NISTIR 7099

Automated composition of conversion software

David Flater

National Institute of Standards and Technology
Technology Administration, U.S. Department of Commerce

NISTIR 7099

Automated composition of conversion software

David Flater
Manufacturing Systems Integration Division
Manufacturing Engineering Laboratory

March 2004

U.S. DEPARTMENT OF COMMERCE
Donald L. Evans, Secretary
TECHNOLOGY ADMINISTRATION
Phillip J. Bond, Under Secretary of Commerce for Technology
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
Arden L. Bement, Jr., Director

Automated composition of conversion software

David Flater

NISTIR 7099
2004-03-04

Abstract

Given a library of converters with known behaviors, an integration engine must determine which are useful in a given scenario and produce a plan for how they can be used to achieve specified goals. This report documents the algorithm implemented by the AMIS3 integration engine. First, the problem is formalized and shown to be NP-hard. Then, the algorithm is explained and shown to be correct for all deterministic scenarios. Next, the behavior for nondeterministic scenarios is discussed. Finally, an alternate approach is outlined.

Keywords: algorithm, integration, search, software

1 Introduction

This report documents the algorithm implemented by the "AMIS3" integration engine that was designed for use in the Automated Methods of Integrating Systems (AMIS) project [1]. The objective of the AMIS project is to reduce the cost and time for software integration by devising methods, algorithms, and tools by which activities of the human systems engineer can be automated [2].

AMIS3 is the third stage in the AMIS architecture. The input to AMIS3 is an unambiguous, machine-readable specification of an integration scenario in a known context. The other two stages of AMIS are responsible for the information extraction and semantic analysis that are required to create that specification. AMIS3's responsibility is to utilize available libraries and knowledge bases to generate an executable plan for the specified scenario.

A significant subproblem is to generate a converter to convert one thing, "what you have," to another thing, "what you want," on the assumption that "what you have" and "what you want" are known, correct, and unambiguous. Given a library of converters with known behaviors, AMIS3 must determine which are useful in a given scenario and produce a plan for how they can be used to transform "what you have" into "what you want."

Seldom does a converter implement a one-to-one mapping. Most conversions that could be useful in an integration scenario are destructive in some way. A poorly chosen sequence of conversions can easily destroy the information content that the integrator wanted to communicate from one system to another. Therefore, the plan that AMIS3 generates must be "reasonably good" in the sense of avoiding unnecessary destructive conversions and showing preference for less destructive alternatives.

2 Formalization of the problem

Although they are intuitively attractive for both uses, the terms "problem" and "solution" could refer ambiguously either to the general case (the algorithm is a "solution" to the general AMIS3 "problem") or to the specific case (the algorithm finds a "solution" to the specific "problem" given as input). To disambiguate, *problem* and *solution* are used only in the general case. For the specific case, *scenario* and *plan* are used.

A scenario consists of a set of *processes* representing the library of converters available to AMIS3, a *starting set* of *tokens*, H, representing "what you have," and a *goal set* of *tokens*, W, representing "what you want." These sets are assumed to be finite and immutable for a given scenario.

A *token* is an identifier for a *datum*. A *datum* (plural, *data*) is a specific machine-readable artifact whose role in the scenario and appropriateness to serve as input to available *processes* are known without ambiguity. For example, a file whose role in the scenario and appropriateness to serve as input to available processes is known without ambiguity would be a datum; a file name that unambiguously identifies that file would suffice as a token.

Data can be created by processes, but they cannot be modified or destroyed. If an output of a process is in any way distinguishable from all of its inputs, then it is a different datum. Thus, to continue the previous example, no process may overwrite or delete the file, but any process may create a new file, identified by a different token, that is understood to be a newer revision. This restriction does not mean there can be no loss of information in a conversion: the relationship between the input data and the output data is unspecified.

For a process p, the *input set* $I(p)$ is a finite set of tokens representing the prerequisites for executing the process, the *outcome set* $M(p)$ is the finite set of *outcomes* for the process, and the *cost*, $C(p) > 0$, is a number representing the amount of damage caused by an execution of the process.

An outcome o consists of a finite *output set* of tokens, $T(o)$, and a *probability*, $0 < P(o) \leq 1$, such that

$$\sum_{o \in M(p)} P(o) = 1 \tag{1}$$

A process is called *deterministic* if every execution of that process in a given scenario will produce the same outcome. If it is possible to obtain different outcomes by repeating the execution of a process, the process is called *nondeterministic*. For example, consider a document conversion process that has a "success" outcome and a "failure" outcome. If the process always succeeds on some documents and always fails on others, it is deterministic. But if it sometimes succeeds and sometimes "randomly" fails on a given document, such that one might recover from a failure by retrying the process with the same input, it is nondeterministic.

The outcome of a process cannot be affected by any data except those identified in the input set of the process. Nondeterminism is accepted at face value; any hypothesized influence that other data may have on the outcome of the process is out of scope.

All of the above properties are immutable for a given process.

A *plan* is a decision tree of process executions. In the trivial case in which none of the processes in a plan has more than one outcome, the plan is simply a sequence of process

executions. When a process in the plan has more than one outcome, separate *branches* give the continuation for each of its outcomes.

A *branch* of a plan is a sequence of process executions *with identified outcomes* leading from the root of the plan to a leaf. The different outcomes of a leaf process execution identify separate branches even though the continuation is null.

Let $B(l)$ represent the set of branches of a plan l.

$$|B(l)| \geq 1$$

Let $|b|$ represent the number of processes executed along a branch b (i.e., the *length* of the branch). Let $M_i(b)$ represent the i^{th} outcome along a branch b. Let $P(b)$ represent the probability of a branch.

$$P(b) = \prod_{i=1}^{|b|} P(M_i(b)) \tag{2}$$

$$\sum_{b \in B(l)} P(b) = 1 \tag{3}$$

Let $S_i(b)$ represent the *state* resulting from the i^{th} outcome along a branch b.

$$S_i(b) = \begin{cases} H & \text{for } i = 0 \\ S_{i-1}(b) \cup T(M_i(b)) & \text{for } 0 < i \leq |b| \end{cases}$$

(H is the *starting set* that was defined at the top of this section.)

Define $S(b) = S_{|b|}(b)$.

Let $R(b)$ represent the set of processes executed along a branch b, and let $R_i(b)$ represent the i^{th} process executed along a branch. A process can only be executed if its input set is a subset of the current state.

$$I(R_i(b)) \subseteq S_{i-1}(b)$$

Let $C(b)$ represent the cost of a branch.

$$C(b) = \begin{cases} 0 & \text{for } |b| = 0 \\ \sum_{i=1}^{|b|} C(R_i(b)) & \text{for } |b| > 0 \end{cases} \tag{4}$$

The unique *null plan* has exactly one branch, whose length is 0, whose cost is 0, and whose probability is 1. No other plan is permitted to contain any branches of length 0.

The cost of a plan l, $C(l)$, is defined as the expected cost,

$$C(l) = \sum_{b \in B(l)} P(b)C(b) \tag{5}$$

The *completeness* of a branch b, $D(b)$, is

$$D(b) = \begin{cases} 1 & \text{if } |W| = 0 \\ \frac{|S(b) \cap W|}{|W|} & \text{otherwise} \end{cases} \tag{6}$$

(W is the *goal set* that was defined at the top of this section.)

A branch b is called *complete* if and only if $D(b) = 1$.

The completeness of a plan l, $D(l)$, is

$$D(l) = \sum_{b \in B(l)} P(b)D(b) \tag{7}$$

A plan l is called complete if and only if $D(l) = 1$.

If there exists at least one complete plan for a given scenario, let Z represent the minimum cost of a complete plan. An *optimal plan* is a complete plan whose cost is Z. A *reasonably good plan* is a complete plan whose cost is $\leq rZ$ for some constant value $r \geq 1$ chosen by the intended user. All other complete plans are *bad plans.*

A simple example to illustrate the use of this formalism can be found in the Appendix.

3 Hardness of the problem

Theorem 1 *Finding a reasonably good plan for automated composition of conversion software is NP-hard.*

Proof: Without loss of generality, assume that $C(p) = 1$ and $|M(p)| = 1$ for every process p, and assume that $|T(o)| = 1$ for every outcome o. The general problem cannot be less hard than this special case.

For each token contained in the starting set, the goal set, the input set of any process, or the output set of the outcome of any process, assign a logical literal x_n. The starting set can then be represented as a set of facts, and each process can be represented as a definite Horn clause,

$$\neg x_j \vee \neg x_k \vee \cdots \vee x_l$$

The problem then is to find a proof of the goal set whose length, measured by number of steps in the proof, is within the constant factor r of the length of a shortest proof (i.e., a *reasonably good* proof).

The problem of finding the minimum proof length, measured by number of steps in the proof, in a Horn clause resolution system is known to be NP-hard to approximate within any constant factor [3, 4]. Given a reasonably good proof, the time required to approximate the minimum proof length within the constant factor r would be linear; it follows that finding a reasonably good proof or a reasonably good plan is NP-hard. $\triangle$

4 Search algorithm

It often happens that the problem of interest in practice is a less hard special case of an NP-hard general problem. Unfortunately, AMIS3 was specifically intended to address the general problem. The requirements may be relaxed in future iterations of the project, but for now, an algorithm is given to solve the general problem.

Since finding a reasonably good plan did not appear to be significantly less hard than finding an optimal plan, AMIS3's search was configured to find an optimal plan. As Section 5 will show, the algorithm is equivalent to a specialization of A* [5, 6, 7, 8, 9, 10, 11, 12, 13].

The AMIS3 algorithm is shown in Figure 1. Several new definitions are used.

Two plans are *equivalent* if their formalizations according to Section 2 are identical. They must have identical branches, with the same processes executed in the same order.

For a plan l,

$$\max(l) = \bigcup_{b \in B(l)} S(b)$$

$$\min(l) = \bigcap_{b \in B(l)} S(b)$$

The *footprint* of a process p is

$$F(p) = \bigcup_{o \in M(p)} T(o)$$

MaxCost is a user-defined limit on the maximum cost of any branch in an acceptable plan. It is used to force termination in scenarios where every optimal plan contains a branch of infinite length (see Section 6). If not required it should be set to infinity.

"Note:" indicates no operation, but provides information for logging and accounting purposes. "Exception:" indicates an abnormal termination of the algorithm. "[Continue]" is the target for an early exit from nested blocks.

5 Correctness for deterministic scenarios

Assumption 1 *Assume that MaxCost is set to infinity.*

With MaxCost set to infinity, the AMIS3 search algorithm is equivalent to a specialization of the well-known graph search algorithm A* [5, 6, 7, 8, 9, 10, 11, 12, 13].

Apart from pruning, which is discussed below, the only nontrivial deviation from the canonical form occurs in the handling of nodes that are generated more than once. When a generated successor n' is already on Open or Closed, A* specifies that the "pointers" of n' should be directed along whichever path was cheapest, and if n' required adjustment and was found on Closed, it should be reopened. As will be shown below, all paths to a given node in this problem have the same cost; therefore no "pointers" ever need to be adjusted and no nodes on Closed ever need to be reopened. Consequently, no action should be taken with respect to n'.

Instead of taking no action, the AMIS3 algorithm adds a duplicate representation of n' to Open. Each of these representations may be visited in turn. If the first visited representation of n' falls victim to Prune-I, all others do as well. Otherwise, the first visitation results in n' being added to Closed, and all duplicate representations are discarded via Prune-III when they are visited. Thus, it is not possible for n' to be expanded more than once. A* makes no reference to nodes on Open except to remove them for expansion or to search for duplicates (generated successors already on Open), so the presence of duplicate nodes on Open has no impact, and the properties of A* are preserved.

Let l = the null plan
Let Open = empty list
Let Closed = empty list
While $D(l) < 1$

    If l is equivalent to any plan in Closed, Note: Prune-III
    Else if $\max(l) \subseteq \min$(any plan in Closed), Note: Prune-I
    Else

        Add l to Closed
        Let Expansions = empty list
        Let b iterate through incomplete branches of l

            Let DeadEndFlag = True
            Let p iterate through all processes such that ($p \notin R(b) \vee p$ is nondeterministic) $\wedge\ I(p) \subseteq S(b) \wedge F(p) \not\subseteq S(b)$

                If $C(b) + C(p) >$ MaxCost, Note: MaxCost exceeded
                Else

                    Set DeadEndFlag = False
                    Add to Expansions an expansion of l in which b is extended by p, yielding one branch for each outcome of p

            If DeadEndFlag = True

                Note: Prune-II
                Go to Continue

        Add Expansions to Open

    [Continue] If Open is empty, Exception: goal not feasible
    Set l = any lowest cost plan in Open
    Remove l from Open

Return l

Figure 1: AMIS3 search algorithm

In all other respects, the algorithm is clearly equivalent to the specialization of A* induced by the following bindings for variables and terminology in the definition that appears in [5, Section 2.4.4].

A node n represents a plan, $L(n)$. The start node represents the null plan. A node is a goal node if it represents a complete plan.

An arc a from node n to a successor n' represents the expansion of some branch $b \in B(L(n))$ by the execution of a process, $R(a)$, where $I(R(a)) \subseteq S(b)$. The cost function $c(n, n')$ that gives the cost of the arc is

$$c(n, n') = P(b)C(R(a)) \tag{8}$$

All process costs and all outcome probabilities are greater than 0, so $c(n, n') > 0$.

It is possible to get more than one path leading to a given node by expanding branches in different orders, ultimately producing an identical plan.

Theorem 2 *In a graph where nodes represent plans and the cost of an arc is given by Eq. (8), the cost of any path from the start node to a node n is $C(L(n))$.*

Proof: (By induction) Base case: the cost of a null plan is 0; no arcs are traversed to reach the start node, so the sum of arc costs is also 0. Inductive hypothesis: hypothesize a node n such that the cost of any path from the start node to n is $C(L(n))$. Inductive step: consider a successor of n, n', in which branch b of $L(n)$ is expanded by an execution of process p, yielding branches $b_1 \ldots b_{|M(p)|}$. The cost of $L(n')$ relative to $L(n)$ using Eq. (5) is found by subtracting the cost of the branch that was expanded and adding the costs of the new branches,

$$C(L(n')) = C(L(n)) - P(b)C(b) + \sum_{i=1}^{|M(p)|} P(b_i)C(b_i)$$

By Eqs. (2) and (4),

$$= C(L(n)) - P(b)C(b) + \sum_{o \in M(p)} P(b)P(o)\left(C(b) + C(p)\right)$$

By Eq. (1),

$$\begin{aligned} &= C(L(n)) - P(b)C(b) + P(b)\left(C(b) + C(p)\right) \\ &= C(L(n)) + P(b)C(p) \\ &= C(L(n)) + c(n, n') \end{aligned}$$

which is the cost of the path to n' using Eq. (8). $\triangle$

Corollary 1 *All paths from the start node to a given node have the same cost.*

The heuristic function $h(n)$ that gives an estimate of the cost of a cheapest path from node n to any goal node is fixed at 0 (the uniform-cost strategy). $h(n) = 0$ satisfies the definition of an *admissible* heuristic function,

$$h(n) \leq h^*(n)$$

where $h^*(n)$ is the actual cost of a cheapest path from n to any goal node. It follows that *if* A* terminates, it will either return an optimal plan or correctly exit with failure because no complete plan exists.

A* always terminates on finite graphs, but the graph as specified above is not finite. Ensuring termination requires two additional assumptions.

Assumption 2 *Assume that all processes are deterministic.*

Recall from Section 2 that a process is called *deterministic* if every execution of that process in a given scenario will produce the same outcome. If all processes are deterministic, then no branch of any optimal plan can have length greater than the total number of processes available, so a finite graph will suffice to represent all optimal plans.

Assumption 3 *Assume that only process executions where $p \notin R(b)$ are represented with arcs in the graph.*

With the above assumptions, the graph is finite and termination is ensured.

It remains to be shown that the pruning that is done by the AMIS3 algorithm preserves the correctness of A*.

Theorem 3 *Nodes removed by Prune-I are always redundant.*

Proof: Prune-I prevents the expansion of a plan l for which $\max(l) \subseteq \min(l')$ for some $l' \in$ Closed. Since $h(n)$ is *monotone* (satisfying $h(n) \leq c(n, n') + h(n')$), the costs of the plans visited by A* are non-decreasing. Therefore $C(l') \leq C(l)$. The pruning condition dictates that l' must produce with a probability of 1 every datum that l produces with any nonzero probability. If any expansion of l leads to a complete plan, the lowest cost sequence of process executions among those used to complete its branches could be applied to *every* branch of l' to produce a complete plan of lesser or equal cost. If any expansion of l leads to an optimal plan, this substitution yields an optimal plan derived from l'. Therefore l is always redundant. $\triangle$

Theorem 4 *Nodes removed by Prune-II are always redundant.*

Proof: Prune-II prevents the expansion of a plan l for which some branch b is incomplete but not expandable. From Eqs. (3), (6) and (7), no plan containing an incomplete branch can itself be complete. Since b is not expandable, b will remain incomplete in any possible descendant of l. Therefore expansion of l cannot lead to any complete plan, and l is always redundant. $\triangle$

As discussed above, Prune-III is actually part of the behavior of the canonical A* algorithm.

The long condition on branch expansion implements a combination of things, including more pruning.

- ($p \notin R(b) \vee p$ is nondeterministic) prunes arcs representing repeat executions of deterministic processes, incidentally validating Assumption 3 whenever Assumption 2 holds. By definition, repeated execution of a deterministic process cannot yield a different outcome; correctness is trivial.
- $I(p) \subseteq S(b)$ implements the precondition on execution of a process.
- $F(p) \not\subseteq S(b)$ prunes arcs representing process executions that could not possibly yield new data. Correctness is trivial.

6 Behavior for nondeterministic scenarios

If one nondeterministic process is admitted, the search is no longer guaranteed to terminate. By inspection, one can verify that it will not terminate if every optimal plan includes a branch of infinite length. The following example shows that this is possible.

Imagine a strange dice game in which the goal is to escape with the smallest loss. At any time, the player can just forfeit \$2 and walk away. However, the player is also entitled to leave if he or she forfeits \$1 and rolls any number except 5 on a 6-sided die. If the player rolls a 5, he or she remains in the game, faced with the same choice again.

Let p_1 be a nondeterministic process for paying \$1 and rolling the die; let p_2 be a deterministic process for paying \$2 and walking away.

$$H = \{\}$$
$$W = \{X\}$$
$$I(p_1) = \{\},\ C(p_1) = 1,\ M(p_1) = \{o_1, o_2\}$$
$$T(o_1) = \{X\},\ P(o_1) = 5/6$$
$$T(o_2) = \{\},\ P(o_2) = 1/6$$
$$I(p_2) = \{\},\ C(p_2) = 2,\ M(p_2) = \{o_3\}$$
$$T(o_3) = \{X\},\ P(o_3) = 1$$

Let l be the plan with infinite retries of p_1. The cost of l is \$1.20.

$$C(l) = \frac{5}{6} + \frac{1}{6}\left(2 \times \frac{5}{6} + \frac{1}{6}\left(3 \times \frac{5}{6} + \frac{1}{6}\left(4 \times \frac{5}{6} + \cdots\right)\right)\right)$$
$$C(l) = \frac{5}{6} + \frac{2 \times 5}{6^2} + \frac{3 \times 5}{6^3} + \frac{4 \times 5}{6^4} + \cdots$$
$$\frac{1}{6}C(l) = \frac{5}{6^2} + \frac{2 \times 5}{6^3} + \frac{3 \times 5}{6^4} + \frac{4 \times 5}{6^5} + \cdots$$
$$\frac{5}{6}C(l) = C(l) - \frac{1}{6}C(l) = \frac{5}{6} + \frac{5}{6^2} + \frac{5}{6^3} + \frac{5}{6^4} + \cdots$$
$$\frac{5}{36}C(l) = \frac{1}{6} \times \frac{5}{6}C(l) = \frac{5}{6^2} + \frac{5}{6^3} + \frac{5}{6^4} + \cdots$$
$$\frac{25}{36}C(l) = \frac{5}{6}C(l) - \frac{5}{36}C(l) = \frac{5}{6}$$
$$C(l) = \frac{6}{5}$$

Plan l is clearly optimal. At every turn, regardless of how much money has been spent already on failed attempts, one is faced with the same decision between a certain loss of \$2 or an expected loss of \$1.20. In theory, the best choice is always to roll again.

In practice, money is finite, and one can only try so many times before it becomes necessary to limit further losses. Moreover, as the observed behaviors become increasingly unlikely in theory, one must also consider the relative likelihood that the theory is not a correct model of reality (i.e., that the die is loaded or has a 5 on more than one side). Either way, at some point the theoretically optimal actions become unacceptable.

By pruning partial plans in which the cost of any branch exceeds a threshold (MaxCost), it is possible to find reasonably good plans for scenarios such as the above with a finite amount of searching. However, if the threshold is set too low, the search will terminate prematurely, returning either no plan or a bad plan. The work involved in analytically determining the minimum acceptable threshold generally renders the search redundant, so proof of correctness for tractable nondeterministic scenarios would be uninteresting. In practice, different thresholds are tried until one is satisfied with the results.

7 Alternate approach

An alternate approach is to formulate the search space as an AND/OR graph and apply the more complicated AO* algorithm [5, 8, 14, 15, 16, 17, 18]. Variables and terminology in the following description again follow [5, Section 2.4.4].

The start node is an OR node. The start node has one successor for each process p such that $I(p) \subseteq H \wedge F(p) \not\subseteq H$. These successors are themselves AND nodes that have one successor for each outcome in $M(p)$. The successors of these AND nodes are OR nodes. The expansion continues similarly, alternating OR nodes for the selection among applicable processes with AND nodes for the resulting outcomes.

Each OR node corresponds to a branch. Every OR node corresponding to a complete branch is a goal node and is assumed to have no successors. Every node except the start node has exactly one predecessor—the graph is a tree.

A *solution base* is a subgraph that contains the start node, every successor of every expanded AND node, exactly one successor of every expanded OR node, and *no* nodes labeled "unsolvable." Every node on the frontier of a solution base is either in Open or a goal node.

A solution base g in which all leaf nodes are OR nodes represents a plan, $L(g)$. For completeness, if a solution base g' is formed by expanding one or more leaf OR nodes in g, define $L(g') = L(g)$.

Process costs are ascribed to the transitions out of AND nodes; transitions out of OR nodes are assigned 0 costs.

Let the successors (if any) of node n be denoted by $n_1 \ldots n_m$. If n is an AND node, let $C(n)$ represent the cost of the process associated with that node and let $P(n_i)$ represent the

probability of the outcome associated with successor n_i. Substituting *merit* as the negation of cost, the *backed-up evaluation function* $e(n)$ is

$$e(n) = \begin{cases} h(n) = 0 & \text{if } n \text{ is in Open or is a goal node} \\ -C(n) + \sum_{i=1}^{m} P(n_i)e(n_i) & \text{if } n \text{ is a closed AND node} \\ \max_{i=1}^{m} e(n_i) & \text{if } n \text{ is a closed OR node} \end{cases} \quad (9)$$

The assignment $h(n) = 0$ for nodes in Open is an "optimistic" heuristic function to once again ensure that only optimal plans will be found.

Theorem 5 *In an AND/OR graph where a solution base represents a plan and the merit estimate of a node is given by Eq. (9), the merit estimate of the starting node of any solution base g is $-C(L(g))$.*

Proof: (By induction) Base case: the cost of a null plan is 0; the start node s is either in Open or a goal node, so $e(s) = 0$. Inductive hypothesis: hypothesize a solution base g with start node s such that $e(s) = -C(L(g))$. Inductive step: let g' be a solution base with start node s' that was formed by expanding a nongoal node n on the frontier of g.

There is exactly one path from the start node to n, which represents a branch, b. The arcs leading out of AND nodes along that path indicate process outcomes $M_1(b) \ldots M_{|b|}(b)$.

If n is an AND node, the 0 value assigned to n is replaced by $-C(n) + \sum_{i=1}^{m} P(n_i)e(n_i)$. All of the just-generated successors n_i are either in Open or goal nodes, so $\sum_{i=1}^{m} P(n_i)e(n_i) = 0$ and the value assigned is simply $-C(n)$. Propagating this change back up the path, the impact on the merit of the start node is

$$e(s') = -C(L(g)) - P(b)C(n)$$

This can be shown to equal $-C(L(g'))$ by a derivation mirroring the one in Theorem 2.

If n is an OR node, the 0 value assigned to n is replaced by the merit of its successor. Since the just-generated successor is either in Open or a goal node, the value assigned is still 0, hence $e(s') = e(s) = -C(L(g))$. By definition, $L(g') = L(g)$, so $e(s') = -C(L(g'))$. $\triangle$

8 Conclusion

This report documented the algorithm implemented by the AMIS3 integration engine. First, the problem was formalized and shown to be NP-hard. Then, the algorithm was explained and shown to be correct for all deterministic scenarios. Next, the behavior for nondeterministic scenarios was discussed. Finally, an alternate approach was outlined.

With one subproblem of general integration effectively handled, future work on AMIS3 will focus on other subproblems, such as generating adapters for interactive protocols that have conflicting choreographies for equivalent transactions, or on special cases of the general problem that allow for more efficient solutions.

An implementation in Common LISP is available upon request.

Acknowledgments

The author thanks Paul Black and Al Jones, whose reviews helped to improve this report.

References

[1] Edward J. Barkmeyer, Allison Barnard Feeney, Peter Denno, David W. Flater, Donald E. Libes, Michelle Potts Steves, and Evan K. Wallace. Concepts for automating systems integration. NISTIR 6928, National Institute of Standards and Technology, 100 Bureau Drive, Gaithersburg, MD 20899, 2003.

[2] Michelle Potts Steves, Edward J. Barkmeyer, Peter Denno, David Flater, Don Libes, Evan Wallace, and Allison Barnard Feeney. The AMIS approach to systems integration: An overview. NISTIR to appear, National Institute of Standards and Technology, 100 Bureau Drive, Gaithersburg, MD 20899, 2004.

[3] Michael Alekhnovich, Sam Buss, Shlomo Moran, and Toniann Pitassi. Minimum propositional proof length is NP-hard to linearly approximate. *Lecture Notes in Computer Science*, 1450:176–184, 1998.

[4] Michael Alekhnovich, Sam Buss, Shlomo Moran, and Toniann Pitassi. Minimum propositional proof length is NP-hard to linearly approximate. *Journal of Symbolic Logic*, 66:171–191, 2001.

[5] Judea Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* Addison-Wesley, 1984.

[6] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[7] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. Correction to a formed basis for the heuristic determination of minimum cost paths. *SIGART Newsletter*, 37:28–29, 1972.

[8] Nils J. Nilsson. *Problem-Solving Methods in Artificial Intelligence.* McGraw-Hill, New York, 1971.

[9] Donald Michie and R. Ross. Experiments with the adaptive graph traverser. *Machine Intelligence*, 5:301–308, 1970.

[10] James R. Slagle and Philip Bursky. Experiments with a multipurpose theorem-proving heuristic program. *Journal of the ACM*, 15:85–99, 1968.

[11] I. Pohl. First results on the effect of error in heuristic search. *Machine Intelligence*, 5:219–236, 1970.

[12] I. Pohl. Practical and theoretical considerations in heuristic search algorithms. *Machine Intelligence*, 8:55–72, 1977.

[13] Nils J. Nilsson. *Principles of Artificial Intelligence.* Tioga, Palo Alto, California, 1980.

[14] Saul Amarel. An approach to heuristic problem-solving and theorem proving in the propositional calculus. In J. F. Hart and Satoru Takasu, editors, *Systems and Computer Science*. University of Toronto Press, Toronto, 1967.

[15] Nils J. Nilsson. Searching problem-solving and game-playing trees for minimal cost solutions. *Information Processing*, 68:1556–1562, 1969.

[16] Chin-Liang Chang and James R. Slagle. An admissible and optimal algorithm for searching AND/OR graphs. *Artificial Intelligence*, 2(2):117–128, 1971.

[17] Alberto Martelli and Ugo Montanari. Additive AND/OR graphs. In *Proceedings of the Third International Joint Conference on Artificial Intelligence (IJCAI 3)*, pages 1–11, Stanford, 1973.

[18] Alberto Martelli and Ugo Montanari. Optimizing decision trees through heuristically guided search. *Communications of the ACM*, 21(12):1025–1039, 1978.

Appendix: a simple example

Suppose Alice wants to order some product from Bob, but Alice and Bob use incompatible ordering systems. The incompatibility occurs because Alice's system expresses the effective period of the order using from-date and duration while Bob's system expresses it using from-date and to-date. The task for AMIS3 is to convert the order output by Alice's system into a form that is usable by Bob's system.

$H = \{\text{AliceOrder}\}$

$W = \{\text{BobOrder}\}$

$I(\text{Extract-From-AliceOrder}) = \{\text{AliceOrder}\}$
$C(\text{Extract-From-AliceOrder}) = 1$
$M(\text{Extract-From-AliceOrder}) = \{o_1\}$
$T(o_1) = \{\text{OrderItem}, \text{FromDate}, \text{Duration}\}$, $P(o_1) = 1$

$I(\text{Convert-Effective-Period}) = \{\text{FromDate}, \text{Duration}\}$
$C(\text{Convert-Effective-Period}) = 1$
$M(\text{Convert-Effective-Period}) = \{o_2\}$
$T(o_2) = \{\text{ToDate}\}$, $P(o_2) = 1$

$I(\text{Assemble-BobOrder}) = \{\text{OrderItem}, \text{FromDate}, \text{ToDate}\}$
$C(\text{Assemble-BobOrder}) = 1$
$M(\text{Assemble-BobOrder}) = \{o_3\}$
$T(o_3) = \{\text{BobOrder}\}$, $P(o_3) = 1$

The sequence Extract-From-AliceOrder, Convert-Effective-Period, Assemble-BobOrder is a complete plan for this scenario, with cost 3. Let b represent the single branch of that plan.

$S_0(b) = \{\text{AliceOrder}\}$
$S_1(b) = \{\text{AliceOrder}, \text{OrderItem}, \text{FromDate}, \text{Duration}\}$
$S_2(b) = \{\text{AliceOrder}, \text{OrderItem}, \text{FromDate}, \text{Duration}, \text{ToDate}\}$
$S_3(b) = \{\text{AliceOrder}, \text{OrderItem}, \text{FromDate}, \text{Duration}, \text{ToDate}, \text{BobOrder}\}$