

Validity Checking

Propositional and First-Order Logic (part I: semantic methods)

Slides based on the book: *“Rigorous Software Development: an introduction to program verification”*, by José Bacelar Almeida, Maria João Frade, Jorge Sousa Pinto and Simão Melo Sousa. Springer-Verlag 2011.

1 Validity Checking in Propositional Logic

- General Remarks
- Normal Forms
- Validity/Satisfiability in CNFs
- SAT solvers

2 Validity in First-Order Logic

- General Remarks
- Normal Forms
- Herbrand's Theorem and Semi-Decidability
- Decidable Fragments

3 Validity in First-Order Theories

- Basic Concepts
- Some Theories
- SMT Provers

Propositional Logic

Validity Checking in Propositional Logic

- Given a propositional formula A , there are two obvious decision problems regarding its validity status:

Validity problem (VAL): *Given a formula A , is A valid?*

Satisfiability problem (SAT): *Given a formula A , is A satisfiable?*

- Recall:

- A is valid if $\mathcal{M} \models A$ **for every** model (valuation) \mathcal{M} ;
- A is satisfiable if $\mathcal{M} \models A$ **for some** model \mathcal{M} .
- Hence, A is valid iff $\neg A$ is not satisfiable.

- Two conceivable approaches to settle these problems:

Semantic method – directly using the definition of validity;

Deductive method – exploit *soundness* and *completeness* theorems.

Truth-Tables

- Only propositional symbols used in a formula play a role in its validity.

A	B	$((A \rightarrow B) \rightarrow A) \rightarrow A$
F	F	T
F	T	T
T	F	F
T	T	T

- truth-tables* can be used to decide both VAL and SAT
- 2^n entries (n the number of propositional symbols)
- unfeasible for moderately big formulas
- is it possible to devise better decision procedures?

- The structure of logical validity allows for much better algorithms.
- Strategy for tackling these problems:
 - 1 one first preprocesses the input formula to a restricted syntactic class, preserving the property under evaluation (validity for VAL, and satisfiability for SAT)
 - 2 an efficient method is then applied to check the validity of formulas in this restricted class
- both steps should be kept “reasonably effective” since they are intended to be run in sequence

Complexity Theoretic Considerations

- SAT and VAL are indeed difficult problems
- Both problems play a distinctive role in the hierarchy of complexity classes:
 - SAT is a *NP-complete* problem, i.e. any problem in NP is reducible in polynomial-time to SAT;
 - VAL is a *coNP-complete* problem.
- Hence, it is believed that both SAT and VAL cannot be solved in polynomial-time.

If a polynomial-time algorithm to solve SAT or VAL were ever found, this would settle the $P = NP$ question

Normal Forms

- *Normal forms* are syntactical classes of formulas (i.e. formulas with a restricted “shape”)
- ...that can be considered to be representative of the whole set of formulas.
- The idea is that we associate to a normal form a *normalization procedure* that, for any formula, computes a formula of this restricted class that is *equivalent* (or *equisatisfiable*) with the original.

Negation Normal Form

Definition

A propositional formula A , we say that it is in *negation normal form (NNF)*, if the implication connective is not used in A , and negation is only applied to atomic formulas (propositional symbols or \perp);

- Propositional symbols or their negation are called *literals*
- Hence, a formula in NNF is a formula built up from literals, constants \perp and \top (i.e. $\neg\perp$), disjunctions and conjunctions.
- For every formula A , it is always possible to find an equivalent formula B in NNF (B is called a NNF of A).
- **Normalisation procedure**: repeatedly replace any subformula that is an instance of the left-hand-side of one of the following equivalences by the corresponding right-hand-side.

$$\begin{array}{ll}
 A \rightarrow B \equiv \neg A \vee B & \neg\neg A \equiv A \\
 \neg(A \wedge B) \equiv \neg A \vee \neg B & \neg(A \vee B) \equiv \neg A \wedge \neg B
 \end{array}$$

- Complexity of the normalisation procedure: linear on the size of formula.

Conjunctive/Disjunctive Normal Form

Definition

Given a propositional formula A , we say that it is in:

- **Conjunctive Normal Form (CNF)** if it is a conjunction of disjunctions of literals, i.e. $A = \bigwedge_i \bigvee_j l_{ij}$, for literals l_{ij} ;
- **Disjunctive Normal Form (DNF)** if it is a disjunction of conjunctions of literals, i.e. $A = \bigvee_i \bigwedge_j l_{ij}$, for literals l_{ij} ,

where \perp (resp. \top) is considered to be the empty disjunction (resp. the empty conjunction). The inner conjunctions/disjunctions are called *clauses*.

- CNFs and DNFs are dual concepts. We will restrict attention to CNFs.
- **Normalisation Procedure**: to a formula already in *NNF* apply, the following equivalences (left-to-right):

$$\begin{array}{ll}
 A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C) & (A \wedge B) \vee C \equiv (A \vee C) \wedge (B \vee C) \\
 A \wedge \perp \equiv \perp & \perp \wedge A \equiv \perp & A \wedge \top \equiv A & \top \wedge A \equiv A \\
 A \vee \perp \equiv A & \perp \vee A \equiv A & A \vee \top \equiv \top & \top \vee A \equiv \top
 \end{array}$$

Examples

Let us compute the CNF of $((P \rightarrow Q) \rightarrow P) \rightarrow P$. The first step is to compute its NNF by transforming implications into disjunctions and pushing negations to proposition symbols:

$$\begin{aligned}
 ((P \rightarrow Q) \rightarrow P) \rightarrow P &\equiv \neg((P \rightarrow Q) \rightarrow P) \vee P \\
 &\equiv \neg(\neg(P \rightarrow Q) \vee P) \vee P \\
 &\equiv \neg(\neg(\neg P \vee Q) \vee P) \vee P \\
 &\equiv \neg((P \wedge \neg Q) \vee P) \vee P \\
 &\equiv (\neg(P \wedge \neg Q) \wedge \neg P) \vee P \\
 &\equiv ((\neg P \vee Q) \wedge \neg P) \vee P
 \end{aligned}$$

To reach a CNF, distributivity is then applied to pull the conjunction outside:

$$((\neg P \vee Q) \wedge \neg P) \vee P \equiv (\neg P \vee Q \vee P) \wedge (\neg P \vee P).$$

- The CNF translation has an exponential worst-case running time
 - distributive equivalences duplicate formulas...
 - ...the resulting formula can thus be exponentially bigger than the original formula.
- The following formula illustrates this bad behaviour:

$$\begin{aligned}
 & (P_1 \wedge Q_1) \vee (P_2 \wedge Q_2) \vee \dots \vee (P_n \wedge Q_n) \\
 \equiv & (P_1 \vee (P_2 \wedge Q_2) \vee \dots \vee (P_n \wedge Q_n)) \wedge (Q_1 \vee (P_2 \wedge Q_2) \vee \dots \vee (P_n \wedge Q_n)) \\
 \equiv & \dots \\
 \equiv & (P_1 \vee \dots \vee P_n) \wedge \\
 & (P_1 \vee \dots \vee P_{n-1} \vee Q_n) \wedge \\
 & (P_1 \vee \dots \vee P_{n-2} \vee Q_{n-1} \vee P_n) \wedge \\
 & (P_1 \vee \dots \vee P_{n-2} \vee Q_{n-1} \vee Q_n) \wedge \\
 & \dots \wedge \\
 & (Q_1 \vee \dots \vee Q_n)
 \end{aligned}$$

- The original formula has $2 \cdot n$ literals,
- while the corresponding CNF has 2^n disjunctive clauses, each with n literals.
- Conclusion: in practice, **it is not reasonable to reduce a formula in its equivalent CNF** as part of a VAL procedure.

Definitional CNF

- There are alternative conversions to CNF that avoid this exponential growth.
 - instead of producing an equivalent formula, produce formulas that are *equisatisfiable* with the original formula, i.e.
the resultant formula is satisfiable iff the original formula is
 - These alternative conversions compute what is called the *Definitional CNF* of a formula,
 - ...because they often rely on the introduction of new proposition symbols that act as names for subformulas of the original formula.
- The weaker requirements of *definitional CNF* makes them *suitable for solving SAT* (not VAL).

Example

- The previous example can be handled by associating a new proposition symbol R_i to each conjunctive clause $(P_i \wedge Q_i)$.
- New clauses are added to enforce that new proposition symbols are tied with the original conjunctive clauses: $(\neg R_i \vee P_i)$ and $(\neg R_i \vee Q_i)$.
- The resulting formula is thus:

$$(R_1 \vee \dots \vee R_n) \wedge (\neg R_1 \vee P_1) \wedge (\neg R_1 \vee Q_1) \wedge \dots \wedge (\neg R_n \vee P_n) \wedge (\neg R_n \vee Q_n)$$

- Let \mathcal{M} be any model satisfying this CNF:
 - If $\mathcal{M} \models R_i$ (for some i), then $\mathcal{M} \models P_i$ and $\mathcal{M} \models Q_i$.
 - It is then clear that \mathcal{M} witnesses that the original formula is satisfiable.
- The resultant CNF is not significantly bigger than the original formula (but has more propositional symbols).

Validity in CNFs

- Recall that CNFs are formulas with the following shape (each l_{ij} denotes a literal):

$$(l_{11} \vee l_{12} \vee \dots \vee l_{1k}) \wedge \dots \wedge (l_{n1} \vee l_{n2} \vee \dots \vee l_{nj})$$

- Associativity, commutativity and idempotence of both disjunction and conjunction allow us to treat each CNF as a **set of sets of literals** S

$$S = \{\{l_{11}, l_{12}, \dots, l_{1k}\}, \dots, \{l_{n1}, l_{n2}, \dots, l_{nj}\}\}$$

- An empty inner set (clause) will be identified with \perp , and an empty outer set with \top .
- Simple observations:
 - a CNF is a tautology if and only if all of its clauses are tautologies;
 - If a clause $c \in S$ is a tautology, it can be removed from S without affecting its validity status, i.e. $S \equiv S \setminus \{c\}$;
 - A clause c is a tautology precisely when there exists a proposition symbol P such that $\{P, \neg P\} \subseteq c$. A clause c such that $\{P, \neg P\} \subseteq c$ for some P is said to be *closed*.
 - A CNF is a tautology if and only if all of its clauses are closed.*
- Dually, a DNF is a contradiction iff all of its clauses are closed.

Example

- Consider the formula $A = ((P \rightarrow Q) \rightarrow P) \rightarrow P$ (previous example). Its CNF is

$$\{\{\neg P, Q, P\}, \{\neg P, P\}\}$$

Since all clauses are closed, we conclude that A is a tautology.

- Consider now $B = (P \rightarrow Q \vee R) \wedge \neg(P \wedge \neg Q \rightarrow R)$. Its CNF is

$$\{\{\neg A, A, \neg B\}, \{A, \neg B\}\}$$

the clause $\{A, \neg B\}$ is not closed, hence the formula is not a tautology (i.e. it is refutable).

- However, the applicability of this simple criterion for VAL is compromised by the potential exponential growth in the CNF transformation.
- As explained before, this limitation is overcome considering instead SAT...
- ...with satisfiability preserving CNFs (definitional CNF).
- obs.: The dual criterion can be used to decide (un)SAT on a propositional formula A (using its equivalent DNF).

Satisfiability in CNFs

- One of the most important methods to check satisfiability of CNFs is the *Davis-Putnam-Logemann-Loveland procedure (DPLL)*.
- DPLL is an algorithm for verifying if a particular CNF is a contradiction.
- It incrementally constructs a model compatible with a CNF...
- ...if no such model exists, the formula is signaled as a contradiction. Otherwise it is satisfiable.
- Basic observation: *if we fix the interpretation of a particular proposition symbol, we are able to simplify the corresponding CNF accordingly*
- Consider a proposition symbol P , a CNF S and a clause $c \in S$. For any model \mathcal{M} :
 - If $P \in \mathcal{M}$,
 - if $P \in c$ then $\mathcal{M} \models c$. Thus $\mathcal{M} \models S$ iff $\mathcal{M} \models S \setminus \{c\}$. In short, clauses containing P can be ignored.
 - $\mathcal{M} \models c$ iff $\mathcal{M} \models c \setminus \{\neg P\}$. In short, $\neg P$ can be removed from every clause in S .
 - Analogously if $P \notin \mathcal{M}$ (i.e. $\mathcal{M} \models \neg P$):
 - if $\neg P \in c$ then $\mathcal{M} \models c$ iff $\mathcal{M} \models S \setminus \{c\}$;
 - $\mathcal{M} \models c$ iff $\mathcal{M} \models c \setminus \{P\}$.

Davis-Putnam

These observations can be summarised as follows.

Definition

Let l be a literal and S a CNF.

- 1 The *opposite* of l (denoted by $\neg l$) is defined as

$$\neg l = \begin{cases} \neg P & , \text{ if } l = P; \\ P & , \text{ if } l = \neg P. \end{cases}$$

- 2 The *split* of S by l is

$$\text{split}^l(S) = \{c \setminus \neg l \mid c \in S, l \notin c\}$$

- Informally, $\text{split}^l(S)$ is a simplification of S assuming l holds.
- Note that neither l nor $\neg l$ occur in any clause of $\text{split}^l(S)$ or $\text{split}^{\neg l}(S)$.
- For a CNF S and proposition symbol P ,

$$S \equiv (P \rightarrow \text{split}^P(S)) \wedge (\neg P \rightarrow \text{split}^{\neg P}(S))$$

Recursively applying this simplification for every symbol occurring in a CNF is the heart of the DPLL algorithm.

Definition (DPLL Algorithm)

Let S be a CNF. The DPLL algorithm is defined recursively by

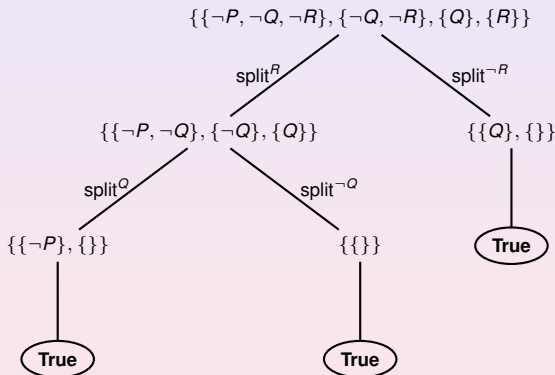
$$\text{DPLL}(S) = \begin{cases} \mathbf{F} & \text{if } S = \top \\ \mathbf{T} & \text{if } \perp \in S \\ \text{DPLL}(\text{split}(S)) \text{ and } \text{DPLL}(\text{split}(S)) & \text{otherwise} \end{cases}$$

where the literal l chosen in the recursive step is any literal appearing in S .

- The CNF S is a contradiction if $\text{DPLL}(S) = \mathbf{T}$;
- ...and satisfiable otherwise (a model can be extracted from the path of choices performed by the algorithm).

Example

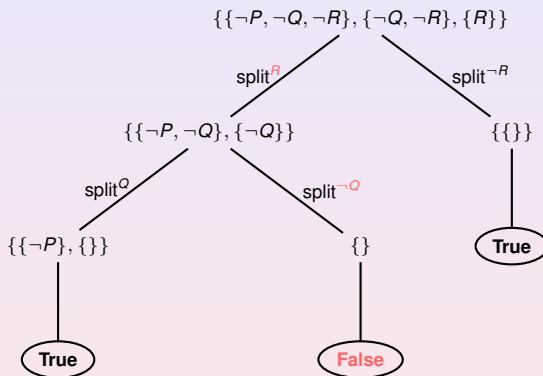
- The recursion tree for the execution of DPLL on the CNF $(\neg P \vee \neg Q \vee \neg R) \wedge (\neg Q \vee \neg R) \wedge Q \wedge R$.



- Since all the leaves are tagged with **T**, the formula is a *contradiction*.

Example

- Consider now the recursion tree for $(\neg P \vee \neg Q \vee \neg R) \wedge (\neg Q \vee \neg R) \wedge R$.



- A *false* leaf signals that the formula is *satisfiable*.
- The positive literals that occur in the corresponding path give rise to a model that validates the formula.

- The behaviour of the algorithm is highly dependent on the order in which the proposition symbols are chosen.
- Particular attention to how the next symbol is selected, in order to maximize the efficiency of the algorithm.
- Additional optimisations and heuristics are often explored to avoid unnecessary branches during execution
 - **unit-propagation:** singleton clauses $\{l\} \in S$ can (should) be used to simplify the CNF;
 - **pure literals:** literals that occur in clauses of S always with a given polarity can be removed.
- An heuristic often used is to choose the most frequent propositional symbol in S .

SAT solvers

- Propositional satisfiability has been successfully applied to perform hardware and software verification.
- Specialised tools exist that are capable of handling large instances of the satisfiability problem.
- A particular class of tools that are close to the computational approach exposed are the so called *SAT solvers*.
- The *satisfiability library* SATlib¹ is an online resource that proposes, as a standard, a unified notation and a collection of benchmarks for performance evaluation and comparison of tools.
- Such a uniform test-bed has been serving as a framework for regular tool competitions organised in the context of the regular SAT conferences.²

¹<http://www.satlib.org/>

²<http://www.satcompetition.org>

First-Order Logic

Validity in First-Order Logic

- Unsurprisingly, the problem of determining whether an arbitrary first-order sentence is valid is significantly harder than for the propositional case.
- In fact, it is impossible to solve this problem in its full generality.

Theorem

*The validity problem for first-order logic is **undecidable**.*

- This negative result (undecidability) is a direct consequence of a positive feature of first-order logic – its *expressive power*.
- Moreover, it does not preclude however restricted instances of the general problem from being solvable.
- We will see that the problem of validity-checking of first-order formulas can, to some extent, be reduced to the propositional case.
- This requires to restrict the use of quantifiers in formulas.

Negation Normal Form

Definition

A first-order formula is in *negation normal form* (NNF) if the implication connective is not used in it, and negation is only applied to atomic formulas.

- Every first-order formula is equivalent to a NNF formula.
- It can be computed by extending the propositional NNF normalisation with specific laws to handle quantifiers.

$$\begin{array}{ll}
 \phi \rightarrow \psi \equiv \neg\phi \vee \psi & \neg\neg\phi \equiv \phi \\
 \neg(\phi \wedge \psi) \equiv \neg\phi \vee \neg\psi & \neg(\phi \vee \psi) \equiv \neg\phi \wedge \neg\psi \\
 \neg\forall x. \phi \equiv \exists x. \neg\phi & \neg\exists x. \phi \equiv \forall x. \neg\phi
 \end{array}$$

- Example: to compute the NNF of $\forall x. (\forall y. P(x, y) \vee Q(x)) \rightarrow \exists z. P(x, z)$.

$$\begin{array}{ll}
 \forall x. (\forall y. P(x, y) \vee Q(x)) \rightarrow \exists z. P(x, z) & \equiv \\
 \forall x. \neg(\forall y. P(x, y) \vee Q(x)) \vee \exists z. P(x, z) & \equiv \\
 \forall x. \exists y. (\neg P(x, y) \wedge \neg Q(x)) \vee \exists z. P(x, z) &
 \end{array}$$

Prenex Normal Form

- If x does not occur free in ψ , then the following equivalences hold.

$$(\forall x. \phi) \wedge \psi \equiv \forall x. \phi \wedge \psi$$

$$\psi \wedge (\forall x. \phi) \equiv \forall x. \psi \wedge \phi$$

$$(\forall x. \phi) \vee \psi \equiv \forall x. \phi \vee \psi$$

$$\psi \vee (\forall x. \phi) \equiv \forall x. \psi \vee \phi$$

$$(\exists x. \phi) \wedge \psi \equiv \exists x. \phi \wedge \psi$$

$$\psi \wedge (\exists x. \phi) \equiv \exists x. \psi \wedge \phi$$

$$(\exists x. \phi) \vee \psi \equiv \exists x. \phi \vee \psi$$

$$\psi \vee (\exists x. \phi) \equiv \exists x. \psi \vee \phi$$

- The applicability of these equivalences can always be assured by appropriate renaming of bound variables.
- Applying these equations to a NNF leads to formulas where quantifiers are in the outermost position.

Definition

A formula is in *prenex form* if it is of the form $Q_1 x_1. Q_2 x_2. \dots Q_n x_n. \psi$ where each Q_i is a quantifier (either \forall or \exists) and ψ is a quantifier-free formula.

Herbrand/Skolem Normal Form

Definition (Herbrand and Skolem Forms)

Let ϕ be a first-order formula in prenex normal form. The *Herbrandization* of ϕ (written ϕ^H) is an existential formula obtained from ϕ by repeatedly and exhaustively applying the following transformation:

$$\exists x_1, \dots, x_n. \forall y. \psi \rightsquigarrow \exists x_1, \dots, x_n. \psi[f(x_1, \dots, x_n)/y]$$

with f a **fresh** function symbol with arity n (i.e. f does not occur in ψ).

Dually, the *Skolemization* of ϕ (written ϕ^S) is a universal formula obtained from ϕ by repeatedly applying the transformation:

$$\forall x_1, \dots, x_n. \exists y. \psi \rightsquigarrow \forall x_1, \dots, x_n. \psi[f(x_1, \dots, x_n)/y]$$

again, f is a **fresh** function symbol with arity n .

Herbrand normal form (resp. *Skolem normal form*) formulas are those obtained by this process.

Proposition

Let ϕ be a first-order formula in prenex normal form. ϕ is valid iff its Herbrandization ϕ^H is valid. Dually, ϕ is unsatisfiable iff its Skolemization ϕ^S is unsatisfiable.

- It is convenient to write Herbrand and Skolem formulas using vector notation $\exists \bar{x}. \psi$ and $\forall \bar{x}. \psi$ (with ψ quantifier free), respectively.
- The quantifier-free sub-formula can be furthered normalised:
Universal CNF: $\forall \bar{x}. \bigwedge_i \bigvee_j l_{ij}$
Existential DNF: $\exists \bar{x}. \bigvee_i \bigwedge_j l_{ij}$
 where *literals* are either atomic predicates or negation of atomic predicates.
- Herbrandization/Skolemization change the underlying vocabulary. These additional symbols are called *Herbrand/Skolem functions*.
- (obs: this observation alone suffices to show that a formula and its Herbrandization/Skolemization are not equivalent.)

Herbrand Model

Definition (Herbrand Interpretation)

Let \mathcal{V} be a first-order vocabulary and assume \mathcal{V} has at least one constant symbol (otherwise, we explicitly expand the vocabulary with such a symbol). A *Herbrand Interpretation* $\mathcal{H} = (D_{\mathcal{H}}, I_{\mathcal{H}})$ is a \mathcal{V} -structure specified by a set of *ground atomic predicates* (i.e. atomic predicates applied to ground terms), also denoted by \mathcal{H} . The interpretation structure is given as follows:

- Interpretation domain: $D_{\mathcal{H}}$ is the set of ground terms for the vocabulary \mathcal{V} . It is called the *Herbrand universe* for \mathcal{V} .
- Interpretation of constants: for every $c \in \mathcal{V}$, $I_{\mathcal{H}}(c) = c$;
- Interpretation of functions: for every $f \in \mathcal{V}$ with $\text{ar}(f) = n$, $I_{\mathcal{H}}(f)$ consists of the n -ary function that, given ground terms t_1, \dots, t_n , returns the ground term $f(t_1, \dots, t_n)$;
- Interpretation of predicates: for every $P \in \mathcal{V}$ with $\text{ar}(P) = n$, $I_{\mathcal{H}}(P)$ is the n -ary relation $\{(t_1, \dots, t_n) \mid P(t_1, \dots, t_n) \in \mathcal{H}\}$.

Herbrand's Theorem

Lemma

An existential formula ϕ is valid iff for every Herbrand model \mathcal{H} , $\mathcal{H} \models \phi$. Dually, a universal formula ϕ is unsatisfiable iff there exists no Herbrand model \mathcal{H} such that $\mathcal{H} \models \phi$.

Theorem (Herbrand's Theorem)

An existential first-order formula $\exists \bar{x}. \psi$ (with ψ quantifier-free) is valid iff there exists an integer k and ground instances $\psi\sigma_1, \dots, \psi\sigma_k$ such that $\psi\sigma_1 \vee \dots \vee \psi\sigma_k$ is propositionally valid.

Dually, a universal formula $\forall \bar{x}. \psi$ (with ψ quantifier-free) is unsatisfiable iff there exists an integer k and closed instances $\psi\sigma_1, \dots, \psi\sigma_k$ such that $\psi\sigma_1 \wedge \dots \wedge \psi\sigma_k$ is propositionally unsatisfiable.

Application

Theorem (Semi-Decidability)

The problem of *validity of first-order formulas is semi-decidable*, i.e. there exists a procedure that, given a first-order formula, answers “yes” iff the formula is valid (but might not terminate if the formula is not valid).

- An interesting refinement is to investigate fragments in which bounds can be established for searching the ground instance space.
- This immediately leads to a bound on the number of instances whose search is required by Herbrand's theorem...
- ...turning validity of formulas decidable.
- Clearly if the set of ground terms is finite, the set of ground instances of the formula under scrutiny will be finite as well.

Decidable Fragments

- If the underlying vocabulary has no function symbol, the set of ground terms is finite.
- Note however that function symbols might be introduced during the Herbrandization/Skolemization.
- Restricting attention to formulas whose prenex normal form has the shape

$$\forall \bar{x}. \exists \bar{y}. \psi$$

ensures that only constants are introduced by Herbrandization.

- This fragment of formulas is normally known as the *AE fragment*, owing its name to the alternation of quantifiers allowed (*A* refers to the universal quantifier and *E* to existential quantifier).
- The class of formulas can be further enlarged by observing that a formula not in AE may be equivalent to one in AE (e.g. *miniscope* — pushing existential quantifiers inside the formula, thus minimizing their scopes).
- *Monadic formulas* (i.e. formulas containing only unary predicates) are such a class of formulas. Hence, they constitute a *decidable fragment* of first-order logic.

First-Order Theories

- When judging the validity of first-order formulas we are typically interested in a particular domain of discourse...
- ... which in addition to a specific underlying vocabulary includes also properties that one expects to hold.
- That is, we are often interested in *moving away from pure logical validity* (i.e. validity in all models) *towards a more refined notion* of validity restricted to a specific class of models.
- A natural way for specifying such a class of models is by providing a *set of axioms* (sentences that are expected to hold in them).
- Alternatively, one can pinpoint the *models of interest*.
- *First-order Theories* provides the basis for the kind of reasoning just described.

First-Order Theories

Definition

Let \mathcal{V} be a vocabulary of a first-order language.

- A first-order *theory* \mathcal{T} is a set of \mathcal{V} -sentences that is closed under derivability (i.e., $\mathcal{T} \vdash \phi$ implies $\phi \in \mathcal{T}$). A *\mathcal{T} -structure* is a \mathcal{V} -structure that validates every formula of \mathcal{T} .
- A formula ϕ is *\mathcal{T} -valid* (resp. *\mathcal{T} -satisfiable*) if every (resp. some) \mathcal{T} -structure validates ϕ .
- A first-order theory \mathcal{T} is said to be a *consistent* theory if at least one \mathcal{T} -structure exists. \mathcal{T} is said to be a *complete* theory if, for every \mathcal{V} -sentence ϕ , either $\mathcal{T} \models \phi$ or $\mathcal{T} \models \neg\phi$. \mathcal{T} is said to be a *decidable* theory if there exists a decision procedure for checking \mathcal{T} -validity.
- Let K be a class of \mathcal{V} -structures. The *theory of K* , denoted by $\text{Th}(K)$, is the set of sentences valid in all members of K , i.e., $\text{Th}(K) = \{\psi \mid \mathcal{M} \models \psi, \text{ for all } \mathcal{M} \in K\}$. Conversely, given a set of \mathcal{V} -sentences Γ , the class of *models for Γ* is defined as $\text{Mod}(\Gamma) = \{\mathcal{M} \mid \text{for all } \phi \in \Gamma, \mathcal{M} \models \phi\}$.
- A subset $\mathcal{A} \subseteq \mathcal{T}$ is called an *axiom set* for the theory \mathcal{T} when \mathcal{T} is the deductive closure of \mathcal{A} , i.e. $\psi \in \mathcal{T}$ iff $\mathcal{A} \vdash \psi$. A theory \mathcal{T} is *finitely* (resp. *recursively*) *axiomatisable* if it possesses a finite (resp. recursive) set of axioms.

- Whenever a theory \mathcal{T} is axiomatisable (by a finite or recursive set of axioms \mathcal{A}), it makes sense to extend the first-order logic proof system \mathcal{N}_{FOL} with an axiom-schema:

$$\frac{}{\Gamma \vdash \phi} \text{ if } \phi \in \mathcal{A}$$

- Observe that the requirement that \mathcal{A} be a recursive set is crucial to ensure that the applicability of these axioms can effectively be checked.
- Moreover, if a theory \mathcal{T} has a recursive set of axioms, the theory itself is recursively enumerable (hence, the \mathcal{T} -validity problem is semi-decidable).
- If \mathcal{T} is a complete theory, then any \mathcal{T} -structure validates exactly the same set of \mathcal{T} -sentences (the theory itself).

- For a given \mathcal{V} -structure \mathcal{M} , the theory $\text{Th}(\mathcal{M})$ (of a single-element class of \mathcal{V} -structures) is complete. These semantically defined theories are useful when one is interested in reasoning in some specific mathematical domain such as the natural numbers, rational numbers, etc.
- However, we remark that such theory may lack an axiomatisation, which seriously compromises its use in purely deductive reasoning.
- If a theory is complete and has a recursive set of axioms, it can be shown to be *decidable*.
- The decidability criterion for \mathcal{T} -validity is crucial for mechanised reasoning in the theory \mathcal{T} .
- It may be necessary (or convenient) to restrict the class of formulas under consideration to a suitable *fragment*;
- The \mathcal{T} -validity problem in a fragment refers to the decision about whether or not $\phi \in \mathcal{T}$ when ϕ belongs to the fragment under consideration.
- A fragment of interest is the fragment consisting of universal formulas, often referred to as the *quantifier-free fragment*

Some Theories

Equality and Uninterpreted-Functions \mathcal{T}_E : theory whose the only axioms are the ones related with equality (reflexivity and congruence).
 \mathcal{T}_E -validity is undecidable in general, but efficiently decidable for the quantifier-free.

Natural Numbers and Integers $\mathcal{T}_{\mathbb{N}}$, $\mathcal{T}_{\mathbb{Z}}$: the semantic theory of natural numbers (with operations 0, succ, +, *) and integers. It is neither axiomatisable nor decidable (*Godël incompleteness theorem*).

Peano Arithmetic \mathcal{T}_{PA} : a first-order approximation of the theory of natural numbers. Its axiomatisation includes <an axiom scheme for induction

$$\frac{\phi [0/x] \quad \forall n. \phi [n/x] \rightarrow \phi [n + 1/x]}{\forall n. \phi [n/x]}$$

It is incomplete and undecidable (even for the quantifier-free fragment).

Linear Arithmetic \mathcal{T}_{LA} : with vocabulary $\mathcal{V} = \{\dots, -2, -1, 0, 1, 2, \dots - 2\cdot, -1\cdot, 1\cdot, 2\cdot, \dots, +, =, <\}$, where $n\cdot$ is a unary function that multiplies its argument by a constant. This theory is both complete and decidable, and it is in fact one of the most widely used in the context of program verification.

Rational Numbers: the full theory of rational numbers (with addition and multiplication) is undecidable, since the property of being a natural number can be encoded in it. But the theory of *linear arithmetic over rational numbers* \mathcal{T}_{QLA} is decidable, and actually more efficiently than the corresponding theory of integers.

Reals $\mathcal{T}_{\mathbb{R}}$: surprisingly, this theory is decidable even in the presence of multiplication and quantifiers. However, the time complexity of the associated decision procedure may make its application prohibitive.

Fixed-size bit vectors: model bit-level operations of machine words, including 2^n -modular operations (where n is the word size), shift operations, etc. Decision procedures for the theory of fixed-sized bit vectors often rely on appropriate encodings in propositional logic.

Arrays, Finite Maps, Lists...

Satisfiability Modulo Theories

- The SMT problem is a variation of the propositional SAT problem for first-order logic, with the interpretation of symbols constrained by (a combination of) specific theories.
- More precisely, SMT solvers address the issue of satisfiability of quantifier-free first-order CNF formulas, using as building blocks:
 - 1 a propositional SAT-solver,
 - 2 and state-of-the-art theory-solvers.
- For a first-order CNF ϕ :
 - Let $\text{prop}(-)$ be a map from first-order formulas to propositional formulas that substitutes every atomic formula by a fresh propositional symbol.
 - For a valuation ρ of $\text{prop}(\phi)$, the set $\Phi(\rho)$ of first-order literals be defined as follows

$$\Phi(\rho) = \{\text{prop}^{-1}(P_i) \mid \rho(P_i) = \mathbf{T}\} \cup \{\neg\text{prop}^{-1}(P_i) \mid \rho(P_i) = \mathbf{F}\}$$

- Given a CNF, the SAT-solver answers either “**unsat**”, or “**sat**” with a particular valuation (model).
- Given a conjunction of atomic formulas, the *theory-solver* answers either “**T-consistent**”, or “**T-inconsistent**” with a particular “**unsatisfiable kernel**” (i.e. a subset of the given set that is already unsatisfiable)

SMT loop

```

SMT-Solver ( $\psi$ ) =
   $A \leftarrow \text{prop}(\psi)$ 
  loop
     $(r, \rho) \leftarrow \text{SAT}(A)$ 
    if  $r = \text{unsat}$  then return unsat
     $(r, \Upsilon) \leftarrow \text{TSolver}(\Phi(\rho))$ 
    if  $r = \text{sat}$  then return sat
     $C \leftarrow \bigvee_{B \in \Upsilon} \neg \text{prop}(B)$ 
     $A \leftarrow A \wedge C$ 
  
```

- The main loop invokes the propositional SAT solver with a propositional formula A that is initialised with $\text{prop}(\psi)$.
- If a valuation ρ satisfying A is found, the theory solver is invoked to check if $\Phi(\rho)$ is satisfiable.
- If not, it will add to A a clause which will have the effect of excluding ρ when the SAT solver is invoked again in the next iteration.
- The algorithm stops whenever the SAT solver returns “unsat”, in which case ψ is unsatisfiable,
- or the theory solver returns “sat”, in which case ψ is satisfiable.

Example

- Consider the formula

$$\underbrace{g(a) = x}_1 \wedge \underbrace{(f(g(a)) \neq f(c))}_{-2} \vee \underbrace{g(a) = d}_3 \wedge \underbrace{c \neq d}_{-4}$$

- Send to SAT-solver the CNF $\{\{1\}, \{-2, 3\}, \{-4\}\}$. It answers *satisfiable* with model $\{1, -2, -4\}$
- Send correspondent model to the *Theory-solver*. It answers *T-inconsistent*.
- Consider additional clause negating the unsatisfiable kernel returned by the theory-solver (assumed to be the whole model, i.e. $\{-1, 2, 4\}$).
- Send to SAT-solver $\{\{1\}, \{-2, 3\}, \{-4\}, \{-1, 2, 4\}\}$. It answers *satisfiable* with model $\{1, 2, 3, -4\}$.
- Send model to *Theory-solver*. It answers *T-inconsistent*.
- Send to SAT-solver $\{\{1\}, \{-2, 3\}, \{-4\}, \{-1, 2, 4\}, \{-1, -2, -3, 4\}\}$. It answers *unsatisfiable*.