Universität Innsbruck **SS 2008**
Institut für Informatik **Exercise Sheet 5**
Prof. Dr. Clemens Ballarin **Due 12 June 2008**

Automatic Deduction — Introduction to Isabelle
LVA 703522

1 Two Grammars

The most natural definition of valid sequences of parentheses is this:

$$S \quad \rightarrow \quad \epsilon \quad | \quad '(' \; S \; ')' \quad | \quad S \; S$$

where ϵ is the empty word.
A second, somewhat unusual grammar is the following one:

$$T \quad \rightarrow \quad \epsilon \quad | \quad T \; '(' \; T \; ')'$$

▷ Make the definitions of sets `S` and `T` in Isabelle and give structured Isar proofs leading to the theorem `S = T`.
Hint: use lists over a specific type to repesent words.

The alphabet:

```
datatype alpha = A | B
```

Standard grammar:

```
inductive_set S :: "alpha list set"
  where
    S1: "[] ∈ S"
  | S2: "w ∈ S ⟹ [A] @ w @ [B] ∈ S"
  | S3: "v ∈ S ⟹ w ∈ S ⟹ v @ w ∈ S"
```

Nonstandard grammar:

```
inductive_set T :: "alpha list set"
  where
    T1: "[] ∈ T"
  | T23: "v ∈ T ⟹ w ∈ T ⟹ v @ ([A] @ w @ [B]) ∈ T"
```

Equivalence proof

```
lemma T_in_S: "w ∈ T ⟹ w ∈ S"
proof (induct set: T)
  case T1
  show "[] ∈ S" by (rule S1)
next
```

```
  case (T23 v w)
  have "v ∈ S" .
  moreover
  {
    have "w ∈ S" .
    then have "[A] @ w @ [B] ∈ S" by (rule S2)
  }
  ultimately
  show "v @ ([A] @ w @ [B]) ∈ S" by (rule S3)
qed

lemma T2: "w ∈ T ⟹ [A] @ w @ [B] ∈ T"
proof -
  have "[] ∈ T" by (rule T1)
  moreover assume "w ∈ T"
  ultimately have "[] @ ([A] @ w @ [B]) ∈ T" by (rule T23)
  then show ?thesis by simp
qed

lemma T3:
  assumes u: "u ∈ T"
    and v: "v ∈ T"
  shows "u @ v ∈ T"
  using v
proof induct
  case T1
  from u show "u @ [] ∈ T" by simp
next
  case (T23 v w)
  have "u @ v ∈ T" .
  moreover have "w ∈ T" .
  ultimately have "(u @ v) @ ([A] @ w @ [B]) ∈ T" by (rule T.T23)
  then show "u @ (v @ [A] @ w @ [B]) ∈ T" by simp
qed

lemma S_in_T: "w ∈ S ⟹ w ∈ T"
proof (induct set: S)
  case S1
  show "[] ∈ T" by (rule T1)
next
  case (S2 w)
  have "w ∈ T" .
  then show "[A] @ w @ [B] ∈ T" by (rule T2)
next
  case (S3 v w)
  have "v ∈ T" and "w ∈ T" .
  then show "v @ w ∈ T" by (rule T3)
qed
```

```
theorem "S = T"
  using S_in_T T_in_S by blast
```

2 Polynomial sums

▷ Produce structured proofs of the following theorems, using induction and calculational reasoning in Isar.

Note that the given tactic scripts are of limited use in reconstructing structured proofs; nevertheless the hints of automated steps below can be re-used to finish sub-problems. The ∑ symbol can be entered as "\<Sum>"; note that numerals in Isabelle/HOL are polymorphic.

```
theorem — problem
  fixes n :: nat
  shows "2 * (∑i=0..n. i) = n * (n + 1)"
  by (induct n) simp_all

theorem — solution
  fixes n :: nat
  shows "2 * (∑i=0..n. i) = n * (n + 1)"
proof (induct n)
  case 0
  have "2 * (∑i=0..0. i) = (0::nat)" by simp
  also have "(0::nat) = 0 * (0 + 1)" by simp
  finally show ?case .
next
  case (Suc n)
  have "2 * (∑i=0..Suc n. i) = 2 * (∑i=0..n. i) + 2 * (n + 1)" by simp
  also have "2 * (∑i=0..n. i) = n * (n + 1)" by (rule Suc.hyps)
  also have "n * (n + 1) + 2 * (n + 1) = Suc n * (Suc n + 1)" by simp
  finally show ?case .
qed

theorem — problem
  fixes n :: nat
  shows "(∑i=0..<n. 2 * i + 1) = n²"
  by (induct n) (simp_all add: power_eq_if nat_distrib)

theorem — solution
  fixes n :: nat
  shows "(∑i=0..<n. 2 * i + 1) = n²"
proof (induct n)
  case 0
  have "(∑i=0..<0. 2 * i + 1) = (0::nat)" by simp
  also have "(0::nat) = 0²" by simp
  finally show ?case .
next
  case (Suc n)
```

```
  have "(∑i=0..<Suc n. 2 * i + 1) = (∑i=0..<n. 2 * i + 1) + 2 * n +
1"
    by simp
  also have "(∑i=0..<n. 2 * i + 1) = n²"
    by (rule Suc.hyps)
  also have "n^2 + 2 * n + 1 = (Suc n)²"
    by (simp add: power_eq_if nat_distrib)
  finally show ?case .
qed

theorem — problem
  fixes n :: nat
  shows "(∑i=0..<n. 2^i) = 2^n - (1::nat)"
  by (induct n) (simp_all split: nat_diff_split)

theorem — solution
  fixes n :: nat
  shows "(∑i=0..<n. 2^i) = 2^n - (1::nat)"
proof (induct n)
  case 0
  have "(∑i=0..<0. 2^i) = (0::nat)" by simp
  also have "(0::nat) = 2^0 - (1::nat)" by simp
  finally show ?case .
next
  case (Suc n)
  have "(∑i=0..<Suc n. 2^i) = (∑i=0..<n. 2^i) + 2^n"
    by simp
  also have "(∑i=0..<n. 2^i) = 2^n - (1::nat)"
    by (rule Suc.hyps)
  also have "(2^n - (1::nat)) + 2^n = 2^(Suc n) - (1::nat)"
    by (simp split: nat_diff_split)
  finally show ?case .
qed

theorem — problem
  fixes n :: nat
  shows "2 * (∑i=0..<n. 3^i) = 3^n - (1::nat)"
  by (induct n) (simp_all add: nat_distrib)

theorem — solution
  fixes n :: nat
  shows "2 * (∑i=0..<n. 3^i) = 3^n - (1::nat)"
proof (induct n)
  case 0
  have "2 * (∑i=0..<0. 3^i) = (0::nat)" by simp
  also have "(0::nat) = 3^0 - (1::nat)" by simp
  finally show ?case .
next
  case (Suc n)
```

```
  have "(2::nat) * (∑i=0..<Suc n. 3^i) = 2 * (∑i=0..<n. 3^i) + 2 *
3^n"
    by (simp add: nat_distrib)
  also have "2 * (∑i=0..<n. 3^i) = 3^n - (1::nat)"
    by (rule Suc.hyps)
  also have "(3^n - 1) + 2 * 3^n = 3^(Suc n) - (1::nat)"
    by simp
  finally show ?case .
qed

theorem — problem
  fixes n :: nat
  assumes k: "0 < k"
  shows "(k - 1) * (∑i=0..<n. k^i) = k^n - (1::nat)"
  by (induct n) (insert k, simp_all add: nat_distrib)

theorem — solution
  fixes n :: nat
  assumes k: "0 < k"
  shows "(k - 1) * (∑i=0..<n. k^i) = k^n - (1::nat)"
proof (induct n)
  case 0
  have "(k - 1) * (∑i=0..<0. k^i) = (0::nat)" by simp
  also have "(0::nat) = k^0 - (1::nat)" by simp
  finally show ?case .
next
  case (Suc n)
  have "(k - 1) * (∑i=0..<Suc n. k^i) =
      (k - 1) * (∑i=0..<n. k^i) + (k - 1) * k^n"
    using k by (simp add: nat_distrib)
  also have "(k - 1) * (∑i=0..<n. k^i) = k^n - (1::nat)"
    by (rule Suc.hyps)
  also have "(k^n - 1) + (k - 1) * k^n = k^(Suc n) - (1::nat)"
    using k by (simp add: nat_distrib)
  finally show ?case .
qed
```