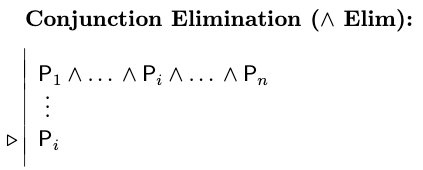
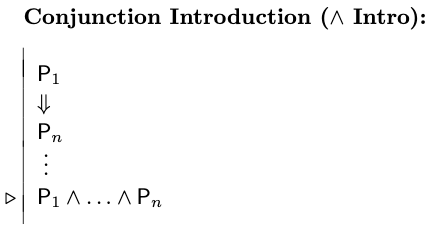
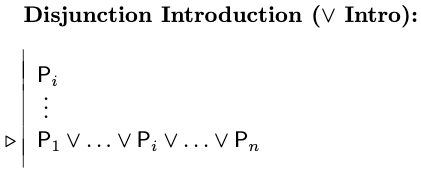
**Ch 6 - Formal Proofs and Boolean Logic**

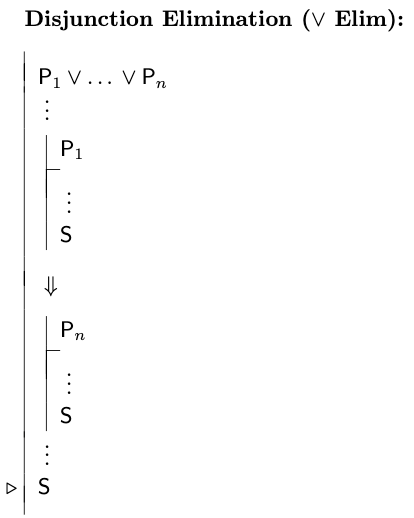
* **deductive system** F: **system of natural deduction**
  + such systems are intended to be models of valid principles for reasoning used in **informal proofs**
* **previous chapter:** informal principles of Boolean reasoning
* **this chapter:** inference rules of F that correspond to the principles
  + formal counterparts of some of the principles

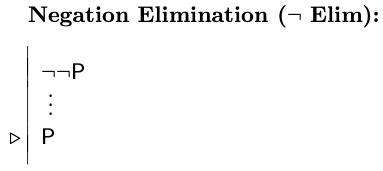




means that each of the constituents P1 to Pn must appear in the proof before you can assert their conjunction

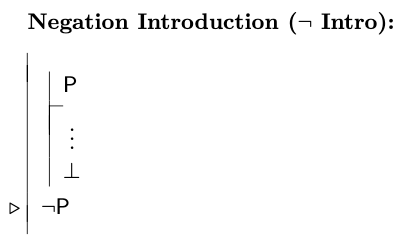






**Negation Introduction:** corresponds to the method of indirect proof or proof by contradiction

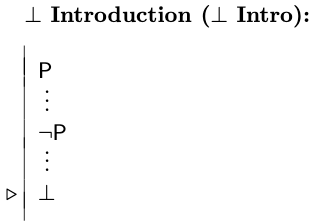
* if we can prove a contradiction ⊥ on the basis of an additional assumption P, then we are entitled to infer ~P from the original premises



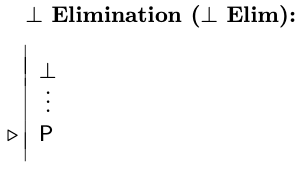
* There are different ways of understanding this rule depending on how we interpret the contradiction symbol ⊥
  + some interpret it as simply shorthand for any contradiction of the form Q & ~Q, but we will treat it as a symbol in its own right to be read “contradiction”
  + it represents an always-false atomic sentence

⊥ **Introduction**

* allows us to obtain the contradiction symbol if we have established an explicit contradiction in the form of some sentence P and its negation ~P
* the only time we will be able to derive ⊥ in our main proof (as opposed to a subproof) is when our premises are inconsistent
* a formal proof of inconsistency is one that derives ⊥ at the main leel of the proof



* note that this rule, as presented, allows us to derive ⊥ from the blatant contradiction of form P & ~P
* we might come across an inconsistency of some other form such as being truth-table contradictory, for example ~(A | ~A)
  + it turns out that if we can prove any TT-contradictory sentence or sentences, the rules we’ve shown above allow us to prove ⊥; it may take some effort, but it is possible
* there are also other forms of contradiction besides TT-contradictions
* **built-in mechanisms in Fitch**
  + enter ⊥, cite the sentences, choose Taut Con; if it checks out, it means we can definitely prove ⊥ using just the introduction and elimination rules for |, &, ~, and ⊥
  + if we suspect we have sentences whose inconsistency results from Boolean connectives plus the identity predicate, we can check this using the FO Con mechanism, since it understands the meaning of =
    - if ⊥ checks out (and the cited sentences do not contain quantifiers), then we should be able to prove ⊥ with the known rules as well
* **the only time we may arrive at a contraction but not be able to prove** ⊥ **using the rules of F is if the inconsistency depends on the meanings of predicates other than identity**
  + Ana Con understands predicates in the block language (though it excludes Adjoins and Betweens); with it we can derive ⊥ that can’t be derived in F
  + Cube(b) & Tet(b)
    - this contradiction can’t be established in F



* you can prove any sentence from a contradiction even without this rule, however. It just takes longer.

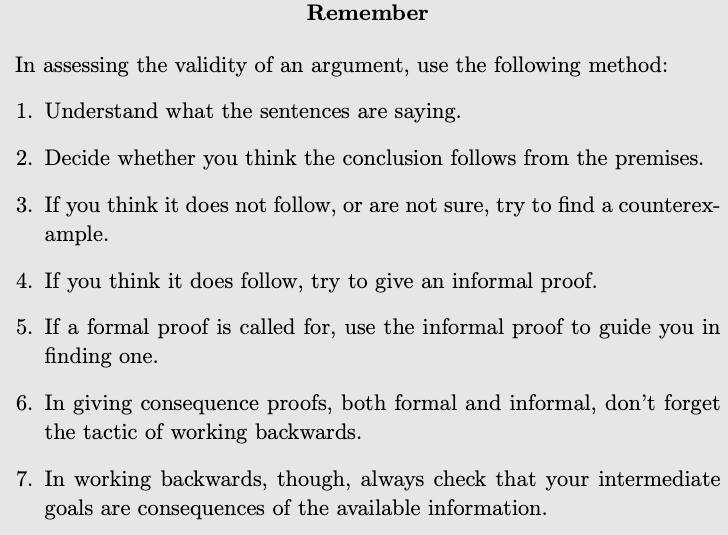


**Use of Subproofs**

* subproofs are the characteristic feature of Fitch-style deductive systems

**Strategy and Tactics for Finding Proofs**

* always keep in mind what the sentences in your proof mean
  + first step in constructing a proof is to convince yourself that the claim made by the conclusion is a consequence of the premises
  + in this process of recognizing the argument’s validity, you will often get some idea how to prove it
* next, try giving an informal proof
* if it gets hard try working backwards
  + look at the conclusion and see what additional sentence or sentences would allow you to infer that conclusion
  + insert these steps in your proof, not worrying how they will be justified, and cite the in support of your goal sentence
  + take these intermediate steps as new goals and see if you can prove them



**Proofs without Premises**

* a proof without any premises shows that its conclusion is a logical truth