even (S (S 0))
      by ES
      ... (1.2) <=====
  done

  1 goal yet to prove!

  *:forall n:N,Odd n->Even (S n)
  |----- 1.2
  th:Even (S (S 0))

fsp < elim [S 0].

> Proof term:
  ;:thesis:Even (S (S 0)).<;:th:Even (S (S 0)).<ES||S 0*?1.2.1>||thesis>
  Natural language:

```


2 goals yet to prove!

```
*:Even 0->Odd (S 0)
|----- 1.2.1.1.2.1
th:Odd (S 0)
```

fsp < elim; [focus EO th; axiom | axiom].

> Closed a branch.

Proof term:

```
::thesis:Even (S (S 0)).<;:th:Even (S (S 0)).<ES||S 0*;;:th:Odd (S 0).<OS||0*;;:th:Even
```

Natural language:

```
we need to prove Even (S (S 0))
  we need to prove Even (S (S 0))
    by ES
    and we need to prove Odd (S 0)
      by OS
      and we need to prove Even 0
        by EO
        done
      done
    ... (1.2.1.2) <=====
done
```

1 goal yet to prove!

```
*:Even (S (S 0))
|----- 1.2.1.2
th:Even (S (S 0))
```

fsp < axiom.

> Closed the last branch:

Proof completed!

Proof term:

```
::thesis:Even (S (S 0)).<;:th:Even (S (S 0)).<ES||S 0*;;:th:Odd (S 0).<OS||0*;;:th:Even
```

Natural language:

```
we need to prove Even (S (S 0))
  we need to prove Even (S (S 0))
    by ES
    and we need to prove Odd (S 0)
```

```
    by OS
    and we need to prove Even 0
      by EO
    done
  done
done
done
```

```
fsp < qed.
```

```
> even_2 defined.
```

```
fsp < checkout proof term coq.
```

```
> Firing up Coq for confirmation.
```

```
fsp < checkout proof term coq.
```

```
> Firing up Coq for confirmation.
```

```
fsp < export natural language.
```

```
> File script.nl written. Enjoy the reading.
```

```
fsp < quit.
```

```
> Out.
```

REFERENCES

1. Pierre-Louis Curien and Hugo Herbelin, *The duality of computation*, ICFP'00: Proceedings of 5th ACM SIGPLAN International Conference on Functional Programming (New York, USA), SIGPLAN Notices, vol. 35(9), ACM Press, 2000.
2. Christian Urban and Gavin Bierman, *Strong normalisation of cut-elimination in classical logic*, TLCA '99: Proceedings of the 4th International Conference on Typed Lambda Calculi and Applications (London, UK), Springer-Verlag, 1999, pp. 365–380.
3. Philip Wadler, *Call-by-value is dual to call-by-name*, ACM SIGPLAN Notices **38** (2003), no. 9, 189–201.