SAT-BASED ALGORITHMS

Input: IAF = (A, R₁, R₂), a ∈ A.

Basis: SAT Encodings

Boolean variables:

- yₐ for each argument a, τₐb for each attack (a, b)
- true if element included in completion
- xₐ for each argument a
- true if argument included in ϵ-extension
- zₐ for each argument a
- true if argument attacked by ϵ-extension

Propositional formulas:

- ϕ(AF) encodes valid completions
- ϕ(AF) for ϵ ∈ {AD, ST, CF, GR} encodes the semantics

The encoding for complete semantics is

ϕᵾ(AF) = ϕᵣ(AF) \land \bigwedge_{a \in A} \bigwedge_{(b,a) \in R₁} \bigwedge_{(a,b) \in R₂} \bigwedge_{τₐb} \bigwedge_{zₐ} \bigwedge_{xₐ} \bigwedge_{τₐb} \bigwedge_{zₐ} \bigwedge_{xₐ}.

PCA and NSA solved via direct SAT calls.

SAT-based CEGAR Algorithms

Algorithms where a SAT solver is called iteratively and incrementally using different assumptions.

Task: Possible skeptical acceptance under preferred.

ϕ ← ABSTRACTION(AF, a) \triangleright Initialize abstraction
while true do

(sat, τ) ← SAT(ϕ) \triangleright Solve abstraction
if sat = false then return reject
AF ← COMPLETE(τ)
if sat = false then return accept \triangleright example?
ϕ ← ϕᵣ REFINEMENT(AF, AF') \triangleright Exclude completion
end while

Strong refinement: take into account which atomic changes preserve the counterexample extension of the completion: e.g. for preferred semantics adding arguments which are attacked by the extension can be safely ignored.

In journal article: SAT encodings, CEGAR algorithms, and strong refinements for all other variants.