14.1 Inductive Definitions

Previously, we saw that there is no first order formula φ that expresses the property of graph-connectedness: "connected" and "path" cannot be expressed in $\mathcal{L}(\Sigma_{\text{st graph}})$ where $\Sigma_{\text{st graph}} = (E^2; s, t)$ – the vocabulary of graphs with two constants symbols, s and t.

Define the problem REACH to be the set of directed graphs having a path from s to t,

$$\text{REACH} \quad = \quad \left\{ G \in \text{STRUC}[\Sigma_{\text{st graph}}] \;\middle|\; s \overset{*}{\underset{\text{G}}{\rightarrow}} t \right\} .$$

We know from the handout on Ehrenfeucht-Fraïssé games that expressing $\text{dist}_{\leq n}(x, y)$, i.e., that there is a path from x to y of length $\leq n$, requires quantifier depth exactly $\lceil log(n) \rceil$.

We want to express the path relation, E^*, the reflexive, transitive closure of E. That is E^* is the smallest binary relation that is relexive and transitive and contains E. We can define E^* with the following inductive definition:

$$E^*(x, y) \quad \stackrel{\text{def}}{=} \quad (x = y \vee E(x, y) \vee \exists z \, (E^*(x, z) \wedge E^*(z, y))) \tag{14.1}$$

Here, we have defined E^* in terms of itself. We now show how to make sense of such definitions, assuming that the relation being defined appears only positively: when the formula is in negation normal form, no "$\neg$"s are applied to the relation being defined.

We can understand Eqn. 14.1 better with the following first-order operator on binary relations:

$$\varphi_{\text{tc}}(R, x, y) \quad \stackrel{\text{def}}{=} \quad (x = y \vee E(x, y) \vee \exists z \, (R(x, z) \wedge R(z, y))) \tag{14.2}$$

Given a graph, G, Eqn. 14.2 defines an operation, φ^G_{tc}, mapping binary relations on $|G|$ to binary relations on $|G|$:

$$\varphi^G_{\text{tc}}(R) \quad \stackrel{\text{def}}{=} \quad \left\{ (a, b) \in |G|^2 \;\middle|\; G[a/x, b/y] \models \varphi_{\text{tc}}(R, x, y) \right\} \tag{14.3}$$

Proposition 14.4 (Positive implies Monotone) *If R appears only positively in $\varphi(R^k, x_1, \ldots, x_k)$ then for any appropriate structure, $\mathcal{A}$, $\varphi^{\mathcal{A}}$ is a* monotone *operator on k-ary relations on $|\mathcal{A}|$, that is, for all such relations R, R' on $\mathcal{A}$,*

$$R \subseteq R' \quad \Rightarrow \quad \varphi^{\mathcal{A}}(R) \subseteq \varphi^{\mathcal{A}}(R') .$$

Example 14.5 Let us see the effect of the operator φ^G_{tc} from Eqn. 14.3 as we repeatedly apply it, starting with the empty relation, $\emptyset$, i.e., the relation that is false on every pair of vertices from G.

$$
\begin{aligned}
\varphi_{\mathrm{tc}}^{G}(\emptyset) &= \left\{(a,b)\in|G|^2 \;\middle|\; G,a/x,b/y\models x=y\vee E(x,y)\right\} \;=\; \left\{(a,b)\in|G|^2 \;\middle|\; \mathrm{dist}_{\leq 1}(a,b)\right\}\\
\varphi_{\mathrm{tc}}^{G}(\varphi_{\mathrm{tc}}^{G}(\emptyset)) &= \left\{(a,b)\in|G|^2 \;\middle|\; \mathrm{dist}_{\leq 2}(a,b)\right\}\\
(\varphi_{\mathrm{tc}}^{G})^3(\emptyset) &= \left\{(a,b)\in|G|^2 \;\middle|\; \mathrm{dist}_{\leq 4}(a,b)\right\}\\
(\varphi_{\mathrm{tc}}^{G})^4(\emptyset) &= \left\{(a,b)\in|G|^2 \;\middle|\; \mathrm{dist}_{\leq 8}(a,b)\right\}
\end{aligned}
$$

Note that each time we apply $\varphi_{\mathrm{tc}}^{G}$ we double the length of possible paths.

Thus, $(\varphi_{\mathrm{tc}}^{G})^k(\emptyset)=\left\{(a,b)\in|G|^2 \;\middle|\; \mathrm{dist}_{\leq 2^{k-1}}(a,b)\right\}$. In particular, since paths in an n-vertex graph can have length at most $n-1$, we have that

$$
(E^G)^* \quad = \quad (\varphi_{\mathrm{tc}}^{G})^{\lceil 1+\log(\|G\|)\rceil}(\emptyset) \quad = \quad \mathrm{LFP}(\varphi_{\mathrm{tc}}^{G})\ .
$$

$(E^G)^*$ is the *least fixed point* of $\varphi_{\mathrm{tc}}^{G}$, i.e., the smallest binary relation, $R\subseteq|G|^2$ such that $\varphi_{\mathrm{tc}}^{G}(R)=R$. We take that as the meaning of the inductive defintion Eqn. 14.1, i.e, $E^*\stackrel{\mathrm{def}}{=}\mathrm{LFP}(\varphi_{\mathrm{tc}})$. $\square$

Definition 14.6 [Least Fixed Point] If $\varphi(R^k,x_1,\ldots,x_k)$ is R-positive then the meaning of the inductive definition $R\stackrel{\mathrm{def}}{=}\varphi(R)$ is the least fixed point of φ, LFP(φ). $\square$

We now show that when φ is R-positive, and thus monotone by Prop. 14.4, the least fixed point always exists.

Theorem 14.7 (Tarski-Knaster Theorem) *If $\varphi(R^k,x_1,......x_k)$ is R positive then $LFP(\varphi)$ exists and can be computed in polynomial time.*

Proof: We first show that the process of starting with the emptyset and repeatedly applying φ, as we did in Example 14.5, always gives us a fixed point of φ.

Note that $\emptyset\subseteq\varphi(\emptyset)$. If $\emptyset=\varphi(\emptyset)$ then we are done and $\emptyset=\mathrm{LFP}(\varphi)$. Otherwise, by monotonicity of φ, $\varphi(\emptyset)\subseteq\varphi^2(\emptyset)$. If $\varphi(\emptyset)=\varphi^2(\emptyset)$ then we have reached a fixed point. Otherwise, continue the process:

$$
\emptyset \;\subseteq\; \varphi(\emptyset) \;\subseteq\; \varphi^2(\emptyset) \;\subseteq\; \varphi^3(\emptyset) \;\subseteq\; \cdots\varphi^{n^k}(\emptyset)=\varphi^{n^k+1}(\emptyset) \tag{14.8}
$$

In every step, either a fixed point is reached or a new k-tuple is added to the relation. A structure $\mathcal{A}$ with an n-element universe has n^k possible k-tuples. Therefore, after at most n^k iterations, a fixed point is reached. Let the fixed point be $\varphi^t(\emptyset)$ where $t\leq n^k$ is minimum such that $\varphi^t=\varphi^{t+1}$.

Now we want to show that $\varphi^t(\emptyset)$ is in fact the least fixed point. Let S be a fixed point of φ, i.e., $\varphi(S)=S$

Claim: $\varphi^t(\emptyset)\subseteq S$.

We prove by induction that for all i, $\varphi^i(\emptyset) \subseteq S$.

base case: $\varphi^0(\emptyset) = \emptyset \subseteq S$.

inductive case: assume that $\varphi^k(\emptyset) \subseteq S$.

By monotonicity of φ, it follows that $\varphi(\varphi^k(\emptyset)) \subseteq \varphi(S)$, i.e., $\varphi^{k+1}(\emptyset) \subseteq S$.

Thus, $\varphi^t(\emptyset) \subseteq S$ and as desired, $\varphi^t(\emptyset) = \text{LFP}(\varphi)$. $\square$

14.2 Datalog

Datalog is a database query language that makes use of positive recursions. The following is an example of a recursive definition in Datalog.

$$\begin{array}{lcl} P(x,y) & :- & x = y \\ P(x,y) & :- & E(x,y) \\ P(x,y) & :- & P(x,z), P(z,y) \end{array}$$

Note that this Datalog code is equivalent to the inductive definition,

$$\left(P(x,y) \stackrel{\text{def}}{=} x = y \ \lor \ E(x,y) \ \lor \ \exists z \left(P(x,z) \land P(z,y)\right)\right) .$$

In particular, the separate lines are "or"-ed together; the comma in a single line is treated as "$\land$". Free variables occuring only on the right-hand side are considered existentially quantified, whereas free variables that occur on the left side are universally quantified.

Here is another Datalog example. Given the database relation Parent(x, y), we can make the non-recursive Datalog definition:

$$\text{Sib}(x,y) :- \text{Parent}(z,x), \text{Parent}(z,y), x \neq y$$

Here is another recursive definition:

$$\begin{array}{lcl} \text{Ancestor}(x,y) & :- & x = y \\ \text{Ancestor}(x,y) & :- & \text{Parent}(x,y) \\ \text{Ancestor}(x,y) & :- & \text{Ancestor}(x,z), \text{Ancestor}(z,y) \end{array} \tag{14.8}$$

In Datalog, recursive definitions are implemented exactly as they would be in logic using a Breadth-first search matching algorithm.

14.3 Prolog

Prolog is a programming language older and more complicated than Datalog. In trying to make Prolog a general-purpose programming language, the designers made some choices which take the meanings of programs away from what the meaning would be in logic.

In particular, consider the Ancestor query, $\text{Ancestor}(x, y) :\!-\, ?$, in Prolog, using the definition of Ancestor from Eqn. 14.8.

This is meant to return all pairs (a, b) such that a is an ancestor of b. Unfortunately, Prolog uses a depth-first search matching algorithm. Thus to match $\text{Ancestor}(x, y)$ it would first try to match $\text{Ancestor}(x, z)$. To do this, it would first try to match $\text{Ancestor}(x, z_1)$ and so on, thus going into an infinite loop and never answering.

On the other hand, Prolog would do the right thing with the alternate definition:

$$\begin{array}{lcl} \text{Ancestor}(x, y) & :\!- & x = y \\ \text{Ancestor}(x, y) & :\!- & \text{Parent}(x, z), \text{Ancestor}(z, y) \end{array} \tag{14.8}$$