Static Semantics

- **Static semantics** = strictures on the formation of expressions.
  - When is an expression well-formed?
  - Eg, fun f(x:bool):int = not(x)+1 end is ill-formed, even though it is syntactically correct!
- The static semantics is inductively-defined by a set of **typing rules**.

Static Semantics

- **Typing relation**: G |- e : t
  - G is a **typing context** assigning types to the free variables of e;
    - G : Variables → Types
  - e is an expression whose free variables are governed by G;
  - t is a type.
- "e has type t in context G"

Typing Rules

- Typing relation is inductively-defined by a set of **typing rules**.
  - G |- x : t, if G(x) = t
  - G |- n : int
  - G |- true : bool, G |- false : bool
  - G |- o(e1,…,en) : t, if G |- e1 : t1, and …, and G |- en : tn, and o has arg type t1, …, tn and result type t.

Typing Rules, cont'd

- G |- if e then e1 else e2 fi : t, if G |- e : bool, G |- e1 : t, G |- e2 : t
  NB: same type for "then" and "else"!
- G |- apply(e1,e2) : t if G |- e1 : t2 -> t and G |- e2 : t2
  NB: must exist some t2!
- G |- fun f(x:t):t' is e' end : t->t' , if G[f:t->t'][x:t] |- e' : t'
  provided f, x ∉ dom(G)

Typing Rules, cont'd

- Notation in the notes:

$$\frac{\Gamma \mid\!- e : \texttt{bool} \quad \Gamma \mid\!- e_1 : \tau \quad \Gamma \mid\!- e_2 : \tau}{\Gamma \mid\!- \texttt{if } e \texttt{ then } e_1 \texttt{else } e_2 \texttt{ fi} : \tau}$$

Induction on Typing

- Since the typing relation is inductively-defined, we may prove properties of it by induction on the typing rules.
- To prove "if G |- e : t, then … G … e … t …", it suffices to show that "… G … e … t …" is closed under the typing rules.

Induction on Typing

- This means that we must show:
  - … G … x … G(x) … (since G |- x : G(x))
  - … G … n … int … (since G |- n : int)
  - if … G … e1 … t2->t … and … G … e2 … t2 …, then … G … apply(e1, e2) … t …
  - etc.

Properties of Typing

- **Syntax-directedness**: there is one rule for each form of expression.
  - Proof is by ind. on typing.
- **Weakening**: "extra" variables don't affect typing: if G |- e : t and G' ⊇ G, then G' |- e : t.
  - Proof is by induction on G |- e : t.

Properties of Typing

- **Substitution**: if G |- e : t and G[x:t] |- e' : t', then G |- [e/x]e' : t'.
  - Proof is by induction on the derivation of G[x:t] |- e' : t'.

Proof of Substitution (Sketch)

- Suppose that G |- e : t. We prove by induction on the typing rules that if G[x:t] |- e' : t', then G |- [e/x]e' : t'.
- It suffices to show that this implication is closed under the typing rules.

Proof of Substitution

- If G[x:t] |- y : t', then G |- [e/x]y:t'.
  - Case split on whether x=y or x <> y.
  - If x=y, then [e/x]y = e and t'=G[x:t](x)=t. But G |- e : t by assumption.
  - If x<>y, then [e/x]y = y. Since G[x:t] |- y : t', then t'=G(y), and hence G |- y : t'.

Proof of Substitution

- If G[x:t] |- e' : t', then G |- [e/x]e' : t', where t'=t1->t2 and e' = fun f(y:t1):t2 is e'' end.
- We may assume that x<>f, x<>y, and x,f not in FV(e) (by suitable renaming).
- Inductive assumption: if G[x:t][f:t1->t2][y:t1] |- e'' : t2, then G[f:t1->t2][y:t1] |- [e/x]e'' : t2.

Proof of Substitution

- [e/x]e' = fun f(y:t1):t2 is [e/x]e'' end.
- To show G |- [e/x]e' : t', it suffices to show that G[f:t1->t2][y:t1] |- [e/x]e'' : t2.
- But this follows immediately from the inductive assumption!
- (The other cases are similar.)

Static Semantics: Summary

- The static semantics of MinML is inductively defined by a set of typing rules.
  - Define context-sensitive constraints on formation of expressions.
- Properties of the static semantics may be proved by induction on typing.