Transfer Learning Level Definitions

Timothy Hinrichs
Tyler Hicks-Wright
Charles Petrie
Eric Schkufza
Michael Genesereth

Abstract

Transfer Learning is a generalization of machine learning where instead of starting with no information when given the target task, the machine has already been able to learn in a source task. The degree to which the information gained by learning in the source is useful for the target is dependent on the relationship between source and target tasks. This paper proposes mathematical definitions for the relationships investigated in DARPA's Transfer Learning program.

Stanford Logic Group Technical Report LG-2007-01

Stanford Logic Group
Computer Science Department
Stanford University
353 Serra Mall
Stanford, CA 94305

1 Introduction

Knowledge acquired in one setting can often be employed to aid agents in another setting. The similarity of the settings has great impact on what knowledge can be transferred and the extent to which that knowledge is valuable. DARPA/IPTO has sponsored a program to investigate the idea of Transfer Learning, where agents are given a set of source problems and a set of target problems. The agent's performance on the target after playing the source is compared to the agent's performance on the target. Various metrics have been identified for comparing these two performance curves.

In the Transfer Learning program, the relationship between source and target problems that agents must contend with varies considerably, from problems that are almost identical to problems that have seemingly very little in common. The BAA describes ten such relationships, or transfer levels, that require an agent performing transfer at level ten to utilize far more sophisticated algorithms than are necessary to perform transfer at level one; however, in general there is no correspondence between levels and difficulty.

Those ten levels were described in english, thus allowing individual teams the flexibility to explore transfer learning in whatever domain they so choose; popular choices included Urban Combat and Physics simulations. The downside to english descriptions of levels is that tests of two different agents on the same transfer level in different domains produces results that are incomparable. The two teams might have interpreted the levels differently; they may have chosen particular problems that appeared to belong to the level in question but in fact did not; a team may have chosen problems that allowed an agent to perform transfer at a high level but also allowed an agent to perform transfer using algorithms that work at a lower level.

Mathematically defining transfer levels alleviates many of those problems. The existence of relationships between problems can be proven or disproven. Such proofs can ensure that source and target problems actually belong to the transfer level in question, and those proofs can be used to compare problems in different domains.

In this document, we begin relating our investigations into mathematically defining the class of finite, discrete, synchronous, multi-agent, deterministic problems and the relationships among them. Our approach is based on relational nets, which were developed to formalize such problems, and General Game Playing, an initiative aimed at providing a testbed for evaluating agent performance in such environments.

2 Relational Nets

Some of the problems we have considered in years one and two in DARPAs transfer learning program are finite, discrete, synchronous, multi-agent, and deterministic. Such a problem has a finite number of distinct states, time is broken up into discrete steps, the world only changes in response to agent actions (it does not change while the agent is thinking), every agent performs an action on every step of the game, and no part of the world has any randomness built in.

The model we use for such an environment is the relational net, a compact description of a finite state machine whose transitions are triggered by the actions of external agents. Recall that a state machine is defined formally as a 5-tuple $<S, \Sigma, \delta, s_0, F>$ where

- S is the finite set of states.
- Σ is the finite set of possible inputs (or agent actions).
- δ is the transition function that defines what state results when an input is received in a state. $\delta : S \times \Sigma \rightarrow S$

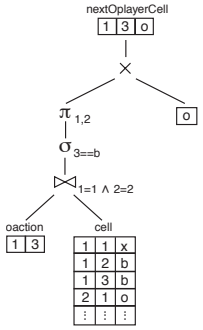

Figure 1: A fragment of the Tic-Tac-Toe relational net with a marking for all tables.

- s_0 is the initial state. $s_0 \in S$
- F is the set of terminal states. $F \subseteq S$

A finite state machine is a system that transitions from state to state based on external inputs. The system halts whenever the machine transitions into one of the terminal states.

In a relational net, a state has structure; it is represented as a database instance, i.e. a set of tables. The set of all states is the set of all possible database instances for the schema of the net. One table is designated for each agent to be writable; an agent acts by filling out that table. The transition function is represented as view definitions on top of the database schema. To compute the next state given the current state and players' actions, a set of distinguished views are materialized. The environment halts whenever the current state is one that is deemed to be terminal.

For example, tic-tac-toe is a particularly simple environment that can be modeled by relational nets. The schema consists of four tables named $cell$, $control$, $xaction$, $oaction$. There are two agents named $xplayer$ and $oplayer$. The $cell$ table always has nine rows: each one stores the value of a square on the board. Squares are identified by their x and y coordinates in the cartesian plane. The $control$ table always has one tuple with one value representing whose turn it is: $xplayer$ or $oplayer$. $xaction$ and $oaction$ are the tables that $xplayer$ and $oplayer$ can write, respectively. At each step, both agents act. When it is the agent's turn it is permitted to place a mark in any empty square, and when it is not the agent's turn it must $noop$.

Fig. 2 shows a part of the relational net for tic-tac-toe. The view $nextOplayerCell$ corresponds to the rule in tic-tac-toe that says if it is $oplayer$'s turn and it marks cell $\langle t, u\rangle$ then in the next state, $\langle t, u\rangle$ is marked with an o.

Each agent is restricted in what it can write into its table based on the current state. Each state is either terminal or not. Every state assigns each player an integer from 0 to 100, representing the goal value (or happiness) of the agent in that state. The initial state is also fixed.

All of this can be formalized mathematically, where the transition function corresponds to a diagram, and legality, termination, goals, and initial states are mathematical entities defined on top of that diagram.

Definition 1 (Relational Net). *A relational net is a structure* $\langle P, C, T, w, b, g, F, l, U, a, s\rangle$.

P: a finite set of predicates, including disjoint subsets I, O, S, and B, the input, output, static, and base predicates

C: a finite set of connectives, partitioned into C_{σ_e}, $C_{\pi_{mn}}$, $C_{\bowtie_n}$, and $C_\cup$

T: a finite set of transitions

w, *a wiring diagram of the following sort*

$$w : C_{\pi_s} \to P$$
$$w : C_{\sigma_e} \to P$$
$$w : C_{\bowtie_n} \to P$$
$$w : C_{\cup} \to P^n$$
$$w : T \to P$$
$$w : P - I - S - B \to C$$
$$w : B \to T$$
$$w : I \to nil$$
$$w : S \to nil$$

b*: a minimal relational marking, as defined below, the beginning state*
g*: a function from minimal markings to [0,100], the goal values*
F*: a set of minimal relational markings, the final states*
l*: a function from minimal markings to sets of partial markings for* I*, the legal inputs*
U*: a finite set, the universe of objects*
a*: a function from predicates to the natural numbers, the arities*
s*: a relational marking for predicates* S

P, C, T, *and* U *are mutually disjoint.*
There are no cycles through w *without transitions.*
The relational algebra operations are defined based on the arities of the relations they are mapped to.

[SHOULDN'T g be a function from minimal markings cross roles to [0,100]? If so, need to change the definition of Goal Preservation below. Also, should legality take roles into account?]

Table names correspond to predicates. Relational algebra operations correspond to connectives. Transitions distinguish the view that defines a table's contents in the next state. w wires the predicates, connectives, and transitions together.

Filling out a set of tables with data corresponds to a partial relational marking. Each table (named p) must consist of a set of tuples, each of the correct length ($a(p)$), whose elements are drawn from the universe (U).

Definition 2 (Partial Relational Marking). *For predicates* $p_1, \ldots, p_n$*, a partial marking is a function from each* p_i *to a subset of* $U^{a(p_i)}$.

A state corresponds to a relational marking for the base tables, which are defined as those predicates wired as the output of a transition.

Definition 3 (Minimal Relational Marking). *A minimal marking for a relational net is a partial marking for the base predicates* B.

A Relational Input Marking is when all of the agents have performed their actions, i.e. written their respective tables. A Relational Static Marking is a marking for all the static predicates.

Definition 4 (Relational Input Marking). *An input marking for a relational net is a partial marking for* $p_1, \ldots, p_n$*, where* $\{p_1, \ldots, p_n\} = I$.

Definition 5 (Relational Static Marking). *A static marking for a relational net is a partial marking for $p_1, \ldots, p_n$, where $\{p_1, \ldots, p_n\} = S$.*

Given a minimal marking, a static marking and an input marking, the marking for the remaining tables can be computed. We call this a Consistent Relational Marking.

Definition 6 (Consistent Relational Marking). *A consistent marking for a relational net with static marking s is a minimal marking m, an input marking i, and s extended to a full marking.*

Definition 7 (Marking Extension). *Define an extension via fixed point computation. Denote with $MarkingExtension[f]$. Given a marking of predicates f, if f marks every predicate in the net, then the marking extension of f is just f. Otherwise,*

for each predicate r in $P - I - S$ that is not marked (i.e. f is not defined on r) and
$w(r) = c$ is a connective and either $w(c)$ is marked or all of the $p \in w(c)$ are marked,
let t be the marking for r defined as follows
if c is C_{π_s} then $t = Project[f(c), s]$
if c is C_{σ_e} then $t = Select[f(c), e]$
if c is $C_{\bowtie_n}$ then $t = Join[mapcar(f, w(c)), n]$
if c is $C_{\cup}$ then $t = Union[mapcar(f, w(c))]$
set $f(r) = t$

Definition 8 (Relational Algebra Operations). *The relational algebra operations are defined using the unnamed perspective, i.e. using column numbers instead of names.*

- $Project[r, s]$, *where r is a relation, i.e. set of m-tuples $\langle e_1, \ldots, e_k \rangle$ and s is a set of indices $\{i_1, \ldots, i_n\}$. The projection is defined as the set of all $\langle e_{i_1}, \ldots, e_{i_n} \rangle$.*
- $Select[r, m = n]$, *where r is a set of tuples $\langle e_1, \ldots, e_k \rangle$ and m is an integer greater than 0 and less than or equal to k and n is either another number in the same range or an element in U. The selection is the set of all $\langle e_1, \ldots, e_k \rangle$ such that $e_m = n$ in the former case and $e_m = e_n$ in the latter.*
- $Select[r, m \neq n]$, *where r is a set of tuples $\langle e_1, \ldots, e_k \rangle$ and m is an integer greater than 0 and less than or equal to k and n is either another number in the same range or an element in U. The selection is the set of all $\langle e_1, \ldots, e_k \rangle$ such that $e_m \neq n$ in the former case and $e_m \neq e_n$ in the latter.*
- $Join[\{r_1, \ldots, r_n\}, m]$, *where each r_i is a set of tuples $\langle e_{i1}, \ldots, e_{ik_i} \rangle$ and each $k_i \geq m$. The join is the set of all tuples $\langle e_{11}, \ldots, e_{1k_1}, e_{21}, \ldots, e_{2(m-1)}, e_{2(m+1)}, \ldots, e_{2k_2}, \ldots, e_{n1}, \ldots, e_{n(m-1)}, e_{n(m+1)}, \ldots, e_{nk_n} \rangle$ such that $e_{1m} = e_{2m} = \cdots = e_{km}$.*
- $Union[\{r_1, \ldots, r_n\}]$ *is only applicable when all of the relations $r_1, \ldots, r_n$ have the same arity. The union is $r_1 \cup \cdots \cup r_n$.*

Given a marking for the base tables and an input marking, we can compute the next marking (state). First, it will be convenient in what follows to have definitions for the union of a function, and the reduction of a function's domain.

Definition 9 (Function Union). *The union of functions f_1 and f_2, denoted $f_1 \cup f_2$, is the minimum function f_3 such that*

$$f_1(x) = y \Rightarrow f_3(x) = y$$
$$f_2(x) = y \Rightarrow f_3(x) = y$$

Definition 10 (Function Domain Reduction). *Denote the domain of a function f with $Domain[f]$. Reducing the domain of a function $f : D \rightarrow R$ to D', which is a subset of D, produces the minimum function f' such that*

$$f' : D' \rightarrow R$$
$$f'(x) = f(x)$$

Denote with Reduce[f,D'].

Now it is simple to define the function that computes the next state when given the current state and an input marking.

Definition 11 ($next$). *For a relational net N, $next$ is a function that applies to a minimal marking m, an input marking i, and N. Recall that B is the set of base predicates in N.*

$$next[m, i, N] = Reduce[MarkingExtension[m \cup i \cup s], B]$$

Environments modeled by relational nets are fully-observable, finite, discrete, multi-agent, synchronous environments. Every agent can observe everything about the environment except what moves its opponents are going to make. The number of distinct states is always finite; multiple-agents work cooperatively, competitively, or in some combination to achieve their goals; and the environment changes only in response to agent actions, i.e. it does not change while the agents are deliberating over what move to make.

When looking at two environments, as is the case in the TL project, it can happen that some predicate appears in both environments. By forcing such a predicate to mean the same thing in both nets, we have the flexibility to make transfer for certain source/target pairs easier by reducing the disparity between the models of the two environments. We call this the Equivalent Commons Assumption.

Definition 12 (Equivalent Commons Assumption). *Two relational nets N_1 and N_2 obey the ECA if and only if*

for each c in both C_1 and C_2
$w_1(c) = w_2(c)$
for each t in both T_1 and T_2
$w_1(t) = w_2(t)$
for each r in both $P_1 - I_1$ and $P_2 - I_2$
$a_1(r) = a_2(r)$
$w_1(r) = w_2(r)$

Notice that the ECA does not force the initial markings or the static markings to be the same for the same predicates.

3 Composition

New problem instances consist of combinations of components from distinct component sets encountered during training.

A combination of components, such as combining chess and checkers, results in a game where the union of the rules for the two games form the rules for the composite game, e.g. in the combined game chess pieces move like chess pieces and checkers pieces move like checkers pieces. Conditions under which the combined rules are consistent must be observed.

Three definitions for composition appear here. The first says just that the rules of the two source games (physics for the first two nets) are combined to produce the rules for the third game (physics for the third net). There are no constraints on the beginning state, the goals, legality, or final states. The second is a specialization of the first that put constraints on the initial state–it requires the initial state of the combined game to be the union of the initial states of the first two games.

Definition 13 (Physics Composition 1). *The three relational nets*

$$N_1 = \langle P_1, C_1, T_1, w_1, b_1, g_1, F_1, l_1, U_1, a_1, s_1 \rangle$$
$$N_2 = \langle P_2, C_2, T_2, w_2, b_2, g_2, F_2, l_2, U_2, a_2, s_2 \rangle$$
$$N_3 = \langle P_3, C_3, T_3, w_3, b_3, g_3, F_3, l_3, U_3, a_3, s_3 \rangle$$

belong to Physics Composition 1 if and only if N_1 and N_2 obey the ECA, N_3 has no cycle in w without a transition, and

$P_3 = P_1 \cup P_2$
$I_3 = (I_1 - (P_2 - I_2)) \cup (I_2 - (P_1 - I_1))$
$C_3 = C_1 \cup C_2$
$T_3 = T_1 \cup T_2$
$w_3 = w_1 \cup w_2$
l_3 *is defined as follows.*
  l'_1 *is* l_1 *after reducing the domain of each* I'_1 *in* l'_1 *to* $I_1 - (P_2 - I_2)$
  l'_2 *is* l_2 *after reducing the domain of each* I'_2 *in* l'_2 *to* $I_2 - (P_1 - I_1)$
  l_3 *is the set of all* $I'_1 \cup I'_2$ *such that* $I'_1 \in l'_1$ *and* $I'_2 \in l'_2$
$U_3 = U_1 \cup U_2$
$a_3 = a_1 \cup a_2$

This is a very general form of physics composition since there are no constraints on l, b, g, F, or s. It essentially guarantees that the wiring diagram of the target is the union of the wiring diagrams of the two source games, i.e. that the physics of the target is the combination of the physics of the two sources. In addition to combining the physics of two nets, it is often natural to combine the initial states of two nets through union. The only problem arises when there is some predicate p shared by the two nets where the initial value for p in the two nets is not the same. When that is the case, we simply union the tuples.

Definition 14 (Physics Composition 2). *The three relational nets*

$$N_1 = \langle P_1, C_1, T_1, w_1, b_1, g_1, F_1, l_1, U_1, a_1, s_1 \rangle$$
$$N_2 = \langle P_2, C_2, T_2, w_2, b_2, g_2, F_2, l_2, U_2, a_2, s_2 \rangle$$
$$N_3 = \langle P_3, C_3, T_3, w_3, b_3, g_3, F_3, l_3, U_3, a_3, s_3 \rangle$$

belong to Physics Composition 2 if and only if $\langle N_1, N_2, N_3 \rangle$ belong to Physics Composition 1 and

$$b_3(r) = \left\{ \begin{array}{l} b_1(r), \textit{ if } r \notin P_2 \\ b_2(r), \textit{ if } r \notin P_1 \\ b_1(r) \cup b_2(r), \textit{ otherwise} \end{array} \right\}$$

Static tables are often important and can be used to represent the initial state, e.g. walls in a maze that never change are often implemented as static tables. Definitions for composition that union the static tables in the two source nets can be defined analogously to the union of the initial states.

4 Differing

Levels 7-9 are special cases of Differing; hence, we start with its definition.

> *Level 10 (Differing): New problem instances bear minimal apparent overlap with the training instances; however, the problem solutions use some common knowledge or problem-solving strategies.*

The central concept needed to define Differing is that of a predicate mapping: a mapping from the predicates in one relational net to the predicates in the other relational net. We have chosen relational algebra as the language for writing these mappings. Thus, p in one relational net might correspond to the natural join of q and r in the other relational net. Since relational algebra is already embedded in the definition of a relational net, we have chosen to define a predicate mapping from one net to another as an extension of the first net so that the predicates of the second net are defined.

The intuition is that a mapping allows us to compute from some state in one net the corresponding state in the other; that means that given the base predicates of the first, we can compute the base predicates of the second. Thus the definition for the base predicates in the second net should not rely on the input predicates. Also, we want to map the input predicates from one net to the other, and this mapping should not rely on the state, i.e. on any of the base predicates or the views defined on top of them. These constraints are embodied in the following definitions.

Definition 15 (Defined in terms of). *A predicate p in a relational net is defined in terms of another predicate q if and only if*

- *p is q or*
- *$w(p) = c$ and*
  (($w(c) = r$ and r is defined in terms of q) or
  ($w(c) = \langle r_1, \ldots, r_n \rangle$ and for some i r_i is defined in terms of q))

Definition 16 (Predicate Mapping). *Suppose the two relational nets*

$$N_1 = \langle P_1, C_1, T_1, w_1, b_1, g_1, F_1, l_1, U_1, a_1, s_1 \rangle$$
$$N_2 = \langle P_2, C_2, T_2, w_2, b_2, g_2, F_2, l_2, U_2, a_2, s_2 \rangle$$

obey the ECA. A predicate mapping from N_1 to N_2 is a relational net $\langle P, C, T, w, b, g, F, l, U, a, s \rangle$ such that

- $P \supseteq P_1 \cup I_2 \cup B_2$
- $I = I_1$
- $S \supseteq S_1$
- $C \supseteq C_1$
- $T \supseteq T_1$

- $U \supseteq U_1$
- *for all* $p \in P$*, if* $a_1(p)$ *is defined then* $a(p) = a_1(p)$ *and if* $a_2(p)$ *is defined then* $a(p) = a_2(p)$
- *for all* $p \in S$*, if* $s_1(p)$ *is defined then* $s(p) = s_1(p)$
- w *obeys the following constraints*

  if $w_1(t)$ *is defined then* $w(t) = w_1(t)$
  if $t \in I_2$ *then* $w(t)$ *is not defined in terms of any* $p \in P_1$ *except* I_1 *otherwise,* $w(t)$ *is not defined in terms of any* $i \in I$

A predicate mapping from N_1 to N_2 allows us to compute from a marking (state) of N_1 what the marking (state) for N_2 would be. We denote the function that from a minimal marking in the first net computes the minimal marking of the second net according to N with h_N.

Definition 17 (h_N). *Suppose* N *is a predicate marking from net* N_1 *to net* N_2 *and* m_1 *is a minimal marking for* N_1*. Then* $h_N(m_1) = Reduce[MarkingExtension[m_1 \cup s], B_2]$.

Likewise, given an input marking for one net and a predicate mapping from that net to another, we can compute the corresponding input marking for the second net.

Definition 18 (i_N). *Suppose* N *is a predicate marking from net* N_1 *to net* N_2 *and* m_1 *is an input marking for* N_1*. Then* $h_N(m_1) = Reduce[MarkingExtension[m_1 \cup s], I_2]$.

Notice that the universe of a predicate mapping can be an expansion of the original net. That expansion could, for example, include all the objects in the second net. Representing a mapping between objects in the two nets can be accomplished by introducing new static tables that represent those mappings. Each predicate can be mapped independently from every other predicate, and for each such mapping, the object mapping can be different.

A predicate mapping is by itself uninteresting since every pair of relational nets has many different possible mappings. For example, one relational net can itself be considered a predicate mapping to any other net: every state in the first corresponds to the empty state in the second. Thus, two relational nets belong to TL level Differing whenever the predicate mapping preserves a nontrivial property, i.e. a property that some net has and some net does not.

Definition 19 (Differing). *Given two relational nets* N_1 *and* N_2*,* N_1 *and* N_2 *belong to the level Differing if and only if there is a predicate mapping from* N_1 *to* N_2 *that preserves a nontrivial property about relational nets, i.e. a property that some nets have and some nets do not.*

5 Reformulation

Levels 7-9 are special cases of Differing, i.e. each is defined as a predicate mapping that preserves some particular property.

> *Level 9 (Reformulation): New problem instances are recognized as similar to training instances only when they are reformulated through a well-specified transformation.*

A relational net is a compact mathematical model of a finite state machine (FSM). The important features of a FSM are (1) a set of states S, (2) a transition alphabet Σ, (3) a function δ that given a state and an action returns the resulting state, (4) the beginning state s_0, (5) the final states F, and (6) the goal values for each state, γ. The Reformulation transfer level contains all pairs of relational nets such that there is a predicate mapping between them that preserves the beginning state, the goals, the final state, legality, or δ. Thus, before defining Reformulation, we must define what it means for a predicate mapping to preserve each one of these features.

The following definitions all apply to relational net N_1, relational net N_2, and a predicate mapping N from N_1 to N_2.

$$N_1 = \langle P_1, C_1, T_1, w_1, b_1, g_1, F_1, l_1, U_1, a_1, s_1 \rangle$$
$$N_2 = \langle P_2, C_2, T_2, w_2, b_2, g_2, F_2, l_2, U_2, a_2, s_2 \rangle$$
$$N = \langle P, C, T, w, b, g, F, l, U, a, s \rangle$$

The beginning state is preserved whenever after applying the mapping to the beginning state of the first net, the result is the beginning state of the second net.

Definition 20 (Beginning State is preserved). *The predicate mapping N from net N_1 to net N_2 preserves the beginning state if and only if $b_2 = h_N(b_1)$.*

Goals are preserved whenever for every minimal marking (state) in the first net, the goal of the first net after applying the mapping is the same as the goal of the second net.

Definition 21 (Goals are preserved). *The predicate mapping N from net N_1 to net N_2 preserves the goals under the following conditions.*

for every minimal marking m_1 of N_1
$$g_1(m_1) = g_2(h_N(m_1))$$

Final states are preserved whenever the final states of the first net under the mapping are a subset of the final states of the second net.

Definition 22 (Final states are preserved). *The predicate mapping N from net N_1 to net N_2 preserves the final states under the following conditions.*

for every $f_1 \in F_1$
there is some $f_2 \in F_2$ such that
$$f_2 = h_N(f_1)$$

Legal moves are preserved whenever for every minimal marking (state) in the first net, the legal moves that are available are a subset of the legal moves available in the second net after mapping both the minimal marking and the legal moves.

Definition 23 (Legality is preserved). *The predicate mapping N from net N_1 to net N_2 preserves legality under the following conditions.*

for every minimal relational marking m_1 for N_1
for each $e_1 \in l_1(m_1)$
$$i_N(e_1) \in l_2(h_N(m_1))$$

The transition function δ, which maps a state and an action into the next state, is preserved whenever every minimal marking and legal action combination where the legal action under the mapping is legal in the second net, the next state in the first net under the mapping is the same state produced in the second net if the next state is computed after applying the mapping to the minimal markings and legal move.

Definition 24 (δ is preserved). *The predicate mapping N from net N_1 to net N_2 preserves δ under the following conditions.*

for every minimal relational marking m_1 for N_1
for every legal marking $e_1 \in l_1(m_1)$
if $i_N(e_1) \in l_2(h_N(m_1))$ then
$$h_N(next[m_1, e_1, N_1]) = next[h_N(m_1), i_N(e_1), N_2]$$

Reformulation requires one of the properties be preserved.

Definition 25 (Reformulation). *Relational net N_2 is a reformulation of relational net N_1 if and only if there is a predicate mapping from N_1 to N_2 that preserves the beginning state, goals, final states, legal moves, or δ.*

6 Abstraction

Level 7 (Abstraction): New problem instances are similar to training instances only when a particular abstraction is applied.

The intuition behind abstraction is that information is being lost when we move from one relational net to the abstraction of that net but that everything is preserved once that information loss is taken into account.

For example, an abstraction of the natural numbers and addition might be the set $\{even, odd\}$, where addition is defined as follows.

$$even + odd = odd$$
$$even + even = even$$
$$odd + even = odd$$
$$odd + odd = even$$

Information is lost since instead of countably many objects we have two, but addition is preserved. In model theory, this would be defined as a homomorphism, i.e. a mapping between objects that preserves the functions and relations.

In terms of relational nets, an abstraction of one net should operate the same as the original net, except certain information is missing. That means that there is some predicate mapping that preserves the beginning state, goals, final states, legality, and δ.

Definition 26 (Abstraction 1). *Relational net N_2 is an abstraction of N_1 if there is a predicate mapping from N_1 to N_2 that preserves the beginning state, goals, final states, legal moves, and δ.*

This form of abstraction is therefore a special case of Reformulation. While not yet proven, we believe a $\wedge bgFld$ mapping from net N_1 to net N_2 ensures there is a homomorphism from the game graph of N_1 to the game graph of N_2. The graphs need not be isomorphic, however, since there may be more states for N_1 than for N_2.

Another form of abstraction allows information to be lost from both the source and the target. In this case, the target is not an abstraction of the source, and the source is not an abstraction of the target, but there is some abstraction shared by the source and the target.

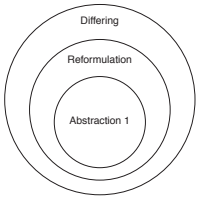

Figure 2: A Venn diagram for Abstraction 1, Reformulation, and Differing.

Definition 27 (Abstraction 2). *Two relational nets N_1 and N_2 share an abstraction if and only if there is some relational net N that is an abstraction of N_1 and an abstraction of N_2.*

It is unclear how Abstraction 2 relates to Reformulation and Differing. But it is clear how Abstraction 1, Reformulation, and Differing relate; their relationships appear in Fig. 6.

7 Generalization

Level 8 (Generalization): New problem instances involve principles drawn from a superset of the distribution or generating process of the training distribution or generating process.

The SLG has not reached consensus on how Generalization differs from Abstraction. What follows is a contentious argument that concludes the two are the same. Arguments that differentiate Generalization and Abstraction are solicited.

The canonical example of generalization comes up in algorithm design. If algorithm A works for some set of cases C and algorithm A' works in a superset of those cases, then we say that A' is more general than A. For example, A might be a sorting algorithm that works exactly when the list of numbers is in reverse sorted order; A' might be a typical sorting algorithm that works in all cases. A' is a more general solution than A.

The canonical example for abstraction comes from hardware design. An ALU is composed of an arithmetic unit and a logic unit. Considering the ALU as a black box with input and output specs is an abstraction of the input/output specs of the arithmetic unit, the input/output specs of the logic unit, and the wiring that hooks them together. That is, we can reason about the ALU without knowing how it is implemented.

If an algorithm A solves a problem P then a generalization of A is required to solve an abstraction of P.