```
theory ivt
imports Complex-Main
begin

theorem ivt:
  fixes f :: real ⇒ real and a b :: real
  assumes ctsf: continuous-on {a..b} f
      and a < b and fa1: f a < 0 and fa2: f b > 0
  shows ∃ x ∈ {a..b}. f x = 0
proof −
  have a ∈ {a..b} using assms by auto
  have [simp]: a ∈ {x. x ∈ {a..b} ∧ f x < 0} using assms by auto
  have bdd-above {a..b} by auto
  have bdd-above {x. x ∈ {a..b} ∧ f x < 0} by auto
  define y where y = Sup {x. x ∈ {a..b} ∧ f x < 0}
  have y ∈ {a..b}
    by (smt ‹a ∈ {x ∈ {a..b}. f x < 0}› ‹bdd-above {a..b}›
          ‹bdd-above {x ∈ {a..b}. f x < 0}› atLeastAtMost-iff
          cSup-atLeastAtMost cSup-mono empty-iff mem-Collect-eq y-def)
  have ¬ f y < 0
  proof
    assume f y < 0
    hence − f y > 0 by linarith
    with ctsf have ∃ δ. δ > 0 ∧
        (∀ x. x ∈ {a..b} ∧ abs (x − y) < δ ⟶ abs (f x − f y) < − f y)
      apply (simp add: continuous-on-def tendsto-iff dist-real-def
              eventually-at)
      by (metis ‹0 < − f y› ‹y ∈ {a..b}› abs-zero atLeastAtMost-iff
            cancel-comm-monoid-add-class.diff-cancel)
    then obtain δ where δ > 0 and
        h: ⋀x. x ∈ {a..b} ⟹ abs (x − y) < δ ⟹ abs (f x − f y) < − f y
      by auto
    let ?δ' = min (δ / 2) (b − y)
    let ?y' = y + ?δ'
    from ‹y ∈ {a..b}› ‹f y < 0› ‹f b > 0› have y < b
      by (smt atLeastAtMost-iff)
    with ‹δ > 0› have ?y' > y by linarith
    from ‹y ∈ {a..b}› ‹δ > 0› have ?y' ∈ {a..b} and abs (?y' − y) < δ
      by auto
    hence abs (f ?y' − f y) < − f y by (auto intro: h)
    with ‹− f y > 0› have f ?y' < 0 by auto
    with ‹?y' ∈ {a..b}› have y ≥ ?y'
      by − (subst (3) y-def, rule cSup-upper, auto)
    thus False
      using ‹0 < δ› ‹y < b› by linarith
  qed
  moreover have ¬ f y > 0
```

```
  proof
    assume f y > 0
    with ctsf have ∃ δ. δ > 0 ∧
        (∀ x. x ∈ {a..b} ∧ abs (x − y) < δ ⟶ abs (f x − f y) < f y)
      apply (simp add: continuous-on-def tendsto-iff dist-real-def
              eventually-at)
      by (metis ‹0 < f y› ‹y ∈ {a..b}› abs-zero atLeastAtMost-iff
            cancel-comm-monoid-add-class.diff-cancel)
    then obtain δ where δ > 0 and
        h: ⋀x. x ∈ {a..b} ⟹ abs (x − y) < δ ⟹ abs (f x − f y) < f y
      by auto
    from ‹y ∈ {a..b}› ‹f y > 0› ‹f a < 0› have y > a
      by (smt atLeastAtMost-iff)
    let ?δ′ = min (δ / 2) (y − a)
    let ?y′ = y − ?δ′
    from ‹0 < δ› ‹a < y› ‹a ∈ {x. x ∈ {a..b} ∧ f x < 0}›
        have ∃ y′′ ∈ {x. x ∈ {a..b} ∧ f x < 0}. ?y′ < y′′
      by (subst y-def, subst less-cSup-iff [symmetric], auto)
    then obtain y′′ where y′′ > ?y′
          and hy′′: y′′ ∈ {x. x ∈ {a..b} ∧ f x < 0} by auto
    hence y′′ ≤ y
      by (metis ‹bdd-above {x ∈ {a..b}. f x < 0}› cSup-upper y-def)
    with ‹y′′ > ?y′› ‹y ∈ {a..b}› ‹δ > 0› hy′′ have abs (f y′′ − f y) < f y
      by (auto intro: h)
    hence f y′′ > 0 by auto
    with hy′′ show False by auto
  qed
  ultimately have f y = 0 by linarith
  with ‹y ∈ {a..b}› show ?thesis by blast
qed

theorem ivt-original-beginning:
  fixes f :: real ⇒ real and a b :: real
  assumes ctsf: continuous-on {a..b} f
      and a < b and fa1: f a < 0 and fa2: f b > 0
  shows True
proof −
  define y where y = Sup {x. x ∈ {a..b} ∧ f x < 0}
  have y ∈ {a..b}
  proof −
    have y ≤ Sup {a..b}
      apply (subst y-def)
      apply (rule cSup-mono)
      using assms apply auto
      apply (rule-tac x = a in exI)
      by auto
    hence y ≤ b using ‹a < b› by auto
    moreover have a ≤ y
      apply (auto simp add: y-def)
```

```
      apply (rule cSup-upper)
      using assms by auto
    ultimately show ?thesis by auto
  qed
  show ?thesis by auto
qed

end
```