Comirit: Commonsense Reasoning by Integrating Simulation and Logic

Benjamin JOHNSTON and Mary-Anne WILLIAMS
Faculty of Information Technology, University of Technology, Sydney, Australia

Abstract. Rich computer simulations or quantitative models can enable an agent to realistically predict real-world behavior with precision and performance that is difficult to emulate in logical formalisms. Unfortunately, such simulations lack the deductive flexibility of techniques such as formal logics and so do not find natural application in the deductive machinery of commonsense or general purpose reasoning systems. This dilemma can, however, be resolved via a hybrid architecture that combines tableaux-based reasoning with a framework for generic simulation based on the concept of 'molecular' models. This combination exploits the complementary strengths of logic and simulation, allowing an agent to build and reason with automatically constructed simulations in a problem-sensitive manner.

Keywords. Commonsense reasoning, simulation, tableaux methods

Introduction

Comirit is the name of both a software system and a corresponding research project that seeks to enhance robotic and software agents with commonsense awareness and reasoning capabilities. The *Comirit* project is exploring the feasibility of rapidly constructing intelligent systems from existing technologies—that is, *Comirit* is a response to the challenge of engineering commonsense-enabled systems from scratch, on minimal budgets and in short time-frames. The objective of this paper is to introduce the core integrative architecture of *Comirit*, in the context of the inspiration of the architecture: the problem of integrating simulation and logic. While the objective of *Comirit* is nominally commonsense reasoning, the project has clear connections to artificial general intelligence and this project is intended as the first in a series of efforts aiming to pragmatically engineer, within a resource-constrained context, systems that are capable of increasingly more general reasoning and learning.

We see an opportunity in identifying well-established and robust techniques within computer science and integrating these in novel combinations that, in harmony, afford powerful new modes of reasoning. In particular, our early efforts have primarily focused on the compatibility of simulation and logical reasoning:

1. While simulations and computer games do not, in themselves, demonstrate commonsense intelligence or enable complex reasoning, they do provide extremely rich and realistic models of 'commonsense' scenarios and behavior.
2. Computable expressive logics, in contrast, allow for flexible and powerful modes of reasoning, but lack the rich models of commonsense situations often found in simulations.

The intent of combining these two methods is to unite the flexibility and power of expressive logics with the richness of simulation, and ideally to do so in a way that remains open to later inclusion of other forms of reasoning.

It turns out that tableaux approaches to automated reasoning provide an effective basis for successful integration of logic and simulation. Tableaux systems are constructive methods for finding contradictions—they attempt to find possible models or worlds in which a logical formula is inconsistent. It is these worlds that form the crux of integration: rather than merely limiting the tableaux search to worlds that lead to logical inconsistency, these possible worlds can also be tested for inconsistency under simulation (or, in fact, inconsistency under a wide range of techniques).

Unfortunately, the technicalities of implementing such an integrated system can overwhelm the simplicity of this idea. In this paper we describe a strategy for managing this complexity: we generalize the basic principles of tableaux systems so that a branch of the tableaux can contain not just logical terms but also simulation data, notes, functions and even the tableaux expansion rules.

In the following two sections (Section 2 and Section 3), we comprehensively motivate the architecture and briefly review the preliminaries. We then describe the conceptual details of the architecture in Section 4. In Section 5 we describe our experiences with the prototypes that we have built, concluding with a brief discussion of future directions in Section 6.

1. Background

While our objective is commonsense intelligence, we do not intend to belabor the philosophical question of what precisely is meant by ‘commonsense’ or ‘intelligence’. Generally speaking, our intent is to develop software and robotic agents with sufficient know-how to appropriately respond to novel problems caused by the open-ended constraints of the real-world. That is, our emphasis is not on capturing commonly known factual knowledge (“Who is the Queen of England?”), but to create systems with real-world ‘know-how’ (“How can I safely rescue this person?”).

In lieu of a rigorous definition of commonsense intelligence, we make use of benchmark problems and analyze system performance in applied situations to evaluate and motivate our work. Morgenstern [1] provides a range of commonsense benchmark problems, contributed by many researchers, that involve naïve or commonsense knowledge about matters including planning, physics and psychology. While many of these ‘challenge problems’ require general purpose problem solving skills, they are presented within restricted domains and with only moderate complexity such that they are useful for benchmarking formalisms, theories and components (as opposed to entire systems). Entire systems may be benchmarked in realistic open-ended scenarios such as competitions like RoboCup Rescue, RoboCup @ Home and the DARPA Grand Challenges. Physical and spatial reasoning play a significant part in all of these problems, however such problems also include aspects of naïve human psychology, economics, game theory, agent behavior, planning and potentially general purpose problem solving. Existing approaches to these benchmarks and challenges tend to rely on significant human engineering: either in the form of large scale manual knowledge elicitation (*e.g.*, [2,3]) or by manually engineering implicit (and task specific) commonsense ‘know-how’ into the low-level faculties of the robot architecture (*e.g.*, [4]). The effort required and the brittleness of these existing approaches is unacceptable for our purposes.

We observe, however, that modern simulations—computer games, computer animations and virtual worlds—have increasingly detailed, life-like environments that sometimes resemble the very situations described in benchmark problems. While simulations are currently difficult to directly exploit as a resource for commonsense reasoning, they do present a rich resource of implicit commonsense 'know-how'. Some projects have attempted to exploit this resource indirectly [5]—placing an agent within a simulated environment to explore and learn about human settings free of the wear, cost, time and concurrency constraints of the real world. Such approaches are interesting, but are limited by progress in machine learning and the ability to automatically represent, generalize and specialize knowledge acquired from simulated experiences. In contrast, our initial objective is to *directly* exploit simulation as a representation and reasoning mechanism (rather than merely as a training environment).

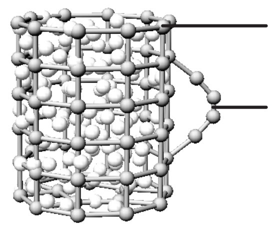
Of course, with (typically) only a forward mode of reasoning and many critical limitations, simulations are not immediately useful as a mechanism for commonsense reasoning. However, in combination with a suitable automatic reasoning system for an expressive logic, these limitations can be avoided. Proof search facilities can be used to reverse the natural 'arrow of time' by postulating possible causes and then simulating to test whether the outcomes match observations. Such proof search facilities can also be expanded to generate the necessary detail to allow partially defined and unground objects to be numerically simulated, and search facilities may even be used to manage the process of converting symbolic scene descriptions into numerical simulations. Furthermore, the reasoning system can solve those sub-problems that are poorly suited to encoding in simulations, such as abstract logical deduction and random-access recall of simple factual data.

In principle, if we are integrating simulation and logic, we have many options for the choice of underlying technologies; the selection of which can have a dramatic influence on the capabilities and ease of integration of the complete hybrid system. Although it is entirely possible (and effective) to simultaneously use a range of simulation engines in the pluggable hybrid architecture we are proposing in this paper, our preference is for a single simulation platform with sufficient flexibility to model the vast range of scenarios that an agent may encounter. The Slick architecture [6]—a 'molecular' or 'ball-and-stick' approach to simulating a wide range of physical and non-physical phenomena—suits this criteria and has, in fact, been designed specifically for commonsense reasoning. Similarly, successful integration requires a reasoning mechanism that is efficient and flexible and that provides suitable 'scaffolding' upon which other methodologies may be added. Tableaux based reasoning systems are ideal in this regard: they have demonstrated efficiency as the basis of modern Semantic Web reasoners [7], and their search strategy is based on the construction of counter-models or counter-worlds to which simulations can be attached. The following section includes a brief review of both Slick simulation and tableaux based reasoning in the context of hybrid commonsense reasoning.

2. Preliminaries

2.1. Slick Simulation

The Slick architecture [6] is a general purpose approach to simulation designed for commonsense reasoning systems (for related approaches see [8,9]). Slick is designed for simulating a wide range of phenomena including domains as diverse as physical solids and liquids, naïve psychology and economic behavior. In Slick, the state of a simulation is represented as a hypergraph in which every vertex and hyperedge is annotated with a frame. A set of functions perform iterative stepwise update to the state of the simulation during



Entity id:#37
x: 3.452 *y*: 0.671 *z*: 6.332 *mass-density*: 2.1
rigidity: 51 *type*: {*matter*, *solid*, *porcelain*}

Join id:#189 **connecting Entities**: {id#56, id#57}
broken: **false** *flex*: 3% *spring-constant*: 9000
type: {*matter*, *solid*, *porcelain*}

Constraint id:#5
name: gravity *applies-to*: **Entity** *target-type*: {*matter*}
function: $z := z + \frac{1}{2} \times g \times (\Delta t)^2$

Figure 1. Sample Entity, Join and Constraint illustrating the frame-like data structures of a Slick simulation.

each tick of a global, adaptive[1] clock. With appropriate update functions and annotations, Slick can be applied to any environment where *global* behavior is governed by or can be predicted by *local* laws or rationality assumptions. For example, in the case of physical environments: the hypergraph is structured to approximate the shape of physical objects; annotations are used to describe the local physical properties of the object (the position of vertex, local weight density, local temperature, appearance, *etc.*); and update rules correspond to discrete-time variants of the laws of Newtonian mechanics. Consider Figure 1 as an example of how Slick might be used to represent a cup of coffee.

Note that this method of simulation is characteristically non-symbolic. We do not need to specify how a cup of coffee behaves, but derive its behavior from the interactions and forces that occur over the simplified 'molecular' structure. That is, macroscopic properties and behaviors emerge from simple microscopic update rules, simplifying the knowledge engineering process and maximizing the generality of the technique.

We have previously demonstrated [6] how this simple architecture can be implemented in a concrete system and applied to established benchmark problems such as the Egg Cracking problem [10]. A concrete implementation of the Slick architecture includes the following three critical classes/types:

1. **`Entity`**. An annotated vertex with operations for vertex-specific parameters to be stored and retrieved by attribute name.
2. **`Join`**. An annotated hyperedge (with a set of end vertices) with operations for hyperedge-specific parameters to be stored and retrieved by attribute name.
3. **`Constraint`**. A stepwise simulation update function that queries the current simulation hypergraph and updates the annotations, in addition to possibly forking the simulation or invalidating the simulation if either multiple or no valid future-states exist.

The Slick architecture, as originally proposed, also incorporates a control language for instantiating and manipulating simulations, in addition to a database mechanism that stores generic models for instantiation and manages the relationship between abstract symbols and the simulation hyper-graph. These not only enable the integration of simulation and symbolic methods, but allow for simulations to be automatically constructed from purely symbolic queries. We will see in Section 3 that while the database and control language remain essential in a hybrid architecture, they are subsumed and generalized by the use of expressive logics.

2.2. Tableaux Reasoning

The method of analytic tableaux is an approach to automatically proving (or disproving) the truth of a logical statement by searching for models or worlds in which the negation of the logical statement is satisfiable. The technique is over 50 years old but has experienced

[1] Clock rate is variable and adapts to ensure numerical precision remains within acceptable bounds.

Table 1. Basic tableaux rules.

Rule	Condition		Action	Explanation
R1a:	$\{A \wedge B\}$	$\rightarrow$	$\{A\}$	(extend the branch)
R1b:	$\{A \wedge B\}$	$\rightarrow$	$\{B\}$	(extend the branch)
R2:	$\{A \vee B\}$	$\rightarrow$	$\{A\}, \{B\}$	(fork into two child branches)
R3:	$\{A, \neg A\}$	$\rightarrow$	$*$	(close the branch due to contradiction)

a recent surge in interest because of its applicability to modal logics, description logics and Semantic Web reasoning. A thorough overview of the method and applications to reasoning with propositional, first order, modal, description and other logics can be found in D'Agostino *et al.*'s (eds) handbook [11] or other standard references for formal computer science. For convenience, we present a brief review of the technique for the propositional case below.

A tableaux system is a method of proof by contradiction: it proves validity by showing that the negation of a formula is unsatisfiable. The method works by updating a tree structure whose nodes are labeled with logical formulae. The algorithm starts with a negated formula as the root of a tree and repetitively applies a set of update rules that close or expand branches of the tree as appropriate. The basic algorithm is simple: when a branch contains a disjunction, the branch is forked into two child branches corresponding to each of the disjuncts; when a branch contains a conjunction, each conjunct is added to the leaf of the branch. The effect is that a formula is converted into a tree where the parent-child relationship can be read conjunctively and the sibling relationship can be read disjunctively. Because the parent-child relationship is read conjunctively, we look along branches (paths from the root to a leaf along the ancestor/descendent axis) for contradiction. If every branch contains a contradiction, then there is no consistent counter-model and therefore the original un-negated formula must be valid. The update rules of this basic algorithm are summarized in Table 1.

The following example illustrates how the tableaux method is used to reason that $((a \vee b) \wedge \neg a) \Rightarrow b$ is universally valid.

To show that $((a \vee b) \wedge \neg a) \Rightarrow b$ is valid, we negate it: $\neg(((a \vee b) \wedge -a) \Rightarrow b)$, simplify to negation normal form: $(a \vee b) \wedge \neg a \wedge \neg b$, and then place it at the root of a tree. We then attempt to show that this negated formula at the root of the tree is unsatisfiable by applying the basic tableaux rules until all branches can be closed. Figures 2a–2c illustrate the results of the iterative rule application.

A contradiction can be seen along the left-hand branch of the completed tableau: node 5 is an ancestor of node 6 so should be read conjunctively, however $\neg a$ and a obviously

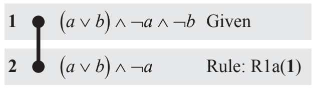

1
$(a \vee b) \wedge \neg a \wedge \neg b$
Given
2
$(a \vee b) \wedge \neg a$
Rule: R1a(**1**)

Figure 2a. Tableau after the first step (applying rule R1a on the first node)

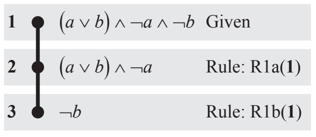

1
$(a \vee b) \wedge \neg a \wedge \neg b$
Given
2
$(a \vee b) \wedge \neg a$
Rule: R1a(**1**)
3
$\neg b$
Rule: R1b(**1**)

Figure 2b. Tableau after the second step (applying rule R1b on the second node).

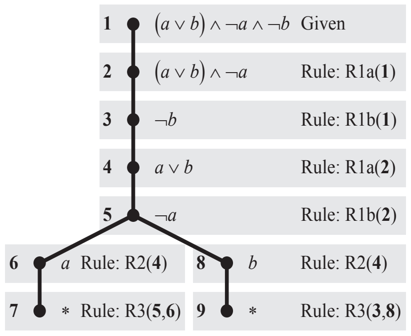


Figure 2c. Completed tableau for $((a \vee b) \wedge \neg a) \Rightarrow b$ (reached after eight steps).

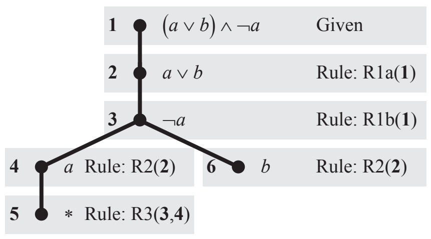


Figure 3. Completed tableau for $(a \vee b) \Rightarrow a$.

form a contradiction. Similarly, node 3 is an ancestor of node 8 and when read conjunctively, trivially forms a contradiction. All branches are contradictory and closed, therefore the original formula is valid.

Similarly, consider the effect if we begin with a formula that is not valid, such as: $(a \vee b) \Rightarrow a$. We first negate it: $\neg((a \vee b) \Rightarrow a) \equiv (a \vee b) \wedge \neg a$, and apply the tableaux rules, then we obtain a completed tableau per Figure 3.

A contradiction can be seen along the left-hand branch: node 3 (a) is an ancestor of node 4 ($\neg a$) so they should be read conjunctively, but are in contradiction. The right-hand branch, however, remains open with neither contradictions nor the possibility of applying a rule (without repeating). In fact, the right-hand branch (when read conjunctively) is a counter model for our original formula: $(a \vee b) \Rightarrow a$ does not hold when we have $\neg a$ and b. The tableaux method is an efficient way of searching for such counter models.

We have assumed conversion into negation normal form (NNF) and the three operators $\wedge$, $\vee$ and $\neg$. However, by extending the tableaux system with rules for NNF conversion, it is possible to accept any formula of propositional logic. Through the addition of yet further rules (and additional types of structures), a range of other operators and logics can be supported [11].

We have provided only a brief overview, ignoring a range of issues concerning rule selection, search strategy, heuristics and optimizations, quantification and modalities, cycles, managing repeated rule application, completeness, correctness and the complexities of undecidable logics (such as first order logic). While these matters are highly significant in any domain, including applications for commonsense reasoning, we have not yet had the opportunity to comprehensively consider every issue but have designed a system that through configurability (and modularity) of tableaux rules, meta-strategies and data-types remains agnostic to the particulars of a given tableaux algorithm. However, the critical observation here is that tableaux systems can be made very efficient, can support a wide range of logics and, as we will see in the following section, elegantly interface with other forms of 'reasoning' such as simulation.

3. Integration

In combining simulation and logic, we hope to create a system greater than merely the sum of its two parts. In combination, the two have the capability for a powerful union: logic unlocks the full potential of the implicit 'know-how' in simulations, and simulation dramatically enhances the capabilities and real-life applicability of logic. However, in creating a hybrid system we must carefully address the mismatch between paradigms and the inherent increase in complexity that the integration introduces. Furthermore, our goal is to create an open-ended system that can admit other technologies and thereby develop sophistication over time. Aside from our primary motivation of creating systems with greater commonsense awareness, we are therefore guided by three important design objectives: cohesion, simplicity and open-endedness. Cohesion is achieved by using a well defined interface between tableau reasoning and simulation, described in Section 3.1. This interface is implemented using highly regularized data-structures, described in Section 3.2, that maximize simplicity and open-endedness.

3.1. Tableaux-Simulation Interface

Clear semantics assist in maximizing the cohesion in the integration of simulation and tableaux based reasoning. Without a clear and unifying abstraction, the mapping between paradigms can be highly convoluted, requiring complex ad-hoc code to monitor, analyze and translate data-structures and data-updates between representations. Instead, a single, well-defined coupling is preferred, such as with the concept of possible worlds.

Both simulation and tableaux reasoning can be seen as processes concerned with dividing and subsequently ‘closing’ the space of possible worlds. In tableaux-based reasoning systems, broad regions of the space of possible worlds are divided and closed by restrictions with logical prepositions. Simulations, in contrast, work only on narrowly defined spaces of possible worlds, but also ultimately divide and ‘close’. This correspondence forms the basis of our integration strategy, and is more clearly illustrated by way of example. Consider the following problem:

> *A commonsense enabled real-estate matching system knows that John values his time but doesn’t have a car: he cycles or catches a bus depending on the weather. A convenient home is less than 15 minutes by bus (60km/hour) and by bicycle (12km/hour) from his workplace. We want to know whether a home 3km from work or 4km by the bus’s circuitous route is convenient.*

We assume a hybrid system, constructed from simplified components:

- A tableaux reasoner using rules for unquantified predicate logic, equality (=), simple relations of order (<,≤) and a rule for expanding a predicate from its definition (≡).
- A simple numerical ‘simulation’ of the process of commuting to work via different mechanisms. For illustrative purposes we use the highly simplified and abstracted simulation of Figure 4.

In the context of this system, we might then encode the sample problem as the logical formula $(bus \vee bike) \Rightarrow conv$, where $bus \equiv speed{=}60 \wedge distance{=}4$, where $bike \equiv speed{=}12 \wedge distance{=}3$ and where $conv \equiv duration{\leq}15$.

We negate the formula and convert it to negation normal form: $(bus \vee bike) \wedge \neg conv$, and apply the tableaux rules until no more rules can be applied (*i.e.*, the tableau has ‘stalled’), resulting in the tableau of Figure 5a. Since open branches remain, we attempt simulation: each branch is read as a narrowly defined space of possible worlds, and the simulation engine is accordingly invoked on each branch. Doing so expands the tableaux with the output of the simulation engine per Figure 5b.

Inputs	*speed, distance*
Outputs	*duration*
Algorithm	`set` x := 60 × *distance* ÷ *speed* `return` x `as` *duration*

Figure 4. Simplified simulation algorithm. Note that while highly simplified, this algorithm has similar constraints to real-life simulations: the inputs are assumed to be numerical and fully specified; and the algorithm can only be used in the ‘forward’ direction (that is, it cannot be directly used to compute speed, given the duration and distance).

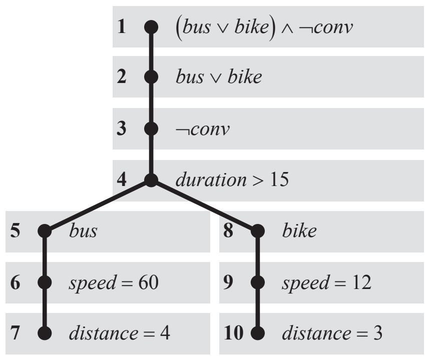


Figure 5a. Stalled tableau
Note that nodes 4, 6, 7, 9 and 10 were created by expanding (from definition) $\neg$*conv*, *bus* and *bike*.

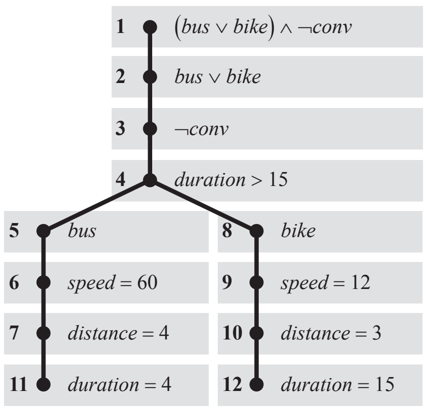


Figure 5b. Tableau after simulation.

Finally, we can revert back to the tableaux rules, and close both branches using the contradictions that occur with *duration* in nodes 4 ($duration > 15$) and 7 ($duration = 4$) in the left-hand branch and in nodes 4 ($duration > 15$) and 7 ($duration = 15$) in the right-hand branch. Because all branches can be closed, we have proven the original query: $(bus \vee bike) \Rightarrow conv$, and have therefore shown the house to be 'convenient'.

The general execution strategy of the *Comirit* hybrid system follows exactly this process of alternating tableaux rule application and simulation. However, practical complexities are introduced by the non-determinism that we have left implicit. It is often possible to apply both simulation and tableaux rules at any given point in time, and there may in fact be multiple applicable tableaux rules for the tableaux system. Sophisticated heuristics may be developed to guide the execution strategy—our research prototypes generate simple cost-estimates and select based on a greedy cheapest-first strategy, but more mature techniques such as a measure of expected benefit could be used. In the following section we outline the generic architecture which allows for pluggable replacement of selection strategies.

Integrating Slick simulation into this framework presents its own challenges. A Slick simulation has potentially millions of outputs—the value of every attribute of every Entity or Join at every point in time—including these as logical terms in the tableau would incur significant computational and memory costs for little benefit. Instead, additional *linking Constraints* are added to the Slick simulation to manage the connection between symbolic and 'molecular' attributes of objects: at each tick of the clock, they calculate the state of an object in aggregate over its 'molecular' representation and generate corresponding abstract symbolic terms for inclusion in the tableau. For example, an 'object broken' *linking Constraint* might report that an object is broken at a given point in time if any of its Joins have 'snapped' at that time. Such constraints can be added to every simulation, however for further performance gains, it suffices to use a simple strategy of only including the *linking Constraints* if they appear to be relevant (that is, the Constraint is only active if its output symbols are referenced by logical terms in the tableau). Such performance optimizations are valid because a simulation can be restarted and executed multiple times to generate necessary values for the tableau if a missing Constraint is later deemed to be relevant.

Finally, tableaux methods are not only applicable to true/false theorem proving—they may be used for query answering. By allowing unification during rule application and unground Prolog-style variables, a tableau can generate output in the form of variable bindings. Syntactically, this is achieved by including a query term in the right-hand side of an implication. For example, if we have a wff, $F(x)$ that constrains the value of x, we can discover the legal values of x by posing a query $F(x) \Rightarrow x = \mathbf{X}$, where $\mathbf{X}$ is an unground

Prolog-style variable. When this formula is negated and converted into negation normal form, the tableau will contain query term $x \neq \mathbf{X}$ that will allow the branch to be closed by unification of **`X`** in the contradiction rule with the true assignment for x.

3.2. Data-Structure Regularity

Combining the two reasoning strategies in a flexible hybrid system requires a large range of data structures and therefore introduces the data management challenge of maintaining internal and mutual consistency. A selection of some of these data structures follow:

Deductive Data Structures. Logical terms, tableaux, Slick molecular representations, database/external-procedure backed predicates.

Strategic Data Structures. Heuristic information and scores, search queues, work queues.

Supportive Data Structures. Cached data, indexes.

Configuration Data. Currently active heuristics, tableau rules, simulation 'laws', meta-strategy.

A systematic approach is required, and this can be found by applying object oriented design principles and through the unification of all reasoner state into a single repository.

We generalize the unit of integration—a space of worlds—into a collection object. This collection, known as a **`WorldSpace`**, corresponds to both a branch of a tableau and an instance of a simulation. A **`WorldSpace`** has three major operations—division (forking), expansion (addition) and closure—corresponding to the tableau operations for disjunction, conjunction and contradiction. Aside from meta-strategy, *all* state and configuration is stored within a **`WorldSpace`**.

A set of **`WorldSpace`**s are grouped into a collection known as a **`Problem`**. Each query is initially represented by a single **`Problem`** with a single **`WorldSpace`** containing the query formula and configuration. The system then uses tableaux rules and simulation to reason about the **`WorldSpace`**; dividing, expanding and closing until all **`WorldSpace`**s are closed or no further progress can be made. Aside from a list of current **`WorldSpace`**s, the only system state stored in a **`Problem`** relates to meta-strategy: the processes of selecting a focus **`WorldSpace`** and interpreting the final state of the **`Problem`**. While focus selection has little impact on the theoretical capabilities of the hybrid system, it has great computational significance. Focus selection drives the high-level search across **`WorldSpace`**s. For example, a stack based focus selection strategy results in depth first search, a queue based focus selection results in breadth first search, and a priority queue may be used for heuristic, best-first or A*-style searching.

The key to minimizing the complexity of the system, and yet allowing significant modularity and extensibility lies in the regularization of all state and data. Rather than storing tableau rules, heuristic scores, simulation configuration and simulation state in separate linked data structures, they are all stored as terms in the tableau. That is, a tableau branch (a **`WorldSpace`**) can contain not only logical terms, but a wide range of objects all implemented as subclasses of the class **`Node`**. Such objects include: simulation objects such as **`Entity`**, **`Join`** and **`Constraint`**; logical objects such as **`Term`** and **`Binding`**, dynamic objects such as **`Function`** and **`Task`**; and book-keeping objects such as **`Note`**. In this setting, the tableaux algorithm is generalized so that in each step a branch is searched for not just matching logical terms, but also the appropriate rules to apply. In fact, the tableaux algorithm is itself stored within a **`WorldSpace`** as a dynamic object—as a **`Function`**.

Reasoning within a given **`WorldSpace`** follows a simple interpretation process. When a **`Node`** is added to a **`WorldSpace`**, the **`WorldSpace`** is searched for **`Functions`** with matching parameter lists. These **`Functions`** are then invoked to either perform trivial state updates or to generate **`Task`** objects that are in turn added to the **`WorldSpace`**. Because **`Task`** objects also extend **`Node`**, the creation of a **`Task`** can result in a cascade of **`Function`** calls to perform other minor state updates (such as setting priority) or create new **`Task`**s. When focus is eventually given to a **`WorldSpace`**, a **`Task`** (or set of **`Task`**s) is chosen from the **`WorldSpace`** based on the priority of the **`Task`**, and subsequently executed. The typical execution process is illustrated in Figure 6.

A **`WorldSpace`** begins as an empty collection. It is initialized by adding (one-by-one) an appropriate set of **`Function`**s for reasoning. The query term is then added to the collection as a **`Term`** (extending **`Node`**), resulting in a cascade of new **`Task`**s, that drive the division, expansion and closure of the **`WorldSpace`**, the creation of new **`Node`**s and in turn the ongoing creation of new **`Task`**s. When all **`Task`**s have been exhausted, the system attempts to execute any remaining 'default' **`Function`**s (*i.e.*, parameterless functions), and if they in turn are unable to make progress, the branch is deemed to have stalled (*i.e.*, no further progress can be made).

Within this architecture, tableau based reasoning is implemented by a **`TableauxReasonerFunction`** that responds to logical **`Term`**s. When a **`Term`** is added, it is deconstructed and an appropriate division/extension action is chosen and created as a new **`Task`**.

Simulation-based reasoning is made possible by a set of **`Function`**s that recognize **`Term`**s that constrain the state of a simulation. If, for example, a **`Term`** stating that *x isa Coffee-Mug* is inserted into a **`WorldSpace`**, then a **`SimulationConstructorFunction`** will expand the **`WorldSpace`** with new **`Entity`**s and **`Join`**s appropriately describing the shape of a *Coffee-Mug*. When no further progress can be made using other reasoning methods, a **`SimulatorFunction`**, implemented as a 'default' **`Function`** is executed, creating a new **`Task`** that will apply every **`Constraint`** object (*i.e.*, every physical law) in the **`WorldSpace`** to every **`Entity`** and **`Join`** in the **`WorldSpace`**. While we have not done so, other forms of simulation can be integrated using an identical process.

A range of other reasoning modes are also possible. Some of the modes that we are currently using are described below:

- Heuristics and prioritization are implemented by **`Function`**s that compute a cost or expected benefit and update the priority of **`Task`**s when either the **`Task`** is added (*i.e.*, before they are executed) or in response to the addition of other kinds of objects to the **`WorldSpace`**.
- Methods of informal default reasoning are implemented in two ways: as **`Function`**s that generate plausible extensions in response to the addition of logical terms; and 'default' (parameter-less) **`Function`**s that generate assumptions

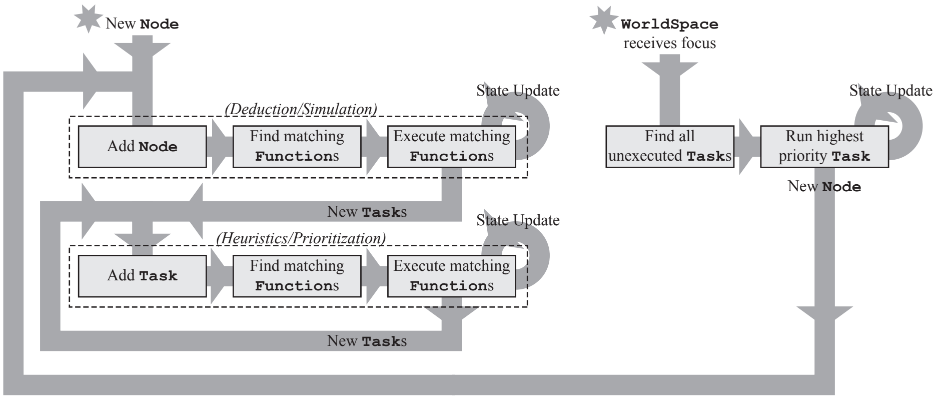

New Node
(Deduction/Simulation)
Add Node
Find matching Functions
Execute matching Functions
State Update
New Tasks
(Heuristics/Prioritization)
Add Task
Find matching Functions
Execute matching Functions
State Update
New Tasks
WorldSpace receives focus
Find all unexecuted Tasks
Run highest priority Task
State Update
New Node

Figure 6. Typical execution/interpretation process for a **`WorldSpace`**.

when ordinary logical deduction has stalled.

- Search *within* a problem space is implemented by 'default' **`Function`**s that generate **`Task`**s to divide a **`WorldSpace`** into a covering set of mutually exclusive conditions.
- Database-driven predicates and 'abbreviated' expressions are implemented as Functions that respond to the addition of a **`Term`**, invoking external routines that test the new **`Term`** or generate expanded forms for addition into the **`WorldSpace`**.

4. Results and Discussion

This architecture is the culmination of our experiences in integrating simulation and logical reasoning in novel combinations; and the only architecture in our experience that so elegantly achieves cohesion, simplicity and open-endedness. We are currently exploring and implementing different logics, simulation laws and heuristics to maximize the usefulness of the system.

The best way to illustrate the full power of this approach is through example. Consider a simple benchmark-style challenge of a reasoning whether it is better to use a slow action or fast action to move a cup of coffee. We can pose the problem as a logical formula (using $\Box$ as the modal 'always' operator):

$$\textit{problem} \equiv \left(\begin{array}{c} \big(\textit{action} = \textit{slow-move}(x) \vee \textit{action} = \textit{fast-move}(\mathrm{x})\big) \\ \wedge\, x \textit{ isa Mug} \wedge x \textit{ contains Coffee} \wedge y \textit{ isa Table} \wedge x \textit{ on } y \\ \wedge\, \textit{standard-physics} \wedge \textit{can-assume-defaults} \end{array} \right) \Rightarrow \Box\big(\neg \textit{mess} \wedge \neg \textit{damage}\big)$$

Alternately, we might pose the problem as a query to discover an appropriate action: (*i.e.*, $\textit{problem} \Rightarrow \textit{action} = \mathbf{A}$).

Given this query, the system builds a tableau with two branches corresponding to the two possible actions. In both branches, the *x isa Mug* and *x contains Coffee* are expanded into a set of **`Entity`**s and **`Join`**s; the *standard-physics* term is expanded into a set of **`Constraint`**s corresponding to the laws of physics; and the *can-assume-defaults* is expanded into **`Function`**s that generate feasible placements, in 3D space, for objects such as the *Table* and the *Mug*. A 3D rendering of the mid-action state of the two branches can be seen in Figure 7.

In the right-hand branch we have a contradiction caused by the *mess* generated in the simulation and the ¬*mess* constraint of our original query. The conclusion that our hybrid reasoner draws is that the preferred action is to use the *slow-move* action on the cup of coffee.

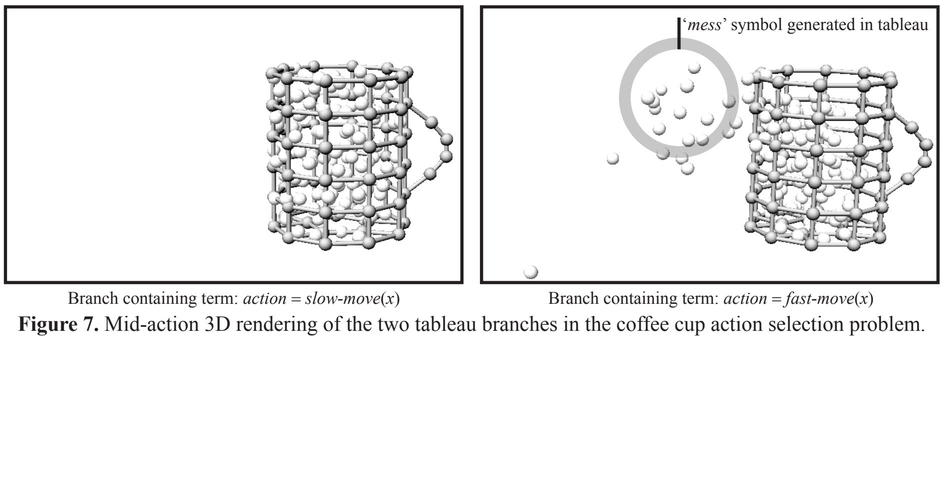

'*mess*' symbol generated in tableau
Branch containing term: *action* = *slow-move*(*x*)
Branch containing term: *action* = *fast-move*(*x*)

Figure 7. Mid-action 3D rendering of the two tableau branches in the coffee cup action selection problem.

While this example is somewhat contrived, a great deal of generality is evident. Little modification is required to query the feasibility of different kinds of actions, different materials or liquids or even the behaviors of entirely novel objects. The advantage of simulation is that it is not necessary to perform comprehensive knowledge elicitation to describe the behavior of every object in every situation; rather, simulation can automatically compute real-world consequences simply from the physical parameters of objects. Combined with the flexibility of logical deduction, this makes for a powerful commonsense reasoning system.

5. Conclusion and Future Work

Though this architecture represents a first step for the *Comirit* project, and remains a long way from true artificial general intelligence, our early results have demonstrated both significant promise as a tool for commonsense reasoning and an enormous capability for extension and modularity. Reasoning in this hybrid architecture combines both the flexibility and power of formal logics with the ease of specification and the implicit commonsense 'know-how' of simulations: even in our simple examples, the architecture elegantly and efficiently solves problems that are difficult to express using other methods.

There is ample scope for future development. In the short term, we intend to explore the potential for including new tableau rules for more efficient and different forms of logical deduction, for including entirely new reasoning mechanisms (integrating along the 'possible worlds') and for developing new heuristics. Our longer term vision includes the integration of the reasoning and simulation mechanisms with machine learning and robot vision systems, and subsequently deploying the technology in real-world robot systems. Given the modularity of the architecture and the generality of integration via 'possible worlds', we believe that the potential is enormous.

References

[1] Morgenstern, L. & Miller, R. 2006, *The Commonsense Problem Page*, viewed 10 May 2006 <http://www-formal.stanford.edu/leora/commonsense/>.

[2] Barker, K., Chaudhri, V., Chaw, S., Clark, P., Fan, J., Israel, D., Mishra, S., Porter, B., Romero, P., Tecuci, D. & Yeh, P., 2004, 'A question-answering system for AP chemistry: assessing KR&R technologies', KR2004.

[3] Lenat, D.B., Guha, R.V., Pittman, K., Pratt, D. & Shepherd, M. 1990, 'Cyc - toward Programs with Common-Sense', *Communications of the ACM*, vol. 33, no. 8, pp. 30-49.

[4] Bryson, J. 2003, 'The behaviour-oriented design of modular agent intelligence' in *Agent Technologies, Infrastructures, Tools and Applications for e-Services*, LNCS 2592/2003.

[5] Goertzel, B. Heljakka, A., Bugaj, S. & Pennachin, M. 2006, 'Exploring android developmental psychology in a simulation world', Proc. of ICCCS-2006.

[6] Johnston, B. & Williams, M-A. 2007, 'A generic framework for approximate simulation in commonsense reasoning systems', 8th Int. Sym. on Logical Formalizations of Commonsense Reasoning.

[7] Horrocks, I. & Sattler, U. 2001, 'Ontology reasoning in the SHOQ(D) description logic', IJCAI 2001.

[8] Gardin, F. & Meltzer, B. 1989, 'Analogical representations of naïve physics', *Artificial Intelligence*, vol. 38, no. 2, pp. 139-159.

[9] Hoyes, K. 2007, '3D simulation: the key to AI' in Goertzel, B. & Pennachin, C. (eds) *Artificial General Intelligence*, Springer, Berlin.

[10] Morgenstern, L. 2001, 'Mid-sized axiomatizations of common-sense problems: a case study in egg-cracking', *Studia Logica*, vol. 67, no. 3, pp. 333-384.

[11] D'Agostino, M., Gabbay, D., Hähnle, R. & Posegga J. (eds) 1999, *Handbook of Tableau Methods*, Kluwer, Netherlands.