Universal second-order logic

Of course, we can negate (2.12) and obtain

$$\forall P \exists x \exists y \exists z \, (\neg C_1 \vee \neg C_2 \vee \neg C_3 \vee \neg C_4) \qquad (2.14)$$

by relying on the familiar de Morgan laws. This is a formula of *universal second-order logic*. This formula expresses reachability.

Theorem 2.28 Let $\mathcal{M} = (A, R^{\mathcal{M}})$ be any model. Then the formula in (2.14) holds under look-up table l in $\mathcal{M}$ iff $l(v)$ is R-reachable from $l(u)$ in $\mathcal{M}$.

PROOF:

1. First, assume that $\mathcal{M}_T \models_l \exists x \exists y \exists z \, (\neg C_1 \vee \neg C_2 \vee \neg C_3 \vee \neg C_4)$ holds for all interpretations T of P. Then it also holds for the interpretation which is the reflexive, transitive closure of $R^{\mathcal{M}}$. But for that T, $\mathcal{M}_T \models_l \exists x \exists y \exists z \, (\neg C_1 \vee \neg C_2 \vee \neg C_3 \vee \neg C_4)$ can hold only if $\mathcal{M}_T \models_l \neg C_3$ holds, as all other clauses C_i $(i \neq 3)$ are false. But this means that $\mathcal{M}_T \models_l P(u, v)$ has to hold. So $(l(u), l(v)) \in T$ follows, meaning that there is a finite path from $l(u)$ to $l(v)$.
2. Conversely, let $l(v)$ be R-reachable from $l(u)$ in $\mathcal{M}$.
   - For any interpretation T of P which is not reflexive, not transitive or does not contain $R^{\mathcal{M}}$ the relation $\mathcal{M}_T \models_l \exists x \exists y \exists z \, (\neg C_1 \vee \neg C_2 \vee \neg C_3 \vee \neg C_4)$ holds, since T makes one of the clauses $\neg C_1$, $\neg C_2$ or $\neg C_4$ true.
   - The other possibility is that T be a reflexive, transitive relation containing $R^{\mathcal{M}}$. Then T contains the reflexive, transitive closure of $R^{\mathcal{M}}$. But $(l(u), l(v))$ is in that closure by assumption. Therefore, $\neg C_3$ is made true in the interpretation T under look-up table l, and so $\mathcal{M}_T \models_l \exists x \exists y \exists z \, (\neg C_1 \vee \neg C_2 \vee \neg C_3 \vee \neg C_4)$ holds.

In summary, $\mathcal{M}_T \models_l \exists x \exists y \exists z \, (\neg C_1 \vee \neg C_2 \vee \neg C_3 \vee \neg C_4)$ holds for all interpretations $T \subseteq A \times A$. Therefore, $\mathcal{M} \models_l \forall P \exists x \exists y \exists z \, (\neg C_1 \vee \neg C_2 \vee \neg C_3 \vee \neg C_4)$ holds. □

It is beyond the scope of this text to show that reachability can also be expressed in existential second-order logic, but this is indeed the case. It is an important open problem to determine whether existential second-order logic is closed under negation, i.e. whether for all such formulas ∃P φ there is a formula ∃Q ψ of existential second-order logic such that the latter is semantically equivalent to the negation of the former. If we allow existential and universal quantifiers to apply to predicate symbols in the same formula, we arrive at fully-fledged second-order logic, e.g

$$\exists P \forall Q \, (\forall x \forall y \, (Q(x, y) \rightarrow Q(y, x)) \rightarrow \forall u \forall v \, (Q(u, v) \rightarrow P(u, v))). \qquad (2.15)$$

If one wants to quantify over relations of relations, one gets third-order logic etc. Higher-order logics require great care in their design. Typical results such as completeness and compactness

may quickly fail to hold. Even worse, a naive higher-order logic may be inconsistent at the meta-level. Related problems were discovered in naive set theory, e.g. in the attempt to define the 'set' A that contains as elements those sets X that do not contain themselves as an element:

$$A \stackrel{\text{def}}{=} \{X \mid X \notin X\}.$$

Micromodels of software

Two of the central concepts developed so far are

- model checking: given a formula φ of predicate logic and a matching model M determine whether M φ holds;
- semantic entailment: given a set of formulas Γ of predicate logic, is Γ φ valid?

In the case of semantic entailment, Γ should contain all the requirements we impose on a software design and φ may be a property we think should hold in any implementation that meets the requirements Γ. Semantic entailment therefore matches well with software specification and validation; alas, it is undecidable in general. Since model checking is decidable, why not put all the requirements into a model M and then check M φ? The difficulty with this approach is that, by comitting to a particular model M, we are comitting to a lot of detail which doesn't form part of the requirements. Typically, the model instantiates a number of parameters which were left free in the requirements. From this point of view, semantic entailment is better, because it allows a variety of models with a variety of different values for those parameters. We seek to combine semantic entailment and model checking in a way which attempts to give us the advantages of both. We will extract from the requirements a relatively small number of small models, and check that they satisfy the property φ to be proved. This satisfaction checking has the tractability of model checking, while the fact that we range over a set of models (albeit a small one) allows us to consider different values of parameters which are not set in the requirements. This approach is implemented in a tool called Alloy, due to D. Jackson. The models we consider are what he calls 'micromodels' of software.