A.1.8 Defining Acyclicity for an Undirected Graph

An undirected graph can be represented as a binary relation, constrained to be symmetric. Write constraints on such a relation that rule out cycles. Here is a suitable template:

```alloy
module exercises/undirected
sig Node {adjs: set Node}
pred acyclic () {
    adjs = ~adjs
    ...
    your constraints here
}
run acyclic for 4
```

A.1.9 Axiomatizing Transitive Closure

Transitive closure is not axiomatizable in first-order logic. In short, that means that if you want to express it, you need a special operator, because it can’t be defined in terms of other operators. Here’s a bogus attempt to do just that; your challenge is to use the Alloy Analyzer to find the flaw.

Recall that the transitive closure of a binary relation \( r \) is the smallest transitive relation \( R \) that includes \( r \). Let’s say \( R \) is a transitive cover of \( r \) if \( R \) is transitive and includes \( r \). To ensure that \( R \) is the smallest transitive cover, we can say that removing any tuple \( a \rightarrow b \) from \( R \) gives a relation that is not a transitive cover of \( r \). Formalize this by completing the following template:

```alloy
module exercises/closure
pred transCover (R, r: univ->univ) {
    ...
    your constraints here
}
pred transClosure (R, r: univ->univ) {
    transCover [R, r]
    ...
    your constraint here
}
assert Equivalence {
    all R, r: univ->univ | transClosure[R, r] iff R = ^r
}
check Equivalence for 3
```

Now execute the command, examine the counterexample, and explain what the bug is.
In fact, for finite domains – which is how Alloy is interpreted – closure can be axiomatized in first-order logic. Some recent technical reports explains how to do this [16, 9]. (Thanks to Masahiro Sakai for telling me about this work.) Define a ternary relation \( C \) such that

\[
x \rightarrow y \rightarrow z \text{ in } C
\]

when \( y \) is at some non-zero distance on a shortest path in the relation \( r \) from \( x \) to \( z \). The reflexive transitive closure \( R \) of \( r \) can be expressed in terms of \( C \) as

\[
R = \{ x, y : \text{univ} | x \rightarrow y \rightarrow y \text{ in } C \text{ or } x = y \}
\]

and—this is the surprising part—\( C \) itself can be defined by the following axioms:

\[
\text{all } x, y, z, u: \text{univ} \{
\begin{align*}
x \rightarrow x \rightarrow y & \text{ not in } C \\
x \rightarrow y \rightarrow u & \text{ in } C \text{ and } y \rightarrow z \rightarrow u \text{ in } C \Rightarrow x \rightarrow z \rightarrow u \text{ in } C \\
x \rightarrow y \rightarrow y & \text{ in } C \text{ and } y \rightarrow z \rightarrow z \text{ in } C \text{ and } x \neq z \Rightarrow x \rightarrow z \rightarrow z \text{ in } C \\
x \rightarrow y \rightarrow y & \text{ in } r \text{ and } x \neq y \Rightarrow x \rightarrow y \rightarrow y \text{ in } C \\
x \rightarrow y \rightarrow y & \text{ in } C \Rightarrow \text{some } v: \text{univ} | x \rightarrow v \text{ in } r \text{ and } x \rightarrow v \rightarrow y \text{ in } C \\
x \rightarrow y \rightarrow z & \text{ in } C \text{ and } y \neq z \Rightarrow y \rightarrow z \rightarrow z \text{ in } C
\end{align*}
\}
\]

To check this axiomatization, just define a predicate like this:

\[
\text{pred} \ \text{transClosure'} (R, r: \text{univ} \rightarrow \text{univ}, C: \text{univ} \rightarrow \text{Univ} \rightarrow \text{univ}) \{
\begin{align*}
\text{... axioms here}
\end{align*}
\}
\]

and edit the assertion to read

\[
\text{assert} \ \text{Equivalence}\{
\begin{align*}
\text{all } R, r: \text{univ} \rightarrow \text{univ}, C: \text{univ} \rightarrow \text{Univ} \rightarrow \text{univ} | \\
\text{transClosure'} [R, r, C] \implies R = *r
\end{align*}
\}
\]

Note that the check cannot be bidirectional. Can you see why? Replace \( \text{implies} \) by \( \text{iff} \) to generate a counterexample, and explain what’s going on.

A.1.10 Address Book Constraints and Expressions

In this exercise, you’ll get some practice writing expressions and constraints for a simple multilevel address book. Consider a set \( \text{Addr} \) of ad-