Comp 311
Functional Programming

Eric Allen, Two Sigma Investments
Robert "Corky" Cartwright, Rice University
Sağnak Taşırlar, Two Sigma Investments

Tactical Theorem Proving

Tactical Theorem Proving

- The approach of starting with the end goal in a proof and working backwards has applicability beyond type checking
- In the general case, there might be more than one rule that could apply
- Thus, we cannot expect to prove every theorem simply by working backwards from the goal

A Tactical Theorem Prover

- A tactical proof assistant allows us to interactively solve a proof by working backwards from goals
- We start with a single goal
- Every time we apply a tactic, we might solve some goals but also generate new subgoals

Tactical Theorem Proving

- We define a *tactic* to be a function that takes a collection of one or more goals and returns a pair consisting of:
  - A partial proof of one of the goals
  - A collection of goals

Tactical Theorem Proving

- A partial proof of a goal is a function that:
  - Takes one or more sequents as arguments and
  - Returns the goal sequent by applying only inference rule functions to its arguments
    - Applying this function effectively checks the validity of the proof

Tactical Theorem Proving

- The collection of goals returned by a tactic might include:
    - Some of the goals passed to the tactic
    - Some new goals produced by the tactic

The Type of a Tactic

- We define the type `ProofState` as consisting of a set of goal sequents:

```
type ProofState = List[Sequent]
```

The Type of a Tactic

- We define the type `PartialProof` as a function from a list of `Sequents` to a `Sequent`

```
type PartialProof = List[Sequent] => Sequent
```

The Type of a Tactic

- The PartialProof has a “hole” in it for each sequent in its parameter list
- The sequents to fill these holes must be supplied via their own proofs

```
type PartialProof = List[Sequent] => Sequent
```

The Type of a Tactic

- We could now define tactics to be functions from ProofStates to pairs of PartialProofs with ProofStates:

```
type Tactic = ProofState => (PartialProof, ProofState)
```

The Type of a Tactic

- Equivalently, we can say that the `Tactic` type is a monad

- It can be defined as an application of StateAction:

```
def tactic(state: ProofState => (PartialProof, ProofState)) =
  StateAction[ProofState, PartialProof](state)
```

An Example Tactic for Assumption

```
val assumptionTactic = tactic {
  (proofState: ProofState) => {
    proofState match {
      case ((gamma :- a) :: goals) =>
        def partialProof(proofs: List[Sequent]) = {
          assumption(gamma :- a)
        }
        (partialProof, goals)
      case _ => throw TacticError(…)
    }
  }
}
```

An Example Tactic for Assumption

```
val andTactic = tactic {
  (proofState: ProofState) => {
    proofState match {
      case ((gamma :- (a /\ b)) :: goals) =>
        def partialProof(proofs: List[Sequent]) = {
          proofs match {
            case proofA :: proofB :: Nil =>
              andIntro(proofA, proofB)
            case _ => throw ProofError(…)
          }
        }
        (partialProof, (gamma :- a) :: (gamma :- b) :: goals)
      case _ => throw TacticError(…)
    }
  }
}
```

An Example Manual Proof Session Using Tactics

```
val seq = (p + empty :- p)
val proof = assumptionTactic(List(seq))
proof._1(Nil)
```

An Example Proof Session Using Map

```
val seq = (p + empty :- p)

assumptionTactic.map(partialProof => partialProof(Nil)) {
    List(seq)
}
```

An Example Proof Session Using For Expressions

```
val seq = (p + empty :- p)

val strategy = for {
  partialProof <- assumptionTactic
} yield partialProof(Nil)

strategy(seq)
```

An Example Manual Proof Session Using Tactics

```
val seq = (p + (q + empty)) :- (p /\ q)
val proofState = List(seq)
val step1 = andTactic(proofState)
val step2 = assumptionTactic(step1._2)
val step3 = assumptionTactic(step2._2)

step1._1(List(step2._1(Nil), step3._1(Nil)))
```

An Example Proof Session Using Map and Flatmap

```
val seq = (p + (q + empty)) :- (p /\ q)

andTactic.flatMap(step1 =>
  assumptionTactic.flatMap(step2 =>
    assumptionTactic.map(step3 =>
      step1(List(step2(Nil),step3(Nil))))))
(List(seq))
```

An Example Proof Session Using For Expressions

```
val seq = (p + (q + empty)) :- (p /\ q)

val strategy = for {
  step1 <- andTactic
  step2 <- assumptionTactic
  step3 <- assumptionTactic
}
yield step1(List(step2(Nil),
                 step3(Nil)))

strategy(List(seq))
```