Artificial Intelligence
Chapter 16
Resolution in the Predicate Calculus

Biointelligence Lab
School of Computer Sci. & Eng.
Seoul National University

Outline

- Unification
- Predicate-Calculus Resolution
- Completeness and Soundness
- Converting Arbitrary wffs to Clause Form
- Using Resolution to Prove Theorems
- Answer Extraction
- The Equality Predicate
- Additional Readings and Discussion

16.1 Unification

- Assumptions
  - Universal quantifications for all variables.
  - *Clause form*.
  - $(\forall \xi_1, \cdots, \xi_n)(\lambda_1 \vee \cdots \vee \lambda_k) \Rightarrow \lambda_1 \vee \cdots \vee \lambda_k$ (literal: λ_1)
- If two clauses have matching but complementary literals, it is possible to resolve them
  - Example: $P(f(y), A) \vee Q(B, C)$, $\neg P(x, A) \vee R(x, C) \vee S(A, B)$

$$\neg P(x, A) \vee R(x, C) \vee S(A, B) \xrightarrow{f(y)/x} \neg P(f(y), A) \vee R(f(y), C) \vee S(A, B)$$

$$R(f(y), C) \vee S(A, B) \vee Q(B, C)$$

16.1 Unification (Cont'd)

- *Unification*: A process that computes the appropriate substitution
- *Substitution instance* of an expression is obtained by substituting terms for variables in that expression.
  - Four substitution instances of $P[x, f(y), B]$ are

$$P[z, f(w), B]$$
$$P[x, f(A), B]$$
$$P[g(z), f(A), B]$$
$$P[C, f(A), B]$$

  - The first instance is called an alphabetic variant.
  - The last of the four different variables is called a ground instance (A *ground term* is a term that contains no variables).

16.1 Unification (Cont'd)

- Any substitution can be represented by a set of *ordered* pairs

$$S = \{\tau_1/\xi_1, \tau_2/\xi_2, \ldots, \tau_n/\xi_n\}$$

  - The pair τ_i/ξ_i means that term τ_i is substituted for every occurrence of the variable ξ_i throughout the scope of the substitution.
  - No variables can be replaced by a term containing that same variable.
  - The substitutions used earlier in obtaining the four instances of $P[x, f(y), B]$

$$\begin{aligned} s_1 &= \{z/x, w/y\} \\ s_2 &= \{A/y\} \\ s_3 &= \{g(z)/x, A/y\} \\ s_4 &= \{C/x, A/y\} \end{aligned} \quad \Rightarrow \quad \begin{aligned} &P[z, f(w), B] \\ &P[x, f(A), B] \\ &P[g(z), f(A), B] \\ &P[C, f(A), B] \end{aligned}$$

  - *ws* denotes a substitution instance of an expression *w*, using a substitution *s*. Thus, $P[z, f(w), B] = P[x, f(y), B]s1$
  - The composition *s1* and *s2* is denoted by *s1s2*, which is that substitution obtained by first applying *s2* to the terms of *s1* and then adding any pairs of *s2* having variables not occurring among the variables of *s1*. Thus,

$$\{g(x, y)/z\}\{A/x, B/y, C/w, D/z\} = \{g(A, B)/z, A/x, B/y, C/w\}$$

16.1 Unification (Cont'd)

- $(\omega s1)s2 = \omega(s1s2), \ (s1s2)s3 = s1(s2s3)$
  - Let w be P(x,y), s1 be {f(y)/x}, and s2 be {A/y} then,
    $(\omega s1)s2 = [P(f(y), y)]\{A/y\} = P(f(A), A)$ and
    $\omega(s1\,s2) = [P(x,y)]\{f(A)/x, A/y\} = P(f(A), A)$
  - Substitutions are not, in general, commutative
    $$\omega(s1s2) = P(f(A), A)$$
    $$\omega(s2s1) = [P(x, y)]\{A/y, f(y)/x\} = P(f(y), A)$$
- Unifiable: a set of $\{\omega_i\}$ expressions is unifiable if there exists a substitution s such that $\omega_1 s = \omega_2 s = \omega_3 s = \ldots$
  - $s = \{A/x, B/y\}$ unifies $\{P[x, f(y), B], P[x, f(B), B]\}$, to yield $\{P[A, f(B), B]\}$

16.1 Unification (Cont'd)

- MGU (Most General (or simplest) Unifier) *g* has the property that if *s* is any unifier of $\{\omega_i\}$ yielding $\{\omega_i\}s$, then there exists a substitution s' such that $\{\omega_i\}s = \{\omega_i\}gs'$. Furthermore, the common instance produced by a most general unifier is unique except for alphabetic variants.
- UNIFY
  - Can find the most general unifier of a finite set of unifiable expressions and that report failure when the set cannot be unified.
  - Works on a set of *list-structured* expressions in which each literal and each term is written as a list.
  - Basic to UNIFY is the idea of a *disagreement set*. The disagreement set of a nonempty set W of expressions is obtained by locating the first symbol at which not all the expressions in W have exactly the same symbol, and then extracting from each expression in W the subexpression that begins with the symbol occupying that position.

16.1 Unification (Cont'd)

UNIFY(Γ) (Γ is a set of list-structured expressions.)

1. $k \leftarrow 0, \Gamma_k \leftarrow \Gamma, \sigma_k \leftarrow \epsilon$ (Initialization step; ϵ is the empty substitution.)
2. If Γ_k is a singleton, exit with σ_k, the mgu of Γ. Otherwise, continue.
3. $D_k \leftarrow$ the disagreement set of Γ_k.
4. If there exists elements v_k and t_k in D_k such that v_k is a variable that does not occur in t_k, continue. Otherwise, exit with failure; Γ is not unifiable.
5. $\sigma_{k+1} \leftarrow \sigma_k\{t_k/v_k\}, \Gamma_{k+1} \leftarrow \Gamma_k\{t_k/v_k\}$ (Note that $\Gamma_{k+1} = \Gamma_k\sigma_{k+1}$.)
6. $k \leftarrow k+1$
7. Go to step 2.

$\{P[f(x), y, g(y)], P[f(x), z, g(x)]\}$ $\quad \sigma = z/y$

$\{P[f(x), z, g(z)], P[f(x), z, g(x)]\}$ $\quad \sigma = \{z/y\}\{x/z\} = \{x/y, x/z\}$

$\{P[f(x), x, g(x)], P[f(x), x, g(x)]\}$

16.2 Predicate-Calculus Resolution

- γ_1, γ_2 are two clauses. Atom θ in γ_1 and a literal $\neg\phi$ in γ_2 such that θ and ϕ have a most general unifier, μ, then these two clauses have a resolvent, ρ. The resolvent is obtained by applying the substitution μ to the union of γ_1 and γ_2, leaving out the complementary literals. $\rho = [(\gamma_1 - \{\theta\}) \cup (\gamma_2 - \{\neg\phi\})]\mu$
- Examples:

$$\{P(x), Q(x,y)\}, \{\neg P(A), R(B,z)\} \Rightarrow \{Q(A,y), R(B,z)\}$$

$$\{P(x,x), Q(x), R(x)\}, \{\neg P(A,z), \neg Q(B)\} \Rightarrow \{Q(A), R(A), \neg Q(B)\}, \{P(B,B), R(B), \neg P(A,z)\}$$

16.3 Completeness and Soundness

- Predicate-calculus resolution is sound
  - If ρ is the resolvent of two clauses ϕ and ψ, then $\{\phi, \psi\} \models \rho$
- Completeness of resolution
  - It is impossible to infer by resolution alone all the formulas that are logically entailed by a given set.
  - As in propositional resolution, this difficulty is surmounted by using resolution refutation.

16.4 Converting Arbitrary wffs to Clause Form

1. Eliminate implication signs.
2. Reduce scopes of negation signs.
3. Standardize variables
   - Since variables within the scopes of quantifiers are like "dummy variables", they can be renamed so that each quantifier has its own variable symbol.

$$(\forall x)[\neg P(x) \vee (\exists x)Q(x)] \Rightarrow (\forall x)[\neg P(x) \vee (\exists y)Q(y)]$$

16.4 Converting Arbitrary wffs to Clause Form (Cont'd)

4. Eliminate existential quantifiers.

- *Skolem function, Skolemization:* $(\forall x)[(\exists y)Height(x, y)]$

  $(\forall x)Height[x, h(x)]$

- Replace each occurrence of its existentially quantified variable by a Skolem function whose arguments are those universally quantified variables

- $[(\forall w)Q(w)] \supset (\forall x)\{(\forall y)\{(\exists z)[P(x,y,z) \supset (\forall u)R(x,y,u,z)]\}\}$

  $\Rightarrow [(\forall w)Q(w)] \supset (\forall x)\{(\forall y)[P(x,y,g(x,y)) \supset (\forall u)R(x,y,u,g(x,y))]$

  $(\forall x)\{\neg P(x) \vee \{(\forall y)[\neg P(y) \vee P(f(x,y))] \wedge (\exists w)[Q(x,w) \wedge \neg P(w)]\}\}$

  $\Rightarrow * (\forall x)\{\neg P(x) \vee \{(\forall y)[\neg P(y) \vee P(f(x,y))] \wedge [Q(x,h(x)) \wedge \neg P(h(x))]\}\}$

16.4 Converting Arbitrary wffs to Clause Form (Cont'd)

- Skolem function of no arguments

$$(\exists x)P(x) => P(Sk)$$

- *Skolem form*: To eliminate all of the existentially quantified variables from a wff, the proceding procedure on each subformula is used in turn. Eliminating the existential quantifiers from a set of wffs produces what is called the Skolem form of the set of formulas.
- The skolem form of a wff is not equivalent to the original wff.
  $[P(A) \vee P(B)] \models (\exists x)P(x), \ but \ [P(A) \vee P(B)] \not\models P(Sk)$. What is true is that a set of formulas, $\triangle$ is satisfiable if and only if the Skolem form of $\triangle$ is. Or more usefully for purpose of resolution refutations, $\triangle$ is unsatisfiable if and only if the Skolem form of $\triangle$ is unsatifiable.

16.4 Converting Arbitrary wffs to Clause Form (Cont'd)

5. Convert to prenex form

- At this stage, there are no remaining existential quantifiers, and each universal quantifier has its own variable symbol.
- A wff in prenex form consists of a string of quantifiers called a prefix followed by a quantifier-free formula called a *matrix*. The prenex form fof the example wff marked with an * earlier is

$$(\forall x)(\forall y)\{\neg P(x) \vee \{[\neg P(y) \vee P(f(x,y))] \wedge [Q(x,h(x)) \wedge \neg P(h(x))]\}\}$$

6. Put the matrix in conjunctive normal form

- When the matrix of the preceding example wff is put in conjunctive normal form, it became

$$(\forall x)(\forall y)\{[\neg P(x) \vee \neg P(y) \vee P(f(x,y))] \wedge [\neg P(x) \vee Q(x,h(x))] \wedge [\neg P(x) \vee \neg P(h(x))]\}$$

16.4 Converting Arbitrary wffs to Clause Form (Cont'd)

7. Eliminate universal quantifiers

- Assume that all variables in the matrix are universally quantified.

8. Eliminate $\wedge$ symbols

- The explicit occurrence of $\wedge$ symbols may be eliminated by replacing expressions of the form $(\omega_1 \wedge \omega_2)$ with the set of wffs $\{\omega_1, \omega_2\}$.

$$\neg P(x) \vee \neg P(y) \vee P[f(x, y)]$$
$$\neg P(x) \vee Q[x, h(x)]$$
$$\neg P(x) \vee \neg P[h(x)]$$

16.4 Converting Arbitrary wffs to Clause Form (Cont'd)

9. Rename variables

- Variable symbols may be renamed so that no variable symbol appears in more than one clause .

$$\neg P(x1) \vee \neg P(y) \vee P[f(x1, y)]$$
$$\neg P(x2) \vee Q[x2, h(x2)]$$
$$\neg P(x3) \vee \neg P[h(x3)]$$

16.5 Using Resolution to Prove Theorem

- To prove wff ω from Δ, proceed just as in the propositional calculus.
  1. Negate ω,
  2. Convert this negation to clause form, and
  3. Add it to the clause form of Δ.
  4. Then apply resolution until the empty clause is deduced.

16.5 Using Resolution to Prove Theorem (Cont'd)

- Problem: the package delivery robot. Suppose this robot knows that all of the packages in room 27 are smaller than any of the ones in room 28.

  1. $(\forall x, y)\{Package(x) \wedge Package(y) \wedge Inroom(x,27) \wedge Inroom(y,28)] \supset Smaller(x,y)\}$
  2. $\neg P(x) \vee \neg P(y) \neg I(x,27) \vee I(y,28) \vee S(x,y)$

  - Suppose that the robot knows the following:

  3. P(A)
  4. P(B)
  5. I(A,27)∨I(A,28) // package A is either in room 27 or in room 28 (but not which)
  6. I(B,27) // package B is in room 27
  7. ¬S(B,A) // package B is not smaller than package A.

16.5 Using Resolution to Prove Theorem (Cont'd)

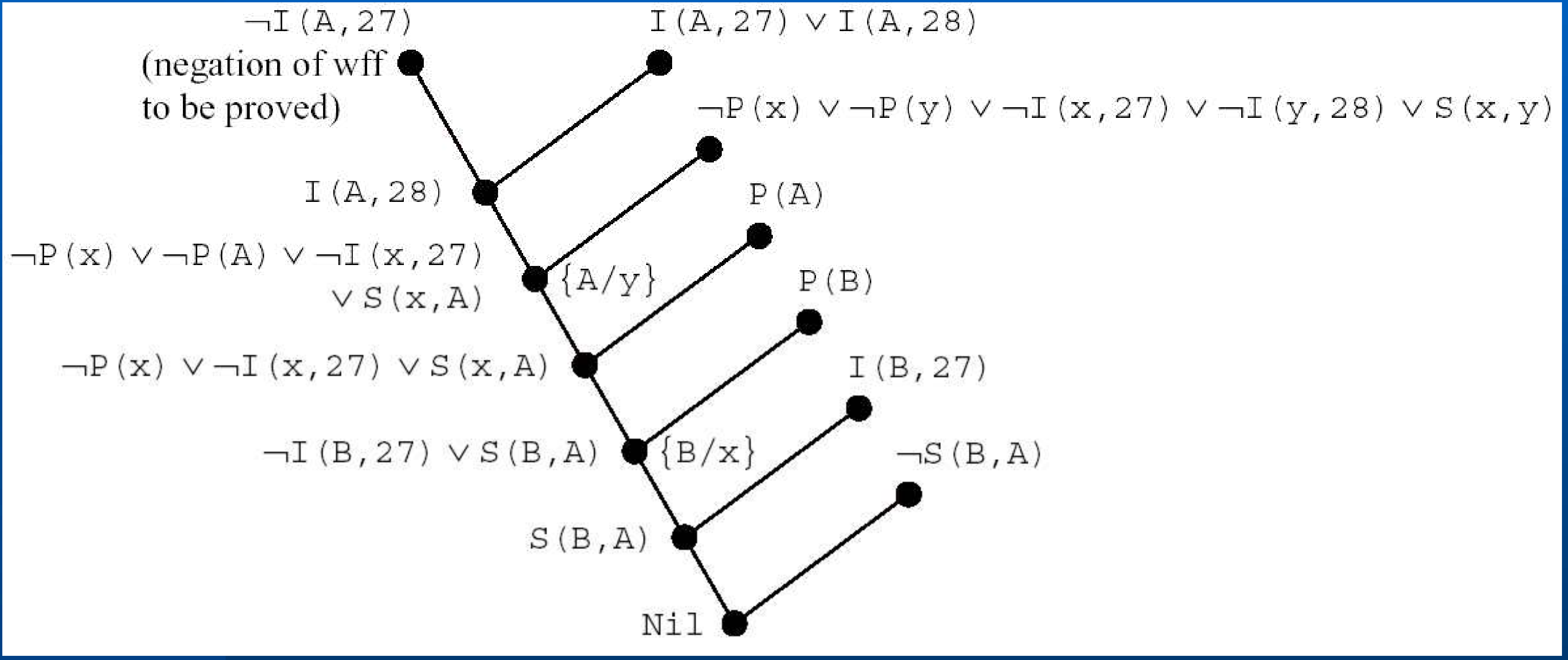
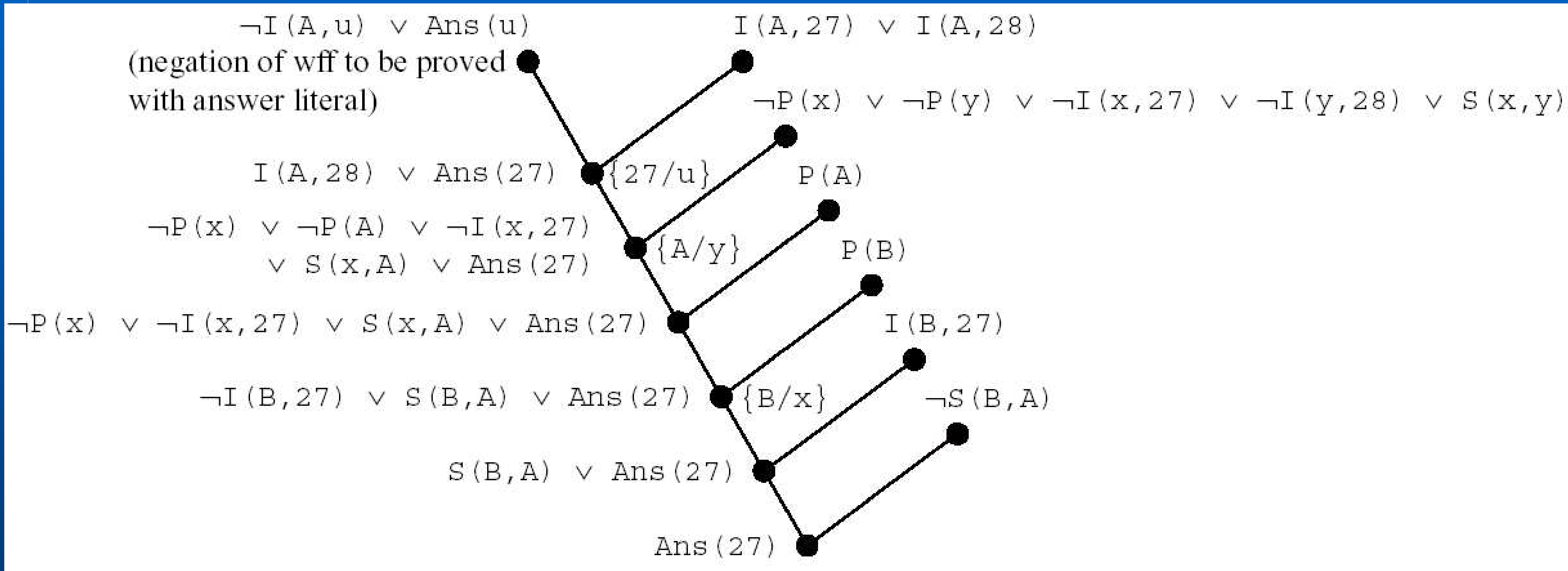


Figure 16.1 A Resolution Refutation

16.6 Answer Extraction



16.7 The Equality Predicate

- The relation constants used in the formulas in a knowledge base usually have intended meanings, but these relations are circumscribed only by the set of models of the knowledge base and not at all by the particular symbols used for relation constants. The result of resolution refutations will be consistent with intended meanings only if the knowledge base suitably constrains the actual relations.
- Equality relation: Equals(A,B) or A=B
  - Reflexive $(\forall x)Equals(x,x)$
  - Symmetric $(\forall x, y)[Equals(x, y) \supset Equals(y, x)]$
  - Transitive $(\forall\ x, y, z)[Equals(x, y) \wedge Equals(y, z) \supset Equals(x, z)]$

16.7 The Equality Predicate

- *Paramodulation*
  - Equality-specific inference rule to be used in combination with resolution in cases where the knowledge base contains the equality predicate .
  - γ_1, γ_2 are two clauses. If $\gamma_1 = \{\lambda(\tau) \bigcup \gamma_1'\}$ and $\gamma_2 = \{Equals(\alpha, \beta) \bigcup \gamma_2'\}$, where τ, α, β are terms, where γ_1` are clauses, and where $\lambda(\tau)$ is a literal containing the term τ, and if τ and α have a most general unifier σ, then infer the binary paramodulant of γ_1 and γ_2: $\pi = \{\lambda\sigma[(\beta\sigma)] \bigcup \gamma_1'\sigma \bigcup \gamma_2'\sigma\}$ where $\lambda\sigma[(\beta\sigma)]$ denotes the result of replacing a single occurrence of $\tau\sigma$ in $\lambda\sigma$ by $\beta\sigma$.
  - Prove P(B) from P(A) and (A=B)
    - For a refutation-style proof, we must deduce the empty clause from the clauses ¬P(B), P(A), and (A=B).
    - Using paramodulation on the last two clauses, $\lambda(\tau)$ is P(A), τ is A, α is A, and β is B. Since A (in the role of τ) and A (in the role of α) unify trivially without a substitution, the binary paramodulation is P(B), which is the result of replacing an occurrence of τ (that is A) with β (that is B). Resolving this paramodulant with ¬P(B) yields the empty clause.

- With a slight extension to the kinds of paramodulants allowed, it can be shown that paramodulation combined with resolution refutation is complete for knowledge bases containing the equality predicate.
- For problem that do not require substituting equals for equals, the power of paramodulation is not needed.
- If an external process is able to return a truth value for an equality predicate, we can replace that predicate by T or F as appropriate. In resolution reputation, clauses containing the literal T can then be eliminated. The literal F in any clause can be eliminated.
- The problem of proving that if a package, say , A, is in a particular room, say, R1, then it cannot be in a different room, say, R2.
  - Statements in knowledge base.
    $(\forall x, y, u, v)[In(x, u) \wedge (u \neq v)] \supset \neg In(x, v)$, $In(A, R1)$
  - In attempting to prove $\neg In(A, R2)$. Converting the first formula into clause form yields $\neg In(x, u) \vee (u=v) \vee \neg In(x, v)$
  - The strategy postpones dealing with equality predicates until they contain only ground terms. Resolving the clause with the negation of the wff to be proved yields $(R2=V) \vee \neg In(A, v)$.
  - Resolving the result with the given wff In(A,R1) yields (R2=R1).
  - If the knowledge base actually contains the wff $\neg(R2=R1)$, then it produces the empty clause, completing the refutation.

16.7 The Equality Predicate (Cont'd)

- If the reasoning involves numbers, it might need an unmanageably large set of wffs. Instead of having all wffs explicitly in the knowledge base, it would be better to provide a routine that would be able to *evaluate* expressions of the form ($\alpha=\beta$) for all (ground) α and β.
- Several other relations (greater than, less than…) and functions (plus, times, divides,…) could be evaluated directly rather than reasoned about with formulas.
- Evaluation of expressions is thus a powerful , efficiency-enhancing tool in automated reasoning systems.

Additional Readings and Discussion

- Some people find the resolution inference rule nonintuitive and prefer so-called natural-deduction methods. These are called “natural” because inference is performed on sentences more or less “as is” without transformations into canonical forms.
- Predicate evaluation is an instance of a more general process called *semantic attachment* in which data structure and programs are associated with elements of the predicate-calculus language. Attached structures and procedures can then be used to evaluate expressions in the language in a way that corresponds to their intended interpretations.