AUTOMATIC SCHEDULING OF OUTAGES OF NUCLEAR POWER PLANTS WITH TIME WINDOWS

CALSPAN-UB RESEARCH CENTER

Dr. Carla Gomes

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

19961125 023

Rome Laboratory
Air Force Materiel Command
Rome, New York
This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RL-TR-96-157 has been reviewed and is approved for publication.

APPROVED:  
KAREN M. ALGUIRE  
Project Engineer

FOR THE COMMANDER:  
JOHN A. GRANIERO  
Chief Scientist  
Command, Control & Communications Directorate

If your address has changed or if you wish to be removed from the Rome Laboratory mailing list, or if the addressee is no longer employed by your organization, please notify RL/C3CA, 525 Brooks Road, Romé, NY 13441-4505. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.
This report describes a successful project for transference of advanced AI technology into the domain of planning of outages of nuclear power plants as part of DOD's dual-use program. ROMAN (Rome Lab Outage Manager) is the prototype system that was developed as a result of this project. ROMAN's main innovation compared to the current state-of-the-art of outage management tools is its capability to automatically enforce safety constraints during the planning and scheduling phase. Another innovative aspect of ROMAN is the generation of more robust schedules that are feasible over time windows. In other words, ROMAN generates a family of schedules by assigning time intervals as start times to activities rather than single start times, without affecting the overall duration of the project. ROMAN uses a constraint satisfaction paradigm combining a global search tactic with constraint propagation. The derivation of very specialized representations for the constraints to perform efficient propagation is a key aspect for the generation of very fast schedules - constraints are compiled into the code, which is a novel aspect of our work using an automatic programming system, KIDS.
Contents

1 Introduction ........................................................................................................... 1

2 Related Work ......................................................................................................... 2

3 Planning and Scheduling of Nuclear Power Plant Outages ............................ 3

   3.1 Definition of Problem ...................................................................................... 4

       3.1.1 Activities ............................................................................................... 4

       3.1.2 Plant Configuration and Risk Management ......................................... 4

       3.1.3 Resources ............................................................................................. 5

4 Outage Planning and Scheduling in the Real World ........................................... 6

   4.1 Time Window Assignment .............................................................................. 7

   4.2 Plant Configuration and Risk Management .................................................. 7

   4.3 Resources ........................................................................................................ 8

5 ROMAN - Rome Lab. Outage Manger ................................................................. 8

   5.1 Search and Control Mechanisms .................................................................. 10

       5.1.2 Constraint Propagation ....................................................................... 10

       5.1.3 Interaction Between The Schedule and the State of Plant ................. 15

6 Performance Results ............................................................................................. 16

7 Conclusions and Future Work .............................................................................. 18

8 Acknowledgments ................................................................................................ 18

9 References ............................................................................................................. 19

Appendix A - Derivation of Constraints from the Separation Constraint .......... 21

Appendix B - Derivation of Constraints from the ACP Constraint ..................... 29
1 Introduction

Planning and scheduling tasks are inherently complex. In computational terms, they are intractable, i.e., NP-hard or worse. As a practical consequence, realistic size planning and scheduling problems cannot be solved optimally in a "reasonable" amount of time. Nonetheless, solutions have to be found for real-world problems, and therefore heuristic approaches have to be adopted, ideally with some guarantee on the quality of the solution.

This paper focus on the real-world problem of multiple resource-constrained project management. This problem is very common in manufacturing and it is a generalization of the well-known job-shop scheduling problem (Blazewicz et al 83, Vaessens et al 94). As a particular instance of this problem, we consider the management of outages of nuclear power plants. An outage is a planned shutdown for refueling, repair, and maintenance. It is a rather daunting real-world task that may involve from 10,000 up to 45,000 activities. In the domain of nuclear power plants, risk and safety management are sine qua non conditions and therefore a planning and scheduling system (automatic or manual) has to enforce safety constraints guaranteeing that the state of the plant is safe at any time during an outage. The current automatic technology for outage scheduling used by the utilities does not take into consideration safety requirements — currently, safety and risk management still heavily rely on the experience of the manual schedulers, rather than on automatic procedures. Furthermore, in this domain, the existence of good automatic solutions is not only crucial for nuclear safety reasons but also for economic reasons — the cost per day of shutdown is in the order of $1,000,000.

We report on a successful project for transference of advanced AI technology into the domain of planning of outages of nuclear power plants, a collaboration between Rome Laboratory, the Electric Power Research Institute, Kaman Science, and Kestrel Institute as part of DOD's dual-use program. The software environment selected for this project was KIDS (Kestrel Interactive Development System)[Smith 91], which is a set of semiautomatic tools to transform declarative problem specifications into correct and efficient programs. The main goal of the project was to evaluate the use of transformational approaches and AI technology to solve real-world planning and scheduling problems involving complex constraints.

ROMAN (Rome Lab Outage Manager) is the prototype system that was developed as a result of this project [Gomes & Smith 96]. ROMAN's main innovation compared to the current state of the art of outage management tools is its capability to automatically enforce safety constraints during the planning and scheduling phase. Another innovative aspect of ROMAN is its generation of more robust schedules that are feasible over time windows. In other words, ROMAN generates a family of schedules by assigning time intervals as start times to activities rather than single point start times, without affecting the overall duration of the project.
Roman uses a rich representation for the state of the plant at any time (as in planning approaches) which allows for efficient constraint-based reasoning, in particular, temporal reasoning (as in scheduling). The problem is modeled as a constraint satisfaction problem combining a global search tactic with constraint propagation. The derivation of very specialized representations for the constraints to perform efficient propagation is a key aspect for the generation of very fast schedules — constraints are compiled into the code, which is a novel aspect of our work using an automatic programming system, KIDS. In order to increase schedule robustness our approach entails the generation of families of schedules with the same completion time and that are feasible over time intervals.

In the next section we describe related work. In section 3 we define the outage problem and in section 4 we discuss the current state-of-the-art of outage management for nuclear power plants and its limitations. Section 5 describes ROMAN in detail. Section 7 summarizes the main results achieved with ROMAN.

## 2 Related Work

Our approach to scheduling uses global search methods as opposed to local search [Gomes & Smith 96]. Local search techniques are based on the idea of improving existing solutions by iteratively making small changes. A local search algorithm defines a walk in which each solution is a neighbor of a previous visited solution. Examples of local search approaches are repair methods (e.g., [Zweben et al 94, Selman & Kautz 93, Minton et al 90]), fix-point iteration[Cai & Paige 89], and linear programming algorithms. Global search methods on the other hand focus on incrementally generating a solution by repeatedly splitting an initial set of solutions into subsets until a feasible or optimal solution can be extracted. Examples of global search methods include backtracking, heuristic search, branch-and-bound. Examples of approaches to scheduling taking a global search perspective are OPIS/DITOPS [Smith 94] Micro-Boss [Sadeh 94].

The main innovation of our approach compared to other AI scheduling approaches is the derivation of very specialized constraints that are compiled into the search and control mechanisms [Gomes & Smith 96]. Other approaches to scheduling use constraint representations and operations that are geared for a broad class of problems, while our approach, a transformational approach, derives specialized representations for constraints allowing fast constraint checking and constraint propagation.

Another novel aspect of our approach is the generation of schedules that are feasible over time windows rather than having single time points as start times. With our approach, we generate an infinite family of schedules that have the same completion time. Existing AI approaches to scheduling with complex state variables only generate a single solution, feasible for single start times, without any guarantees of feasibility over time windows [Gomes & Smith 96].
The framework selected for this project was KIDS (Kestrel Interactive Development System)[Smith 91], which supports users in transforming declarative problem specifications into correct and efficient programs. The transformations provided in KIDS are designed to perform significant and meaningful actions in terms of search efficiency. The various transformations in KIDS include: algorithmic transformations, program optimization techniques and data structures refinement. The algorithmic transformations allow the user to add search and control mechanisms to a given problem specification. Finite differencing is another important transformation provided by KIDS. KIDS uses a form of deductive inference called directed inference to reason about the problem specification in order to automatically apply tactics, derive filters and perform constraint propagation[Smith et al 95].

KIDS has been used to derive a very fast transportation scheduler for the US Transportation Command, KTS (Kestrel Transportation Scheduler) [Smith & Parra 93]. A typical transportation problem with 10,000 movement requirements takes the derived scheduler 1 to 3 minutes to solve, compared with 2.5 hours for a deployed feasibility estimator (JFAST) and 36 hours for deployed schedulers (FLOGEN, ADANS). The computed schedules use relatively few resources and satisfy all specified constraints. The speed of this scheduler was due to the synthesis of strong constraint checking and constraint propagation code [Smith et al 95]. In this paper we show how this approach can be extended to tackle a much richer real-world scheduling task involving complex state variables and time windows.

3 Planning and Scheduling of Nuclear Power Plant Outages

The planning and scheduling of the operations involved in the outages of nuclear power plants has a great impact in terms of the outage costs (replacement power, labor cost, etc.), use of scarce resources and implementation of safety procedures. Prior to 1979, before the accident at Three Mile Island (TMI), refueling was the driving factor of outages of nuclear power plants: maintenance plans were governed by the projected duration of refueling activities. After the TMI accident, the focus turned to improving nuclear power plant effectiveness. The duration of an outage was determined not only by refueling activities, but by the work and plant modifications required to make the plant safer and more effective [PSDI 94, Wallace 90]. Throughout the 1980s, backfits and the aging of nuclear power plants has reversed outage scope priorities and methodologies. Often the refueling activities no longer dictate the critical path in an outage.
3.1 Definition of Problem

The problem of planning and scheduling nuclear power plant outages can be stated as follows:

Given a set of outage activities (refueling operations, repairs, modifications, and maintenance activities), a set of resources, and a set of technological constraints, assign times and resources to the activities in such way that the completion of the outage is minimized while safely performing all the activities required by the outage.

3.1.1 Activities

Depending on the planning and scheduling procedures of each particular plant, as well as the scope of the activities performed during the outage, the planning and scheduling of outages for nuclear power plants might involve from 15,000 up to 45,000 activities. During an outage several activities are performed, such as:

- Refueling operations
- Plant betterment
- Preventive maintenance
- Corrective maintenance
- Technical specification requirements for inspections or surveillance.

Relationships between activities that are explicitly defined in work order activities are temporal relationships (e.g., activity A precedes (follows) activity B). Other constraints between activities arise as a result of different requirements in terms of feasible plans and schedules. Requirements regarding feasible plans and schedules are outlined in the next paragraphs.

3.1.2 Plant Configuration and Risk Management

The general principle underlying the outage procedures is that outages should be as short as possible, maintaining the appropriate level of nuclear safety. In other words, the outage should be planned and managed to reduce shutdown risks through the appropriate consideration of defense in depth and preventive measures. The concept of defense in depth, used for the purpose of managing risk during shutdown consists of:
• providing systems, structures and components to ensure backup of key safety functions using redundant, alternate or diverse methods;

• planning and scheduling outage activities in a manner that optimizes safety system availability

Main safety functions and systems components that are monitored to implement the concept of defense in depth are:

• electricity power control system
• primary and secondary containment
• fuel pool cooling system
• inventory control
• reactivity control
• shutdown cooling
• vital support systems

Figure 1 depicts the decision tree regarding safety levels for a simple safety function, electricity power control.

3.1.3 Resources

The main type of resource taken into consideration when planning and scheduling nuclear power plant outages is labor, organized into different skill groups. Other resources that are considered include:

• reactor building crane
• laydown areas
• water purification and storage systems
• radioactive waste system
• specialized equipment

ROAMN does not include resource assignment.
4 Outage Planning and Scheduling in the Real World

There are approximately 110 nuclear plants operating in the US. Our knowledge about the way outage planning and scheduling is performed in real world environment only considers the power plants that are members of EPRI, the Electric Power Research Institute. Nevertheless, we consider our sample representative since most of the operating nuclear power plants are members of EPRI, with some notable exceptions like Florida Power and Pacific Gas & Electric.
4.1 Time Window Assignment

The current automatic planning and scheduling techniques used by the utilities are very simple - planning and scheduling still heavily relies on the experience of the manual schedulers rather than on automatic procedures. In the late 1970s, utilities began to use the project management techniques to control nuclear refueling outages. Current automatic approaches to outage scheduling mainly consist of the application of automatic project management techniques, such as PERT and CPM techniques. The software currently used by the utilities to perform their outage planning and scheduling tasks are mainly: Primavera Project Planner for Windows (personal computers), Project/2 (mainframes), Project2/X for Windows (personal computers), Prestige (mainframes) and OpenPlan (personal computers).

Some sites use activity based scheduling only. Activities (work orders) and temporal relationships between activities are coded into the software\(^1\) and a PERT/CPM network is generated. The PERT/CPM network can then be manually perturbed to meet resource requirements, safety requirements, and other requirements.

When system windows are used for scheduling, milestones and key events are set up based on experience and the status of key components between the milestones determines the position of the system windows. The activities (work orders) are manually assigned to the system windows where they are allowed to be performed within the scope of the predecessor-successor relationships. PERT/CPM network is generated considering the activities, the pre-defined system windows and the milestones. The PERT/CPM network can then be manually perturbed to meet resource requirements, safety requirements and other requirements.

4.2 Plant Configuration and Risk Management

Safety and risk assessment have been by far manual processes which call on the expertise of the personnel involved to make decisions based on published policies and procedures. In order to ensure that the sequence of activities performed during an outage follows the safety requirements, the schedule produced using PERT/CPM software tools is evaluated using a risk assessment methodology. If the schedule does not meet the safety requirements, manual adjustments have to be performed. ORAM (Outage Risk Assessment Methodology) is one of the most popular software tools used to perform the risk assessment of schedules. It simulates the execution of the schedule keeping track of the configuration of the plant at any time and therefore evaluating the risk inherent to a schedule at any time during its execution.

\(^1\) Actually, the only type of temporal relationship handled by the current software is the relationship before/after with the possibility of definition of slacks.
4.3 Resources

During automatic generation of schedules resources are assumed to be unlimited. Manual adjustments are performed a posteriori in order to meet the resource requirements.

5 ROMAN - Rome Lab. Outage Manager

ROMAN's approach combines a constraint satisfaction paradigm with global search and constraint propagation [Gomes & Smith 96]. ROMAN includes all the technological constraints currently incorporated in the automatic tools used by the utilities for schedule generation. In addition, it includes all the constraints regarding the safety function AC power. Other safety functions could be modeled in a similar way. A top level formal specification of the outage problem including the safety function AC power follows:

\[
\begin{align*}
\text{function} & : \text{safe-outage-windows}(activities) \\
\text{returns} & : (schedule) | \\
& \quad \text{Consistent-Activity-Separation}(schedule) \land \\
& \quad \text{Consistent-AC-power}(schedule) \land \\
& \quad \text{All-activities-scheduled}(activities, schedule))
\end{align*}
\]

In this formulation activities correspond to the set of activities to be performed. Each activity has a given duration, a set of predecessors, and a set of effects on resources. The schedule is a partial order of activities. Activities in the schedule have time windows assigned to it. A time window defines the earliest start time (est) and latest start time (lst) of an activity, such that the activity can start at any time during the window without increasing the overall duration of the project. Given the duration of the activity, the earliest finish time (efl) and latest finish time (lfl) can be calculated. The predicate Consistent-Activity-Separation(schedule) states that all the activities in the schedule satisfy the precedence constraints. The predicate Consistent-ac-power(schedule) states that the schedule verifies the safety constraints, from an AC power point of view. As a completeness condition, the predicate All-activities-scheduled(activities, schedule) states that all the activities have to be scheduled.

The notion of state of the plant is a key concept in enforcing safety constraints. In outage management the state of the plant is measured in colors — green, yellow, orange or red, in this order of increasing risk — and is computed by considering

\[\text{We modeled the AC power safety function as a proof of concept. Other safety functions could be modeled in a similar way.}\]
complex decision trees regarding safety levels as illustrated in Figure 2. For instance, if there is an activity being executed that has the potential to cause AC power loss, then in order for the plant to be in a yellow state it is required to have two off-site AC power sources available and three operable emergency safeguard buses.

![Decision Tree Diagram](image)

**Figure 2: Example of a decision tree for the safety function AC Power**

Since the start times of activities are defined over time windows, we introduce two concepts regarding the execution of an activity: the **definite period** and the **potential period** of an activity. The definite period of an activity corresponds to the period of time during which the activity is definitely being execute — it is the interval of time between the latest start time of the activity (lst) and its earliest finish time (eft). The potential period of an activity corresponds to the period of time during which the activity may be executed — it is the time period between the earliest start time of the activity (est) and its latest finish time (lft). Figure 3 illustrates the notion of definite period of an activity. Notice that activity A does not have a definite period, since its earliest finish time is before its latest start time.
In addition, we define two other concepts: definite state of the plant and potential state of the plant. The definite state of the plant is associated with the concept of definite period: it represents the state of the plant for a given safety function (e.g., AC power) assuming that activities are only executed during their definite period. The concept of potential state of the plant is associated with the concept of potential period of an activity: it represents the state of the plant for a given safety function assuming that activities are executed during the whole extension of their potential periods. The potential state of the plant is always “equal” or “greater” than the state of the plant since the definite period of an activity tends to underestimate the duration of activities while the potential period of an activity tends to overestimate the duration of activities. Figure 4 gives an example. Note that during certain time intervals, the definite and potential states of the plant coincide.

5.1 Search and Control Mechanisms

KIDS provides algorithmic transformations that add control and search mechanisms to a given specification. The search tactic selected for the outage problem was global search (see next section). Figure 5 summarizes the approach adopted in ROMAN [Gomes & Smith 96].

Initially global search is applied to the formal specification of the outage problem in order to generate a schedule, assuming the definite period of activities. Since the notion of definite period tends to underestimate the duration of the activities, it is very likely for the schedule produced in this initial phase not to be feasible from the point of view of the potential state of the plant. In order to enforce the safety threshold for the potential state of the plant at any time during the outage, “refinement” of the time windows of the initial schedule takes place. In the next section, we describe
global search theory.

5.1.1 Global Search Theory

Global search [Smith 87, Smith et al 95] is a backtrack algorithm, a refinement of generate-and-test. The tactic is implemented by finding a space containing all the solutions to the problem that can be divided into nested subspaces. The global search algorithm starts with an initial set that contains all the solutions to the given problem instance, repeatedly extracts solutions, splits sets, and eliminates subsets using propagation, until no sets remain to be split. The process can be described as a tree search in which a node represents a set of candidates, and an arc represents the split relationship between a set and a subset. The principal operations are to extract candidate solutions from a set and to split a set into subsets. The derivation of efficient cutting constraints that eliminate subspaces that do not contain any feasible solution is an important complementary operation in the derivation of the global search tactic.

Figure 6 illustrates the global search theory for the initial scheduling of the activities considering their definite periods. In this global search theory the initial subspace descriptor (partial schedule) is the empty sequence (empty schedule). Splitting corresponds to appending an unscheduled activity, with a given time window, to the partial schedule. Cutting corresponds to propagating the constraints over the time windows of the activities in the partial schedule. Notice that cutting makes the time windows shrink. It can also split a time window as in the case of activity $G$ - due to propagation, activity $G$'s window was split into two. As we can see from figure 6 most of the work in this global search theory is performed by constraint propagation. Splitting corresponds to just selecting the next activity to schedule, using a heuristic
that favors shorter schedules\footnote{We also define a topological sort of the unscheduled activities according to their levels. An activity has level 0 if it has no predecessors. Activities that only have as predecessors activities of level 0 have level 1. Activities of level 2 only have as predecessors activities that have level 0 or 1, etc.}. \textit{Extraction} takes place when all the activities have been scheduled.

The operator \textit{extract} corresponds to the second global search algorithm. Refinement of time windows takes place if after applying the initial global search to the outage problem the potential state of plant does not satisfy the safety requirements. In other words, refinement of time windows is required to enforce the safety constraints over the potential period of all the activities in the initial schedule. This is achieved by applying a new global search to the formal specification of the outage considering now with as input the schedule generated in the initial phase. In this second phase the windows of the activities that contribute to the contention periods, \textit{i.e.}, the periods in which the potential state of the plant is above the safety threshold, are systematically reduced until the potential state of the plant becomes consistent from the safety point of view for all the times during the outage. In this global search theory for the refinement phase splitting corresponds to reducing the size of the windows of the activities involved in the contention periods.

\subsection{Constraint Propagation}

One of the important features of our approach is the propagation of constraints. Figure 7 illustrates the concept, where \textit{psched} is a partial schedule, a set of candidate solutions, a node of the global search tree. The following test states that a partial
Figure 6: Global search theory for the Outage Problem

The schedule can be extended to a complete feasible schedule:\(^{4}\)

\[ \exists \ (sched) \ (sched \in \text{psched} \land \text{feasible}(sched, \text{activities})) \quad (1) \]

However, this test is in general too expensive, computationally. Instead, we derive necessary conditions for (1), filters, i.e.:

\[ \exists (sched)(sched \in \text{psched} \land \text{feasible}(sched, \text{activities}) \implies \Psi(sched, \text{psched})) \quad (2) \]

The next step consists in incorporating the filter derived in (2) into psched, i.e.:

\[ \xi(\text{psched}) \iff \forall (sched)(sched \in \text{psched} \implies \Psi(sched, \text{psched})) \quad (3) \]

The test \(\xi(\text{psched})\) holds when all the candidate solutions in \(\text{psched}\) satisfy \(\Psi\). The main issue is, when a given \(\text{psched}\) does not satisfy \(\xi\), how can we incorporate \(\xi\) into \(\text{psched}\)? The answer is to find the greatest refinement of \(\text{psched, psched}\), that satisfies \(\xi\).

\(^{4}\)In the particular case of the outage problem, \((sched \in \text{psched}) \iff (\text{domain}(\text{psched}) \subseteq \text{domain}(sched) \land \forall (i) \ (i \in \text{domain}(\text{psched})) \implies \text{psched}(i).\text{est} \leq \text{sched}(i).\text{st} \leq \text{psched}(i).\text{lst})\) and \(\text{feasible}(\text{sched, activities}) \iff (\text{consistent-separation}(\text{sched}) \land \text{consistent-acp}(\text{sched})) \land \text{all-activities-scheduled}(\text{activities, sched})\)
\[ \text{psched} = \max \{ q\text{sched} \mid \text{psched} \sqsupseteq q\text{sched} \land \xi(x, q\text{sched}) \} \]  

which asserts that \( \text{psched} \), is maximal over the set of descriptors that refine \( \text{psched} \) and satisfy \( \xi \), with respect to ordering \( \sqsupseteq \). We want \( \text{psched} \) to be a refinement of \( \text{psched} \) so that all of the information in \( \text{psched} \) is preserved and we want \( \text{psched} \) to be maximal so that no other information than \( \text{psched} \) and \( \xi \) is incorporated into \( \text{psched} \). The refinement relation \( \text{psched}_j \sqsupseteq \text{psched}_i \) holds when the completions of \( \text{psched}_i \) are a subset of the completions of \( \text{psched}_j \).

KIDS instantiates a program scheme for global search with constraint propagation, incorporating \( \xi \). For more detail on propagation in KIDS see [Smith et al. 1995]. The challenge in order to take advantage of the propagation mechanisms provided in KIDS lies in finding \( \xi \) - even though KIDS provides a tactic to synthesize propagation code incorporating \( \xi \), the derivation of \( \xi \) using the system relies on lemmas supplied by the user which are derived manually.

In the case of the outage problem, the predicate \textit{Consistent-Activity-Separation} (schedule) states that all the activities in the schedule satisfy the precedence constraints. The derivation of cutting constraints from the constraint \textit{Consistent-Activity-Separation} using formulas (2) and (3) leads to the well known constraints on \textit{est} and \textit{lst}, as used in PERT. Appendix A has the formal derivation of constraints from the constraint \textit{Consistent-Activity-Separation}.

The derivation of cutting constraints for \textit{Consistent-ACP} is less straightforward. Appendix B has the formal derivation of constraints from the constraint \textit{Consistent-Activity-ACP}. An example of a constraint manually inferred from \textit{Consistent-ACP} applying formulas (2) and (3) follows:

\[ \forall(i, t1, t2, act) \]  
\[ i \in \text{domain}(se(\text{psched})) \land t1 = se(\text{psched})(i).\text{time} \land t2 = se(\text{psched})(i + 1).\text{time} \]
\[ act \in \text{domain}(psched) \land \text{sacpl?}(t1, psched) \land \\
\text{unav-sources}(t1, psched) = \text{TSACPL} \land \text{affects-avail-acps?}(act, psched) \\
\Rightarrow \text{psched}(act).lft < t1 \lor \text{psched}(act).est \geq t2 \]

Where \( se(psched) \) computes the state events of the partial schedule considering the definite periods of activities. A state event corresponds to any event that affects the state of the plant. The time of the \( ith \) state event of the partial schedule is represented by \( se(psched)(i).time \), the predicate \( \text{sacpl?}(t, sched) \) tests if at time \( t \) the plant is in a state of AC power loss, \( \text{unav-sources}(t, psched) = \text{TSACPL} \) tests if at time \( t \) the number of unavailable AC power resources equals the threshold for AC power resource unavailability for a state of AC power loss, \( \text{affects-avail-acpsi}(act, psched) \) tests if the activity \( act \) affects an available AC power resource, and \( \text{psched}(act).lft \) and \( \text{psched}(act).est \) correspond respectively to the latest finish time and earliest start time of the activity \( act \) of the partial schedule \( psched \). This constraint triggers propagation for the activities that affect available AC power resources — propagation eliminates from the activities’ time windows the periods that overlap the intervals that correspond to a state of AC power loss with number of unavailable AC power resources equal to \( \text{TSACPL} \) (the threshold). In other words, a new activity that affects available AC power resources cannot occur during a period for which the plant is operating at the threshold regarding the AC power safety function.

### 5.1.3 Interaction Between The Schedule and the State of Plant

A main principle embodied in our approach is incremental computation - propagation illustrates that concept - whenever a new activity is scheduled, all constraints are immediately propagated over the schedule. Finite differencing is another transformation that allows for incremental computation, by efficiently maintaining the state of plant. Roughly, the idea behind finite differencing is to incrementally evaluate an expensive expression in a loop, rather than recomputing it from scratch each time. As an example, let us assume that function \( f(x) \) calls function \( g(x) \) and that \( x \) changes in a regular way. In this case, it might be worthwhile to create a new variable, whose value is maintained and which allows for incremental computation. By abstracting function \( f \) with respect to expression \( g(x) \) a new parameter \( c \) is added to \( f \)'s parameter list (now \( f(x, c) \)) and \( c = g(x) \) is added as a new input invariant to \( f \). Any call to \( f \), whether a recursive call within \( f \) or an external call, must now be changed to supply the appropriate new argument that satisfies the invariant - \( f(x) \) is changed to \( f(x, g(x)) \). In this process all occurrences of \( g(x) \) are replaced by \( c \). Often, distributive laws\(^5\) apply to \( g(h(x)) \) yielding an expression of the form \( h'(g(x)) \) and so \( h'(c) \).

\(^5\)Laws are assertions that define axioms or theorems, i.e., statements that are always true. An assertion is simply a true statement - an example of a law is \( (A + B) \ast C = (A \ast C) + (B \ast C) \), or \( A \land B \rightarrow A \). The idea is to provide information on how to distribute predicates and functions over the main constructors of the variable that changes in a regular way, exactly in the same way.
The real benefit in the optimization comes from the last step, because this is where the new value of the expression \( g(h(x)) \) is computed in terms of the old value of \( g(x) \).

![Diagram](image)

Figure 8: Interaction between the schedule and the state

In the outage problem there are several opportunities for finite differencing since the state of the plant is a function of the schedule represented by the constraint \textit{consistent-ACP(schedule)}. Figure 8 shows the interactions between the state of plant and the schedule - when a new activity is scheduled, it impacts the schedule and propagation is triggered. Changes in the schedule impact the state of the plant, which is incrementally maintained by finite differencing. Changes in the state impact the schedule and propagation is triggered, which impacts the schedule and so on. The key issue to take advantage of finite differencing is to provide good laws on how to distribute the functions to be finite differenced over the main constructors of the partial schedule, e.g., over appending an activity to the schedule, increasing the \textit{est} of an activity, etc.

Appendix A and appendix B contain the formal derivation of cutting constraints from the separation constraint and from the ACP constraint, respectively. Appendix C contains the domain theory for the outage problem considering the separation constraints and the constraints for ACP. Appendix D contains the global search theory for scheduling considering the definite period of activities as well as the global search theory for time window refinement that takes into consideration the potential period of activities.

6 Performance Results

The current version of ROMAN was completed in November 1995, and it has been demonstrated to several large nuclear power plants such as American Electric Power Service, Baltimore Gas & Electric, PECO Energy, etc. The demonstration was successful, and EPRI, a consortium of more than 90% of the utilities in the US, is looking

one would write a law about how to distribute multiplication over addition. Additionally, laws also specify special cases, for instance when dealing with base cases (e.g., empty sequences).
into using the approach embodied in ROMAN to build the next generation of outage scheduling tools — referred to as Advanced Technology Outage Scheduler.

ROMAN has proven successful since it clearly extends the current functionality offered by existing software tools for outage management. All the technological constraints currently used for automatic schedule generation are incorporated into the system. In addition, ROMAN produces schedules enforcing safety constraints — AC power was used as a proof of concept.

The current version of ROMAN schedules up to 2,000 activities in approximately 1 minute on a Sparc 2 (see figure 9). The schedules produced by ROMAN are often better than the current solutions since many new possibilities are explored compared to manual solutions. Human schedulers tend to aggregate tasks and schedule them as blocks rather than exploring interesting possibilities that occur when the activities are scheduled separately.

A key feature of ROMAN that utility personnel find attractive is the robust schedules that are generated. The current scheduler generates a schedule that includes start time windows for each task. Choosing any start time within the window for a task still permits feasible execution of the schedule. The window provides information about how critical the start time for a task is — if a predecessor task is delayed, a user can decide whether there still enough freedom in the start time window to allow on-time completion, or whether it is time to reschedule parts of the overall operation.

ROMAN currently comes configured with a GUI that displays an interactive Gantt chart for tasks, showing their start time window, duration, task description, and
predecessors. Another Gantt chart shows the history of the state of the plant with respect to AC power.

7 Conclusions and Future Work

ROMAN has successfully demonstrated that outage schedules that satisfy safety constraints can be automatically generated [Gomes & Smith 96]. To develop ROMAN into a practical tool requires (1) handling a richer model of the outage domain, and (2) faster code. To date we have focused on one particular safety function dealing with maintaining adequate sources of AC power. Future work is planned to deal with larger and more realistic problems, as well as with other critical safety constraints and scheduling scarce resources such as heavy lifts and skilled personnel. Furthermore, we plan to experiment other search strategies, in particular local search strategies. A more ambitious goal involves the automatic generation of schedules considering different levels of risk.

8 Acknowledgments

I would like to thank several people who contributed to the success of ROMAN. Lou Hoebel for creating the conditions and putting together the resources for this project. Karen Alguire for her contribution to this project, in particular helping with the programming. Doug Smith for his contribution to the design of the domain theory for the outage problem in order to take advantage of KIDS [Gomes & Smith 96]. Stephen Westfold for extending the constraint propagation code generator in KIDS to encompass the new class of propagation constraints that arose from the safety constraints. Eduardo Parra for adapting the CLIM interface of KTS to the needs of outage scheduling, resulting in the current ROMAN interface. Jeff Mitman, Dick Wood, and Shyam Kamadolli were instrumental in providing data and information about outage management.
9 References


Appendix A - Derivation of Constraints from the Separation Constraint

Derivation of $\xi(x, \hat{r})$

The derivation of cutting constraints involves two steps.

The first step consists of deriving necessary conditions for the feasibility of a solution $z$ contained in a space descriptor $\hat{r}$, i.e.:

$$\forall(x : D, \hat{r} : \hat{R}, z : R)(\text{Satisfies}(z, \hat{r}) \land O(x, z) \implies \Psi(x, z, \hat{r}))$$  \hspace{1cm} (5)

The second step consists of making sure that the space descriptors (themselves) satisfy the necessary conditions for containing feasible solutions, i.e.:

$$\xi(x, \hat{r}) \iff \forall(z : R)(\text{Satisfies}(z, \hat{r}) \implies \Psi(x, z, \hat{r}))$$  \hspace{1cm} (6)

In the following section we describe the derivation of $\xi(x, \hat{r})$ for the safety constraint designated by Consistent-Separation.

Satisfies and Output Condition for the Outage Problem

Conventions:

- $\text{psched}$ - partial schedule
- $\text{sched}$ - schedule

Def : $\text{Satisfies}(\text{sched}, \text{psched})$

$$\text{domain}(\text{psched}) \subseteq \text{domain}(\text{sched}) \land$$
$$\forall(i)$$
$$i \in \text{domain}(\text{psched})$$
$$\implies \text{psched}(i).est \leq \text{sched}(i).st \leq \text{psched}(i).lst$$

Def : $O(\text{acts}, \text{psched})$

$$\text{consistent-separation}(\text{sched}) \land$$
consistent-acp(sched)) \land 
all-activities-scheduled(acts, sched)

Derivation of Cutting Constraint for Separation

Given the following definition for a separation constraint:

\textbf{Def : consistent-separation(sched)}

\[ \forall(i, j) \]
\[ i \in \text{domain}(sched) \land j \in \text{domain}(sched(i), pred) \]
\[ \implies sched(i).st \geq sched(i).pred(j).st + sched(i).pred(j).duration \]

Since:

\[ (A \implies B) \land (B \implies B') \]
\[ \implies (A \implies B') \]

And by definition of Sat:

\[ sched(i).st \geq sched(j).st + sched(j).duration \]
\[ \implies psched(i).lst \geq sched(i).pred(j).st + sched(i).pred(j).duration \]

And since:

\[ sched(i).pred(j).duration = psched(i).pred(j).duration \]

And Assuming:

\[ sched(i).pred(j) = sched(j) \]
\[ psched(i).pred(j) = psched(j) \]
\[ sched(i).pred(j).duration = sched(j).duration \]
\[ psched(i).pred(j).duration = psched(j).duration \]

\[ \implies \]

\[ \forall(i, j) \]
\[ i \in \text{domain}(psched) \land j \in \text{domain}(psched(i), pred) \]
\[ \implies psched(i).lst \geq sched(j).st + psched(i).duration \]

Which corresponds to a \textit{cutting constraint}, derived from \textit{consistent-separation(sched)}.\footnote{As we will show below, another \textit{cutting constraint} can be derived from \textit{consistent-separation(sched)}.}
Instantiation of $\xi$ for the cutting constraint for consistent-separation($sched$)).

$$\xi(x, \hat{r}) \iff \forall(z : R)(Satisfies(z, \hat{r}) \Rightarrow \Psi(x, z, \hat{r}))$$  \hspace{1cm} (7)

$$\text{( Satisfies}(z, \hat{r}) \land O(x, z))$$  \hspace{1cm} (8)

In the outage problem, the expressions for $Satisfies(z, \hat{r})$ and $\Psi(x, z, \hat{r})$ (for the consistent-separation) are:

**Def :** $Satisfies(sched, psched)$

$$\text{domain}(psched) \subseteq \text{domain}(sched) \land \forall(i) \quad i \in \text{domain}(psched) \Rightarrow psched(i).est \leq sched(i).st \leq psched(i).lst$$

**Def :** $\psi(acts, sched, psched)$ (for the consistent-acp)

$$\forall(i, j) \quad i \in \text{domain}(psched) \land j \in \text{domain}(psched(i).pred) \Rightarrow psched(i).lst \geq sched(j).st + psched(j).duration$$

Combining both we can instantiate $\xi(\text{act, psched})$ i.e.,

$$\xi(\text{act, psched}) \iff \forall(sched)(Satisfies(sched, psched) \Rightarrow \Psi(\text{act, sched, psched}))$$  \hspace{1cm} (9)

**Def :** $\xi(\text{act, psched})$

$$\forall(sched) \quad \text{domain}(psched) \subseteq \text{domain}(sched) \land \forall(i) \quad i \in \text{domain}(psched) \Rightarrow psched(i).est \leq sched(i).st \leq psched(i).lst \Rightarrow \forall(k, j) \quad k \in \text{domain}(psched) \land j \in \text{domain}(psched(k).pred) \Rightarrow psched(k).lst \geq sched(j).st + psched(j).duration$$
Since: \( A \land B \implies C \)
\[\iff\]
\[A \implies B \implies C\]

\(\forall(sched)\)
\(\text{domain}(psched) \subseteq \text{domain}(sched)\)
\[\implies \forall(i)\]
\[i \in \text{domain}(psched))\]
\[\implies psched(i).est \leq sched(i).st \leq psched(i).lst\]
\[\implies \forall(k,j)\]
\[k \in \text{domain}(psched) \land j \in \text{domain}(psched(k).pred)\]
\[\implies psched(k).lst \geq sched(j).st + psched(j).duration\]

Since: \( j \in \text{domain}(psched(i).pred) \subseteq \text{domain}(psched)\)

\[\iff\]
\(\forall(sched)\)
\(\text{domain}(psched) \subseteq \text{domain}(sched)\)
\[\implies \forall(i)\]
\[i \in \text{domain}(psched))\]
\[\implies psched(i).est \leq sched(i).st \leq psched(i).lst\]
\[\implies \forall(k,j)\]
\[k \in \text{domain}(psched) \land j \in \text{domain}(psched) \land j \in \text{domain}(psched(k).pred)\]
\[\implies psched(k).lst \geq sched(j).st + psched(j).duration\]

Since: \( A \land B \implies C \)
\[\iff\]
\[A \implies B \implies C\]

\[\iff\]
\(\forall(sched)\)
\(\text{domain}(psched) \subseteq \text{domain}(sched)\)
\[\implies \forall(i)\]
\[i \in \text{domain}(psched))\]
\[ \implies psched(i).est \leq sched(i).st \leq psched(i).lst \]
\[ \implies \forall(j) \]
\[ j \in \text{domain}(psched) \]
\[ \implies \forall(k) \]
\[ k \in \text{domain}(psched) \land \land j \in \text{domain}(psched(k).pred) \]
\[ \implies psched(k).lst \geq sched(j).st + psched(j).duration \]

Since:
\[ \forall(K)K \in S \implies P(K) \]
\[ \implies \forall(K)K \in S \land T(K) \implies Q(K) \]
\[ \iff \]
\[ \forall(K)K \in S \implies P(K) \implies Q(K) \]

\[ \iff \]
\[ \forall(sched) \]
\[ \text{domain}(psched) \subseteq \text{domain}(sched) \]
\[ \implies \forall(i) \]
\[ i \in \text{domain}(psched) \]
\[ \implies psched(i).est \leq sched(i).st \leq psched(i).lst \]
\[ \implies \forall(k) \]
\[ k \in \text{domain}(psched) \land \land i \in \text{domain}(psched(k).pred) \]
\[ \implies psched(k).lst \geq sched(i).st + psched(i).duration \]

\[ \iff \]
\[ \forall(sched) \]
\[ \text{domain}(psched) \subseteq \text{domain}(sched) \]
\[ \implies \forall(i) \]
\[ i \in \text{domain}(psched) \]
\[ \implies psched(i).est \leq sched(i).st \leq psched(i).lst \]
\[ \implies \forall(k) \]
\[ k \in \text{domain}(psched) \land \land i \in \text{domain}(psched(k).pred) \]
\[ \implies psched(k).lst \geq sched(i).st + psched(i).duration \]

Since:
\[ A \implies B \implies C \]
\[ \iff \]

25
\[ A \land B \implies C \]

\[ \iff \]

\[ \forall(sched) \\
\text{domain}(psched) \subseteq \text{domain}(sched) \\
\implies \forall(i) \\
i \in \text{domain}(psched) \\
\implies psched(i).est \leq sched(i).st \leq psched(i).lst \\
\land \forall(k) \\
k \in \text{domain}(psched) \land i \in \text{domain}(psched(k).pred) \\
\implies psched(k).lst \geq sched(i).st + psched(i).duration \]

Since: \[ A \implies (B \land C) \]

\[ \implies (A \land B) \implies C \]

\[ \iff \]

\[ \forall(sched) \\
\text{domain}(psched) \subseteq \text{domain}(sched) \\
\implies \forall(i,k) \\
i \in \text{domain}(psched) \land k \in \text{domain}(psched) \\
\land i \in \text{domain}(psched(k).pred) \land psched(i).est \leq sched(i).st \\
\implies sched(i).st \leq psched(i).lst \\
\implies psched(k).lst \geq sched(i).st + psched(i).duration \]

\[ \iff \]

\[ \forall(sched) \\
\text{domain}(psched) \subseteq \text{domain}(sched) \\
\implies \forall(i,k) \\
i \in \text{domain}(psched) \land k \in \text{domain}(psched) \\
\land i \in \text{domain}(psched(k).pred) \land psched(i).est \leq sched(i).st \\
\implies sched(i).st \leq psched(i).lst \\
\implies sched(i).st \leq psched(k).lst - psched(i).duration \]

Replacing sched with qsched:
\begin{align*}
domain(qsched) &= \domain(psched) \\
\forall (a) \ a \in \domain(qsched) \\
\implies qsched(a) &= sched(a)
\end{align*}
(Assuming that each activity has the index in psched, qsched, and sched)

Since: \quad (A' \implies A) \land (A \implies B) \\
\implies \\
(A' \implies B)

\begin{align*}
\forall(qsched) \\
\domain(psched) \subseteq \domain(qsched) \\
\implies \forall (i, k) \\
i \in \domain(psched) \land k \in \domain(psched) \\
\land i \in \domain(psched(k).\text{pred}) \land \text{sched}(i).\text{est} \leq \text{sched}(i).\text{st} \\
\implies \text{sched}(i).\text{st} \leq \text{sched}(i).\text{lst} \\
\implies \text{sched}(i).\text{st} \leq \text{sched}(k).\text{lst} - \text{sched}(i).\text{duration}
\end{align*}

Since: \quad \forall(m) \ S(m) \\
\implies \\
\forall (a) \ a \in \domain(m) \land T(a) \\
\implies \ (m(a) \leq p(a) \implies m(a) \leq q(a))

\iff \\
\forall (a) \ a \in \domain(p) \land T(a) \\
\implies \ (p(a) \leq q(a))

\iff \\
\forall (i, k) \\
i \in \domain(psched) \land k \in \domain(psched) \\
\land i \in \domain(psched(k).\text{pred}) \\
\implies \text{sched}(i).\text{lst} \leq \text{sched}(k).\text{lst} - \text{sched}(i).\text{duration}
\[ \forall (i, k) \]
\[ i \in \text{domain}(\text{sched}) \land k \in \text{domain}(\text{sched}) \]
\[ \land i \in \text{domain}(\text{sched}(k).\text{pred}) \]
\[ \implies \text{sched}(i).\text{lst} + \text{sched}(i).\text{duration} \leq \text{sched}(k).\text{lst} \]

Since: \( \text{sched}(i).\text{lst} + \text{sched}(i).\text{duration} = \text{sched}(i).\text{lft} \)

\[ \iff \]

\[ \forall (i, k) \]
\[ i \in \text{domain}(\text{sched}) \land k \in \text{domain}(\text{sched}) \]
\[ \land i \in \text{domain}(\text{sched}(k).\text{pred}) \]
\[ \implies \text{sched}(i).\text{lft} \leq \text{sched}(k).\text{lst} \]

Since: \( j \in \text{domain}(\text{sched}(i).\text{pred}) \subseteq \text{domain}(\text{sched}) \)

\[ \iff \]

\[ \forall (i, k) \]
\[ k \in \text{domain}(\text{sched}) \]
\[ \land i \in \text{domain}(\text{sched}(k).\text{pred}) \]
\[ \implies \text{sched}(i).\text{lft} \leq \text{sched}(k).\text{lst} \]
Appendix B - Derivation of Constraints from the ACP constraint

Derivation of $\xi(x, \hat{r})$

The derivation of cutting constraints involves two steps.

The first step consists of deriving necessary conditions for the feasibility of a solution $z$ contained in a space descriptor $\hat{r}$, i.e.:

$$\forall (x : D, \hat{r} : \hat{R}, z : R)(\text{Satisfies}(z, \hat{r}) \land O(x, z) \implies \Psi(x, z, \hat{r}))$$  \hspace{1cm} (10)

The second step consists of making sure that the space descriptors (themselves) satisfy the necessary conditions for containing feasible solutions, i.e.:

$$\xi(x, \hat{r}) \iff \forall (z : R)(\text{Satisfies}(z, \hat{r}) \implies \Psi(x, z, \hat{r}))$$  \hspace{1cm} (11)

In the following section we describe the derivation of $\xi(x, \hat{r})$ for the safety constraint designated by AC Power.

Satisfies and Outuput Condition for the Outage Problem

Conventions:

- $es(sched)$ - event sequence - sequence of events of sched
- $sacpl?(t, sched)$ - is the state a state of acp loss, at time $t$ given sched?
- $avacr(t, sched)$ - number of acp resources available, at time $t$, given sched
- $ACPR$ - set of ACP resources
- $av?(x, t, sched)/unav?(x, t, sched)$- is resource $x$ available (unavailable) at time $t$, given sched?
- $affects?(x, y)$ does activity $x$ affect resource $y$?

Def : Satisfies(sched, psched)

\[
\text{domain}(psched) \subseteq \text{domain}(sched) \land \\
\forall (i : i \in \text{domain}(psched)) \land \\
\text{psched}(i).est \leq \text{sched}(i).st \leq \text{psched}(i).lst
\]
Def : \( O(acts, psched) \)

\[
\begin{align*}
\text{consistent-separation}(sched) \land \\
\text{consistent-acp}(sched) \land \\
\text{all-activities-scheduled}(acts, sched)
\end{align*}
\]

Derivation of Cutting Constraint for AC Power

Given the following definition for a safety constraint for AC Power:

Def : \( \text{consistent-acp}(sched) \)

\[
\forall(i, t_1, t_2) \\
i \in \text{domain}(es(sched)) \land t_1 = es(sched)(i) \land t_2 = es(sched)(i + 1) \\
\implies \text{sacpl}(t, sched) \\
\implies \text{avacpr}(< t_1, t_2 >, sched) \geq T'
\]

Since: \( A \implies B \implies C \)

\[\iff\]

\[A \land B \implies C\]

\[
\iff
\forall(i, t_1, t_2) \\
i \in \text{domain}(es(sched)) \land t_1 = es(sched)(i) \land t_2 = es(sched)(i + 1) \\
\land \text{sacpl}(t, sched) \\
\implies \text{avacpr}(< t_1, t_2 >, sched) \geq T'
\]

Since: \( \text{avacpr}(< t_1, t_2 >, sched) \geq T' \)

\[\iff\]

\[\text{unavacpr}(< t_1, t_2 >, sched) \leq TACPR - T' = T)\]

\[
\iff
\forall(i, t_1, t_2) \\
i \in \text{domain}(es(sched)) \land t_1 = es(sched)(i) \land t_2 = es(sched)(i + 1)
\]
\[ \land \text{sacpl}(t, \text{sched}) \]
\[ \implies \text{unavacpr}(< t_1, t_2 >, \text{sched}) \leq T \]

Since: \[ \sum_{x \in S \land P(x)} 1 \leq T \]

\[ \iff \]
\[ \forall (R) \]
\[ R \subseteq S \land |R| = T \land \sum_{x \in R \land P(x)} 1 = T \land \]
\[ \forall (y) y \in S/R \]
\[ \implies \neg P(y) \]

\[ \iff \]
\[ \forall (i, t_1, t_2) \]
\[ i \in \text{domain} (es(\text{sched})) \land t_1 = es(\text{sched})(i) \land t_2 = es(\text{sched})(i + 1) \land \text{sacpl}(t, \text{sched}) \]
\[ \implies \forall (R) R \subseteq ACPR \land |R| = T \land \sum_{x \in R} \land \text{unav}(x, < t_1, t_2 >, \text{sched}) 1 = T \land \]
\[ \forall (y) y \in ACPR/R \]
\[ \implies \text{av}(y, < t_1, t_2 >, \text{sched}) \]

Since: \[ \text{av}(y, < t_1, t_2 >, \text{sched}) \]

\[ \iff \]
\[ \forall (j) j \in \text{domain} (\text{sched}) \land \text{affects}(j, y) \]
\[ \implies \neg \text{during}(j, < t_1, t_2 >) \]

\[ \iff \]
\[ \forall (i, t_1, t_2) \]
\[ i \in \text{domain} (es(\text{sched})) \land t_1 = es(\text{sched})(i) \land t_2 = es(\text{sched})(i + 1) \land \text{sacpl}(t, \text{sched}) \]
\[ \implies \forall (R) R \subseteq ACPR \land |R| = T \land \sum_{x \in R} \land \text{unav}(x, < t_1, t_2 >, \text{sched}) 1 = T \land \]
\[ \forall (y) y \in ACPR/R \]
\[ \implies \forall (j) j \in \text{domain} (\text{sched}) \land \text{affects}(j, y) \]
\[ \implies \neg \text{during}(j, < t_1, t_2 >) \]
Substituting \(\neg \text{during}(j, < t_1, t_2 >)\)

\[
\forall(i, t_1, t_2) \\
i \in \text{domain}(es(sched)) \land t_1 = es(sched)(i) \land t_2 = es(sched)(i + 1) \\
\land \text{sacpl?}(t, sched) \\
\implies \forall(R) R \subseteq ACPR \land |R| = T \land \sum_{x \in R} \neg \text{unav?}(x, <t_1, t_2>, sched) 1 = T \\
\land \forall(y) y \in ACPR/R \\
\implies \forall(j) j \in \text{domain}(sched) \land \text{affects?}(j, y) \\
\implies \text{sched}(j).ft < t_1 \lor \text{sched}(j).st \geq t_2
\]

Since: \(A \implies B \implies C\)

\[
\iff \\
A \land B \implies C
\]

\[
\iff \\
\forall(i, t_1, t_2) \\
i \in \text{domain}(es(sched)) \land t_1 = es(sched)(i) \land t_2 = es(sched)(i + 1) \\
\land \text{sacpl?}(t, sched) \land \forall(R) R \subseteq ACPR \land |R| = T \land \\
\sum_{x \in R} \neg \text{unav?}(x, <t_1, t_2>, sched) 1 = T \land \forall(y) y \in ACPR/R \\
\land \forall(j) j \in \text{domain}(sched) \land \text{affects?}(j, y) \\
\implies \text{sched}(j).ft < t_1 \lor \text{sched}(j).st \geq t_2
\]

Pulling out all the quantifiers:

\[
\iff \\
\forall(i, t_1, t_2, j, R, y) \\
i \in \text{domain}(es(sched)) \land t_1 = es(sched)(i) \land t_2 = es(sched)(i + 1) \\
\land j \in \text{domain}(sched) \land R \subseteq ACPR \land |R| = T \land \\
\sum_{x \in R} \neg \text{unav?}(x, <t_1, t_2>, sched) 1 = T \land y \in ACPR/R \\
\land \text{sacpl?}(t, sched) \land \text{affects?}(j, y) \\
\implies \text{sched}(j).ft < t_1 \lor \text{sched}(j).st \geq t_2
\]
Since: \((A' \implies A) \land (A \implies B)\)
\[
\implies (A' \implies B)
\]

And: \(\) (assuming that each activity has the same index in \(p\)shed and \(s\)ched)
\[\text{domain}(p\text{shed}) \subseteq \text{domain}(s\text{ched})\]
\[
\text{domain}(e\text{shed}(p\text{shed})) \subseteq \text{domain}(e\text{shed}(s\text{ched}))
\]
\[
\forall (i) i \in \text{domain}(p\text{shed}) \land p\text{shed}(p\text{shed})(i) \implies p\text{shed}(p\text{shed})(i) \subseteq e\text{shed}(s\text{ched})(i)
\]
\[
\forall (t_1, t_2) \text{unav?}(x, <t_1,t_2>, p\text{shed}) \implies \text{unav?}(x, <t_1,t_2>, s\text{ched})
\]
\[
\implies
\]
\[
\forall (i, t_1, t_2, j, R, y) i \in \text{domain}(p\text{shed}(p\text{shed})) \land t_1 = p\text{shed}(p\text{shed})(i) \land t_2 = p\text{shed}(p\text{shed})(i + 1) \land j \in \text{domain}(p\text{shed}(p\text{shed})) \land R \subseteq ACPR \land |R| = T \land \sum_{x \in R} \text{unav?}(x, <t_1,t_2>, p\text{shed}) = T \land y \in ACPR/R \land \text{affects?}(j, y) \implies \text{sched}(j).ft < t_1 \lor \text{sched}(j).st \geq t_2
\]

Replacing \(<t_1, t_2>\) with \(<UB, LB \rangle \subset <t_1, t_2>\), where:

given a generic time \(t\) (\(t_1\) or \(t_2\)), \(act\) the corresponding activity that triggers that event at time \(t\), \(type-event = s\) if the event corresponds to the start of \(act\), \(type-event = f\) otherwise:

\[
UB =
\]
\[
\text{If type-event} = s
\]
\[
\text{Then act.lst}
\]
\[
\text{Else act.lft}
\]
\[
(\text{Notice that} \ UB \geq t_1)
\]
\[
LB =
\]

33
If type-event = s
Then act.est
Else act.eft
(Notice that LB ≥ t₂)

And since:

\[(A \implies B) \land (B \implies B') \implies (A \implies B')\]

\[\forall (i, t₁, t₂, j, R, y)
\quad i \in \text{domain}(\text{pes}(\text{sched})) \land t₁ = \text{pes}(\text{sched})(i) \land t₂ = \text{pes}(\text{sched})(i + 1)
\quad \land j \in \text{domain}(\text{sched}) \land R \subseteq \text{ACPR} \land |R| = T \land
\quad \sum_{x \in R} \text{unav}(x, <t₁, t₂>, \text{sched}) = T \land y \in \text{ACPR}/R
\quad \land \text{sacp}(t, \text{sched}) \land \text{affects}(j, y)
\implies \text{sched}(j).ft < UB \lor \text{sched}(j).st \geq LB\]

Which corresponds to the cutting constraint, derived from consistent-acp(sched)).

\textbf{Instantiation of \(\xi\) for the cutting constraint for consistent-acp(sched)).}

\[\xi(x, \hat{r}) \iff \forall (z : R)(\text{Satisfies}(z, \hat{r}) \implies \Psi(x, z, \hat{r}))\]  \hspace{1cm} (12)

In the outage problem, the expressions for Satisfies\((z, \hat{r})\) and \(\Psi(x, z, \hat{r})\) (for the consistent-acp) are:

\textbf{Def : Satisfies\((sched, sched)\)}

\[\text{domain}(\text{sched}) \subseteq \text{domain}(\text{sched}) \land
\quad \forall (i)
\quad i \in \text{domain}(\text{sched})
\implies \text{sched}(i).est \leq \text{sched}(i).st \leq \text{sched}(i).lst\]

\textbf{Def : \(\psi(acts, sched, sched)\) (for the consistent-acp)}

34
\[ \forall(i, t_1, t_2, j, R, y) \]
\[ i \in \text{domain}(\text{pes}(\text{sched})) \land t_1 = \text{pes}(\text{sched})(i) \land t_2 = \text{pes}(\text{sched})(i + 1) \]
\[ \land j \in \text{domain}(\text{sched}) \land R \subseteq \text{ACPR} \land |R| = T \land \]
\[ \sum_{x \in R} \text{unav}(x, t_1, t_2) \cdot \text{sched} = T \land y \in \text{ACPR}/R \]
\[ \land \text{sacpl}(t, \text{sched}) \land \text{affects}(j, y) \]
\[ \implies \text{sched}(j).ft < UB \lor \text{sched}(j).st \geq LB \]

Combining both we can instantiate \( \xi(\text{act}, \text{sched}) \) i.e.,

\[ \xi(\text{act}, \text{sched}) \iff \forall(\text{sched})(\text{Satisfies}(\text{sched}, \text{sched}) \implies \Psi(\text{act}, \text{sched}, \text{sched})) \]

(13)

**Def :** \( \xi(\text{act}, \text{sched}) \)

\[ \forall(\text{sched}) \]
\[ \text{domain}(\text{sched}) \subseteq \text{domain}(\text{sched}) \land \]
\[ \forall(i) \]
\[ i \in \text{domain}(\text{sched}) \]
\[ \implies \text{sched}(i).est \leq \text{sched}(i).st \leq \text{sched}(i).lst \]
\[ \implies \forall(k, t_1, t_2, j, R, y) \]
\[ k \in \text{domain}(\text{pes}(\text{sched})) \land t_1 = \text{pes}(\text{sched})(k) \land t_2 = \text{pes}(\text{sched})(k + 1) \]
\[ \land j \in \text{domain}(\text{sched}) \land R \subseteq \text{ACPR} \land |R| = T \land \]
\[ \sum_{x \in R} \text{unav}(x, t_1, t_2) \cdot \text{sched} = T \land y \in \text{ACPR}/R \]
\[ \land \text{sacpl}(t, \text{sched}) \land \text{affects}(j, y) \]
\[ \implies \text{sched}(j).ft < UB \lor \text{sched}(j).st \geq LB \]

Since:
\[ A \land B \implies C \]
\[ \iff \quad A \implies B \implies C \]

\[ \iff \quad \forall(\text{sched}) \]
\[ \text{domain}(\text{sched}) \subseteq \text{domain}(\text{sched}) \]
\[ \implies \forall(i) \]
\[ i \in \text{domain}(\text{sched}) \]
\[ \implies \text{sched}(i).est \leq \text{sched}(i).st \leq \text{sched}(i).lst \]
\[ \forall(k, t_1, t_2, j, R, y) \]
\[ k \in \text{domain}(\text{pes}(\text{sched})) \land t_1 = \text{pes}(\text{sched})(k) \land t_2 = \text{pes}(\text{sched})(k + 1) \land j \in \text{domain}(\text{sched}) \land R \subseteq \text{ACPR} \land |R| = T \land \sum_{x \in R} \text{unav}(x, <t_1, t_2>, \text{sched}) 1 = T \land y \in \text{ACPR}/R \land \text{sacpl}(t, \text{sched}) \land \text{affects}(j, y) \rightarrow \text{sched}(j).ft < UB \lor \text{sched}(j).st \geq LB \]

Rearranging quantifiers

\[ \Rightarrow \]
\[ \forall(sched) \]
\[ \text{domain}(\text{sched}) \subseteq \text{domain}(\text{sched}) \]
\[ \Rightarrow \forall(i) \]
\[ i \in \text{domain}(\text{sched}) \]
\[ \Rightarrow \text{sched}(i).est \leq \text{sched}(i).st \leq \text{sched}(i).lst \]
\[ \Rightarrow \forall(j) j \in \text{domain}(\text{sched}) \land \forall(k, t_1, t_2, R, y) \]
\[ k \in \text{domain}(\text{pes}(\text{sched})) \land t_1 = \text{pes}(\text{sched})(k) \land t_2 = \text{pes}(\text{sched})(k + 1) \land R \subseteq \text{ACPR} \land |R| = T \land \sum_{x \in R} \text{unav}(x, <t_1, t_2>, \text{sched}) 1 = T \land y \in \text{ACPR}/R \land \text{sacpl}(t, \text{sched}) \land \text{affects}(j, y) \rightarrow \text{sched}(j).ft < UB \lor \text{sched}(j).st \geq LB \]

Since: \[ A \land B \Rightarrow C \]
\[ \equiv \]
\[ A \Rightarrow B \Rightarrow C \]

\[ \Leftrightarrow \]
\[ \forall(sched) \]
\[ \text{domain}(\text{sched}) \subseteq \text{domain}(\text{sched}) \]
\[ \Rightarrow \forall(i) \]
\[ i \in \text{domain}(\text{sched}) \]
\[ \Rightarrow \text{sched}(i).est \leq \text{sched}(i).st \leq \text{sched}(i).lst \]
\[ \Rightarrow \forall(j) j \in \text{domain}(\text{sched}) \]
\[ \Rightarrow \forall(k, t_1, t_2, R, y) \]
\[ k \in \text{domain}(\text{pes}(\text{sched})) \land \]
\[ t_1 = \text{pes} (\text{psched}) (k) \land t_2 = \text{pes} (\text{psched}) (k + 1) \]
\[ \land R \subseteq \text{ACPR} \land |R| = T \land \]
\[ \sum_{x \in R} \text{unav} (x, t_1, t_2, \text{psched}) 1 = T \land y \in \text{ACPR} / R \]
\[ \land \text{sacpl} (t, \text{psched}) \land \text{affects} (j, y) \]
\[ \implies \text{sched} (j). ft < UB \lor \text{sched} (j). st \geq LB \]

Since:
\[ \forall (K) K \in S \implies P (K) \]
\[ \implies \]
\[ \forall (K) K \in S \implies Q (K) \]
\[ \iff \]
\[ \forall (K) K \in S \implies P (K) \implies Q (K) \]
\[ \iff \]
\[ \forall (\text{sched}) \]
\[ \text{domain} (\text{psched}) \subseteq \text{domain} (\text{sched}) \]
\[ \implies \forall (i) \]
\[ i \in \text{domain} (\text{psched}) \]
\[ \implies \text{sched} (i). est \leq \text{sched} (i). st \leq \text{psched} (i). lst \]
\[ \implies \forall (k, t_1, t_2, R, y) \]
\[ k \in \text{domain} (\text{pes} (\text{psched})) \land \]
\[ t_1 = \text{pes} (\text{psched}) (k) \land t_2 = \text{pes} (\text{psched}) (k + 1) \]
\[ \land R \subseteq \text{ACPR} \land |R| = T \land \]
\[ \sum_{x \in R} \text{unav} (x, t_1, t_2, \text{psched}) 1 = T \land y \in \text{ACPR} / R \]
\[ \land \text{sacpl} (t, \text{psched}) \land \text{affects} (i, y) \]
\[ \implies \text{sched} (i). ft < UB \lor \text{sched} (i). st \geq LB \]

Since:
\[ A \implies B \implies C \]
\[ \iff \]
\[ A \land B \implies C \]
\[ \iff \]
\[ \forall (\text{sched}) \]
\[ \text{domain} (\text{psched}) \subseteq \text{domain} (\text{sched}) \]
\[ \implies \forall (i) \]
\[ i \in \text{domain} (\text{psched}) \]
\[ \implies psched(i).est \leq sched(i).st \leq psched(i).lst \]
\[ \land \forall(k, t_1, t_2, R, y) \]
\[ k \in \text{domain}(pes(psched)) \land \]
\[ t_1 = pes(psched)(k) \land t_2 = pes(psched)(k + 1) \]
\[ \land R \subseteq ACPR \land |R| = T \land \]
\[ \sum_{x \in R} \text{unav}(x, t_1, t_2, y)_{psched} = T \land y \in ACPR / R \]
\[ \land \text{sacle}(t, psched) \land \text{affects}(i, y) \]
\[ \implies sched(i).ft < UB \lor sched(i).st \geq LB \]

Since: \[ A \implies (B \land C) \]
\[ \implies (A \land B) \implies C \]

\[ \implies \]
\[ \forall(sched) \]
\[ \text{domain}(psched) \subseteq \text{domain}(sched) \]
\[ \implies \forall(i) \]
\[ i \in \text{domain}(psched) \land \]
\[ \forall(k, t_1, t_2, j, R, y) \]
\[ k \in \text{domain}(pes(psched)) \land \]
\[ t_1 = pes(psched)(k) \land t_2 = pes(psched)(k + 1) \]
\[ \land R \subseteq ACPR \land |R| = T \land \]
\[ \sum_{x \in R} \text{unav}(x, t_1, t_2, y)_{psched} = T \land y \in ACPR / R \]
\[ \land \text{sacle}(t, psched) \land \text{affects}(j, y) \]
\[ sched(i).st \leq psched(i).lst \]
\[ \implies psched(i).est \leq sched(i).st \]
\[ \implies sched(i).ft < UB \lor sched(i).st \geq LB \]

Treating one disjunct:

\[ \forall(sched) \]
\[ \text{domain}(psched) \subseteq \text{domain}(sched) \]
\[ \implies \forall(i) \]
\[ i \in \text{domain}(psched) \land \]
\[ \forall(k, t_1, t_2, j, R, y) \]
\[ k \in \text{domain}(pes(psched)) \land \]

\( t_1 = \text{pes}(\text{psched})(k) \land t_2 = \text{pes}(\text{psched})(k + 1) \)
\( \land \ R \subseteq \text{ACPR} \land |R| = T \land \)
\( \sum_{x \in R} \land \text{unav}?(x, t_1, t_2, \text{psched}) \ 1 = T \land y \in \text{ACPR}/R \land \)
\( \land \ \text{sacpl}?(t, \text{psched}) \land \text{affects}?(j, y) \)
\( \text{sched}(i).st \leq \text{psched}(i).lst \)
\( \implies \ \text{psched}(i).est \leq \text{sched}(i).st \)
\( \implies \ \text{LB} \leq \text{sched}(i).st \)

Replacing sched with qsched:
\( \text{domain}(\text{qsched}) = \text{domain}(\text{psched}) \)
\( \forall(a) \ a \in \text{domain}(\text{qsched}) \)
\( \implies \ \text{qsched}(a) = \text{sched}(a) \)
(Assuming that each activity has the same index in psched, qsched, and sched)

Since:
\( (A' \implies A) \land (A \implies B) \)
\( \implies \)
\( (A' \implies B) \)
\( \implies \)
\( \forall(qscheid) \)
\( \text{domain}(\text{psched}) \subseteq \text{domain}(\text{qsched}) \)
\( \implies \)
\( \forall(i) \)
\( i \in \text{domain}(\text{qsched}) \land \)
\( \forall(k, t_1, t_2, j, R, y) \)
\( k \in \text{domain}(\text{pes}(\text{psched})) \land \)
\( t_1 = \text{pes}(\text{psched})(k) \land t_2 = \text{pes}(\text{psched})(k + 1) \land \)
\( \land \ R \subseteq \text{ACPR} \land |R| = T \land \)
\( \sum_{x \in R} \land \text{unav}?(x, t_1, t_2, \text{psched}) \ 1 = T \land y \in \text{ACPR}/R \land \)
\( \land \ \text{sacpl}?(t, \text{psched}) \land \text{affects}?(j, y) \)
\( \text{qsched}(i).st \leq \text{psched}(i).lst \)
\( \implies \ \text{psched}(i).est \leq \text{qsched}(i).st \)
\( \implies \ \text{LB} \leq \text{sched}(i).st \)

Since:
\( \forall(m) \ S(m) \)
\( \implies \)
\( \forall(a) \ a \in \text{domain}(m) \land T(a) \)
\( \implies (p(a) \leq m(a) \implies q(a) \leq m(a)) \)

39
\[ S(p) \land \forall(a) \ a \in \text{domain}(p) \land T(a) \Rightarrow (q(a) \leq p(a)) \]

\[ \forall(i) \\
i \in \text{domain}(psched) \land \\
\forall(k, t_1, t_2, j, R, y) \\
k \in \text{domain}(pes(psched)) \land \\
t_1 = pes(psched)(k) \land t_2 = pes(psched)(k + 1) \\
\land R \subseteq ACRP \land |R| = T \land \\
\sum_{x \in R} \land \text{unav}(x, <t_1, t_2>, psched) 1 = T \land y \in ACRP/R \\
\land \text{sacpl?}(t, psched) \land \text{affects?}(j, y) \\
\Rightarrow LB \leq psched(i).est \]

Treating the second disjunct:

\[ \forall(sched) \\
\text{domain}(psched) \subseteq \text{domain}(sched) \Rightarrow \forall(i) \\
i \in \text{domain}(psched) \land \\
\forall(k, t_1, t_2, j, R, y) \\
k \in \text{domain}(pes(psched)) \land \\
t_1 = pes(psched)(k) \land t_2 = pes(psched)(k + 1) \\
\land R \subseteq ACRP \land |R| = T \land \\
\sum_{x \in R} \land \text{unav}(x, <t_1, t_2>, psched) 1 = T \land y \in ACRP/R \\
\land \text{sacpl?}(t, psched) \land \text{affects?}(j, y) \\
psched(i).est \leq sched(i).st \\
\Rightarrow sched(i).st \leq psched(i).lst \\
\Rightarrow sched(i).ft < UB \]

Since: \[ sched(i).ft < UB \]
\[ \leftrightarrow \]
\[ sched(i).st < UB - sched(i).duration \]
\[ \forall(sched) \]
\[ \text{domain}(psched) \subseteq \text{domain}(sched) \]
\[ \Rightarrow \forall(i) \]
\[ i \in \text{domain}(psched) \land \]
\[ \forall(k, t_1, t_2, j, R, y) \]
\[ k \in \text{domain}(pes(psched)) \land \]
\[ t_1 = pes(psched)(k) \land t_2 = pes(psched)(k + 1) \land R \subseteq ACPR \land |R| = T \land \]
\[ \sum_{x \in R} \text{unav}(x, t_1, t_2, psched) = 1 \land y \in ACPR/R \land \]
\[ \text{sacpl}(t, psched) \land \text{affects}(j, y) \]
\[ psched(i).est \leq sched(i).st \]
\[ \Rightarrow sched(i).st \leq psched(i).lst \]
\[ \Rightarrow sched(i).st < UB - sched(i).duration \]

Replacing sched with qsched:

\[ \text{domain}(qshed) = \text{domain}(psched) \]
\[ \forall(a) a \in \text{domain}(qshed) \]
\[ \Rightarrow qsched(a) = sched(a) \]

(Assuming that each activity has the same index in psched, qsched, and sched)

Since:
\[ (A' \Rightarrow A) \land (A \Rightarrow B) \]
\[ \Rightarrow (A' \Rightarrow B) \]

\[ \Rightarrow \]
\[ \forall(qshed) \]
\[ \text{domain}(psched) \subseteq \text{domain}(qshed) \]
\[ \Rightarrow \forall(i) \]
\[ i \in \text{domain}(qshed)) \land \]
\[ \forall(k, t_1, t_2, j, R, y) \]
\[ k \in \text{domain}(pes(psched)) \land \]
\[ t_1 = pes(psched)(k) \land t_2 = pes(psched)(k + 1) \land R \subseteq ACPR \land |R| = T \land \]
\[ \sum_{x \in R} \text{unav}(x, t_1, t_2, psched) = 1 \land y \in ACPR/R \land \]
\[ \text{sacpl}(t, psched) \land \text{affects}(j, y) \]
\[ psched(i).est \leq qshed(i).st \]
\[ \Rightarrow qshed(i).st \leq psched(i).lst \]
\[ \Rightarrow qshed(i).st < UB - qshed(i).duration \]
Since: \( \forall(m) S(m) \)
\[ \implies \forall(a) a \in \text{domain}(m) \land T(a) \]
\[ \implies (m(a) \leq p(a) \iff m(a) \leq q(a)) \]
\[ \iff \]
\[ S(p) \land \forall(a) a \in \text{domain}(p) \land T(a) \]
\[ \implies (p(a) \leq q(a)) \]
\[ \iff \]
\[ \forall(i) \]
\[ i \in \text{domain}(psched) \land \]
\[ \forall(k, t_1, t_2, j, R, y) \]
\[ k \in \text{domain}(pes(psched)) \land \]
\[ t_1 = pes(psched)(k) \land t_2 = pes(psched)(k + 1) \]
\[ \land R \subseteq ACPR \land |R| = T \land \]
\[ \sum_{x \in R} \text{unav}?(x, \langle t_1, t_2, psched \rangle) \]
\[ = T \land y \in ACPR/R \]
\[ \land \text{sacpl}?(t, psched) \land \text{affects}?(j, y) \]
\[ \implies psched(i).lst < UB - psched(i).duration \]

Since: \( psched(i).lst < UB - psched(i).duration \)
\[ \iff \]
\[ psched(i).lft < UB \]
\[ \iff \]
\[ \forall(i) \]
\[ i \in \text{domain}(psched) \land \]
\[ \forall(k, t_1, t_2, j, R, y) \]
\[ k \in \text{domain}(pes(psched)) \land \]
\[ t_1 = pes(psched)(k) \land t_2 = pes(psched)(k + 1) \]
\[ \land R \subseteq ACPR \land |R| = T \land \]
\[ \sum_{x \in R} \text{unav}?(x, \langle t_1, t_2, psched \rangle) \]
\[ = T \land y \in ACPR/R \]
\[ \land \text{sacpl}?(t, psched) \land \text{affects}?(j, y) \]
\[ \implies psched(i).lft < UB \]
Appendix C - Domain Theory for the Outage Problem

!! in-package("RE")
!! in-grammar('THEORY-GRAMMAR, 'REGROUP)

THEORY SAFE-OUTAGE

%---------------------------------------------------------------------
THEORY-IMPORTS {}

%---------------------------------------------------------------------
THEORY-TYPE-PARAMETERS {}

%---------------------------------------------------------------------
THEORY-TYPES

%-------------------- Basic Types --------------------
type time = integer

type quantity = integer

type state-res = symbol

type acploss-i = integer  %(0 - yes ; 1 no)

type list-av-ress = set(state-res)

type set-of-acts-acploss = set(activity)

type state-type
    = tuple(acploss?: acploss-i,
            num-unnv-ress: integer,
            unnv-ress-map: map(symbol, set(symbol)),
            list-av-ress: set(state-res))

type st-hist-map = map(time, state-type)

% dependent on the input data
constant *initial-state-map*: st-hist-map
    = {{0 -> < 1, 0, {}}, {
        'div1, 'div2, 'div3, 'div4, 'su10, 'su20 }}
type pred-succ-act
   = tuple(id: symbol, lag: integer, tie: symbol)

%---------------------------------- Activity ----------------------------------
type activity = tuple(act-name : symbol,
                      predecessors : seq(pred-succ-act),
                      duration : integer,
                      est : integer,
                      lst : integer,
                      st : integer,
                      ft : integer,
                      effect-set: set(symbol)
 )

% --------------------------------- Schedule -----------------------------------

type sched = seq(activity)

%---------------------------------- State Related Types and Constants ------------

constant *acp-sources* : map( symbol , seq(symbol))
   = { | 'on-site -> [ 'div1, 'div2, 'div3, 'div4],
            'off-site -> [ 'su10, 'su20 ],
            'control-variable -> [ 'acploss ] }

%----------------------------------------- THEORY-OPERATIONS ---------------------

% resources are assumed to be unlimited

%--------------------------------- Top Level Functions -----------------------------

function pos-safe-outage
   (def-sched: seq(activity)
    | size(def-sched) > 0
    & Consistent-Activity-Separation-EST(def-sched)
    & Consistent-Activity-Separation-LST(def-sched)
    & Consistent-ac-power-prop1(def-sched,
      construct-state-map-def(def-sched) )
    & Consistent-ac-power-prop2(def-sched,
      construct-state-map-def(def-sched) ))
returns (schedule: seq(activity)
| consistent-ac-power(schedule)
& All-activities-scheduled(def-sched, schedule))‰ completeness

function safe-outage-windows
(activities: seq(activity)| size(activities) > 0)
returns (schedule: seq(activity) |
Consistent-Activity-Separation-EST(schedule) &
Consistent-Activity-Separation-LST(schedule) &
Consistent-ac-power(schedule)&
All-activities-scheduled(activities, schedule))‰ completeness

Consistent Activity-Separation

function Consistent-Activity-Separation-EST
(schedule : seq(activity))
: boolean = fa (i : integer, j :integer, act : activity)
(i in [1 .. size(schedule)]
& j in [1 .. size(schedule(i).predecessors)]
& act = get-activity(schedule, schedule(i).predecessors(j))
& defined?(act)
=> (act.est + act.duration + schedule(i).predecessors(j).lag)
<= schedule(i).est)

function Consistent-Activity-Separation-LST
(schedule : seq(activity))
: boolean = fa (i : integer, j :integer, w: integer)
(i in [1 .. size(schedule)] &
j in [1 .. size(schedule(i).predecessors)]) &
w = get-activity-index(schedule, schedule(i).predecessors(j).1)
=> (schedule(i).lst - schedule(w).duration >= schedule(w).lst ))

Consistent Ac-Power

function Consistent-ac-power
( schedule: sched) : boolean
= let (state-hist-map: st-hist-map
= construct-state-map(schedule))
fa (t1:integer)
  (t1 in domain(state-hist-map)
    => state-hist-map(t1).num-unav-ress
    <= (1 + state-hist-map(t1).acploss))

State-of-Plant

function Construct-state-map(schedule: sched)
  : st-hist-map
  = construct-state-map-aux(schedule, *initial-state-map*)

function Construct-state-map-aux(schedule: sched, init-st-hist: st-hist-map)
  : st-hist-map
  = if empty(schedule)
    then init-st-hist
  else if empty(first(schedule).effect-set)
    then construct-state-map-aux(rest(schedule), init-st-hist)
  else construct-state-map-aux(rest(schedule),
      init-st-hist ** add-act-map(first(schedule),
      first(schedule).effect-set,
      first(schedule).st,
      first(schedule).ft, init-st-hist))

function add-act-map
  (act: activity,
    list-effects: set(symbol),
    t1: time, t2: time,
    state-hist-map: st-hist-map)
  : st-hist-map
  = if t1 >= t2
    then {||}
    else (let (start-event-map: st-hist-map
               = add-event-map-seq(act, list-effects,
               t1, t2, state-hist-map))
               start-event-map
               ** add-event-map(act, list-effects, t2, 'f,
               state-hist-map ** start-event-map))

function fd-def-app
  (act: activity, ps: seq(activity),
    list-effects: set(symbol),
t1: time, t2: time,
state-hist-map: st-hist-map
: st-hist-map
  = if t1 >= t2
    then {||}
    else *call-def* <- append(*call-def*, <act, t1, t2, 'app>);
      (let (start-event-map: st-hist-map
             = add-event-map-seq(act,list-effects,
                t1, t2, state-hist-map))
        start-event-map
        ++ add-event-map(act,list-effects, t2, 'f,
        state-hist-map ++ start-event-map))

function fd-def-ext-est
  (ind: integer, ps: seq(activity),
   list-effects: set(symbol),
   t1: time, t2: time,
   state-hist-map: st-hist-map)
  : st-hist-map
  = let (begin-time: integer= max(t1,ps(ind).lst))
      (if ps(ind).lst >= t2
        then {||}
        else *call-def* <- append(*call-def*, <ps(ind), t1, t2, 'est>);
          (let (start-event-map: st-hist-map
                 = add-event-map-seq(ps(ind),list-effects,
                    begin-time, t2, state-hist-map))
              start-event-map
              ++ add-event-map(ps(ind),list-effects, t2, 'f,
              state-hist-map ++ start-event-map)))

function fd-pos-ext-est
  (ind: integer, ps: seq(activity),
   list-effects: set(symbol),
   t1: time, t2: time,
   state-hist-map: st-hist-map)
  : st-hist-map % returns all the entries that changed
  = if t1 >= t2
    then {||}
    else (let (start-event-map: st-hist-map
                = del-event-map-seq(ps(ind),list-effects, t1, t2, state-hist-map))
                  (start-event-map ++ add-event-map(ps(ind),list-effects, t2, 's,
                  state-hist-map ++ start-event-map)))
function fd-pos-ext-lst
(ind: integer, ps: seq(activity),
list-effects: set(symbol),
t1: time, t2: time,
state-hist-map: st-hist-map)
: st-hist-map % returns all the entries that changed
= if t1 >= t2
  then {} |{}
  else (let (start-event-map: st-hist-map
  = del-event-map-seq(ps(ind),list-effects, t1, t2, state-hist-map))
  start-event-map ++ add-event-map(ps(ind),list-effects, t2, 'f,
  state-hist-map ++ start-event-map))

function fd-def-ext-lst
(ind: integer, ps: seq(activity),
list-effects: set(symbol),
t1: time, t2: time,
state-hist-map: st-hist-map)
: st-hist-map
  = let (end-time: integer= min((ps(ind).est + ps(ind).duration), t2))
    (if t1 >= (ps(ind).est + ps(ind).duration)
      then {} |{}
      else (let (start-event-map: st-hist-map =
        add-event-map-seq(ps(ind),list-effects,
        t1, end-time, state-hist-map))
        (if end-time = t2 & t2 =~ ps(ind).est + ps(ind).duration
          then start-event-map
          else (start-event-map ++ add-event-map(ps(ind),
            list-effects, end-time, 'f,
            state-hist-map ++ start-event-map)))))

function add-event-map-seq
(act: activity,
list-effects: set(symbol),
t1: time, t2: time,
state-hist-map: st-hist-map
: st-hist-map
= let (seq-time: seq(integer)=
   sort([x | (x) x in domain(state-hist-map)
       & x > t1 &
       x < t2], lambda(a1, a2) a1 <= a2),
   event-before: state-type =
   get-event-before(t1, state-hist-map))
   let (update-first: st-hist-map =
   [{| t1 -> update-status-start(t1, act.act-name,
       event-before,list-effects)|}])
   add-event-map-rec(act, list-effects, seq-time,
     state-hist-map++ update-first, update-first)

function add-event-map-rec
   (act: activity,
    list-effects: set(symbol),
    seq-time: seq(integer),
    state-hist-map: st-hist-map,
    init-map: st-hist-map)
   = if empty(seq-time)
       then init-map
       else let (add-first: st-hist-map = add-event-map(act, list-effects,
             seq-time(1), 's ,
             state-hist-map))
             add-event-map-rec(act, list-effects, rest(seq-time),
             state-hist-map ++ add-first,
             init-map ++ add-first)

%%
function add-event-map
   (act: activity,
    list-effects: set(symbol),
    t1: time, type-s: symbol,
    state-hist-map: st-hist-map)
   : st-hist-map
   = (let (event-before: state-type
            = get-event-before(t1, state-hist-map))
       if type-s = 's
       then {|
       t1 ->
update-status-start(t1, act.act-name, event-before, list-effects)]]
else { t1 ->
update-status-finish(t1, act.act-name, event-before, list-effects)]}

% this should be del-partial-act-map

function del-act-map
(act: activity,
 list-effects: set(symbol),
 t1: time, t2: time,
 state-hist-map: st-hist-map)
: st-hist-map % returns all the entries that changed
= if t1 >= t2
then {}
else (let (start-event-map: st-hist-map
= del-event-map-seq(act, list-effects, t1, t2, state-hist-map))
if t2 = act.lst + act.duration
then (start-event-map ** add-event-map(act, list-effects, t2, 'f,
state-hist-map ** start-event-map))
else if t1 = ( act.est + act.duration)
then (start-event-map ** add-event-map(act, list-effects, t2, 's,
state-hist-map ** start-event-map))
else start-event-map )

function del-event-map-seq
(act: activity,
 list-effects: set(symbol),
 t1: time, t2: time,
 state-hist-map: st-hist-map)
: st-hist-map
= let (seq-time: seq(integer)=
    sort([ x | (x) x in domain(state-hist-map)
    & x > t1 &
x < t2], lambda( a1, a2) a1 <= a2),
event-before: state-type = get-event-before(t1, state-hist-map))
let (update-first: st-hist-map=
{ t1 ->
update-status-finish(t1, act.act-name, event-before, list-effects)]})
del-event-map-rec(act, list-effects, seq-time,
state-hist-map ** update-first, update-first)
function t-find-event-before
  (ti: integer, m1: map(integer, set(integer)))
  : set(integer)
  = m1(t-time-before-in-domain(m1, ti, m1(ti)))

function t-time-before-in-domain
  (m1: map(integer, set(integer)), i-ti: integer, i-ti-val: set(integer))
  : integer =
  let (val = undefined)
  ti-val = m1(ti)
      & ti < i-ti
    & (defined?(val) => ti > val)
   --> val <- (val; ti);
  val

function del-event-map-rec
  (act: activity,
   list-effects: set(symbol),
   seq-time: seq(integer),
   state-hist-map: st-hist-map,
   init-map: st-hist-map)
  = if empty(seq-time)
    then init-map
  else let (add-first : st-hist-map= add-event-map(act, list-effects,
    seq-time(1), 'f ',
    state-hist-map))
     del-event-map-rec(act, list-effects, rest(seq-time),
    state-hist-map ++ add-first,
    init-map ++ add-first)

function get-event-before
  (ti: time, state-hist-map: st-hist-map)
  : state-type
  = if defined?(state-hist-map(ti))
    then state-hist-map(ti)
  else find-event-before(ti, state-hist-map)

function find-event-before
  (ti: time, state-hist-map: st-hist-map)
  : state-type
  = state-hist-map(time-before-in-domain(state-hist-map,
t1, state-hist-map(t1))

function value-in-interval-map
  (size-def-reserv-m: map(time, alpha), i-ti: time)
  : alpha =
  let (var val = 0, var prev-dom-el = undefined)
  ti-val = size-def-reserv-m(ti)
  & ti <= i-ti
  & (defined?(prev-dom-el) => ti > prev-dom-el)
  -->(val <- (val; ti-val)); % This is a hack to avoid an early stop
  prev-dom-el <- (prev-dom-el; ti));
  val

function load-map-equal?
  (m1: st-hist-map, m2: st-hist-map): boolean =
  fa(ti) (ti in domain(m1) union domain(m2)
  => value-in-interval-map(m1, ti) = value-in-interval-map(m2, ti))

function time-before-in-domain
  (state-hist-map: map(time, state-type), i-ti: time, i-ti-val: state-type)
  : time =
  let (val = undefined)
  ti-val = state-hist-map(ti)
  & ti < i-ti
  & (defined?(val) => ti > val)
  -->(val <- (val; ti));
  val

function time-after-in-domain
  ( i-ti: time, i-ti-val: quantity,
    def-state-map: map(time, state-type)): time =
  let (val = undefined)
  & ti-val ~ i-ti-val
  & ti > i-ti
  & (defined?(val) => ti < val)
  -->(val <- (val; ti));
  val

function update-status-start(ti: time, a-name: symbol,
  state-e-before: state-type, l-effects: set(symbol))
: state-type
  = let (new-acl-status: integer
    = (if 'acploss in l-effects
      then 0
    else state-e-before.acploss?),
      new-num-used-res: integer
    = state-e-before.num-unav-ress
    + size(intersect(l-effects, state-e-before.list-av-ress)),
      new-unav-ress-map: map(symbol, set(symbol))
    = { | x ->
        update-res-map(x, state-e-before.unav-ress-map, a-name, l-effects)
      | (x) x in union(domain(state-e-before.unav-ress-map),
        l-effects) }},
    new-av-list: set(symbol)
  = setdiff(state-e-before.list-av-ress, l-effects))
< new-acl-status,
  new-num-used-res,
  new-unav-ress-map,
  new-av-list >

function update-status-finish
  (t1: time, a-name: symbol, state-e-before: state-type,
   l-effects: set(symbol))
  : state-type
  = let (new-acl-status: integer
    = if 'acploss "in l-effects
      then state-e-before.acploss?
    else (let (acts = state-e-before.unav-ress-map('acploss))
      if defined?(acts)
        & acts-less-act-named(acts, a-name) "= {})
      then 0
    else 1),
      new-num-used-res: integer
    = size({ x | (x: symbol, acts)
        acts = state-e-before.unav-ress-map(x)
      & defined?(acts)
      & x "= 'acploss
      & acts-less-act-named(acts, a-name) "= {}}),
      new-unav-ress-map: map(symbol, set(symbol))
    = { | x -> new-acts
        | (x, acts, new-acts) acts = state-e-before.unav-ress-map(x)
      & defined?(acts)
      & new-acts = acts-less-act-named(acts, a-name)
      & new-acts "= {} })}
    let (new-av-list: set(symbol)
= setdiff(*initial-state-map*(0).list-av-ress,
   domain(new-unav-res-map))
< new-acp-status,
  new-num-used-res,
  new-unav-res-map,
  new-av-list >

function update-res-map
  (res: symbol, res-map: map(symbol, set(symbol)), a-name: symbol,
   l-effects: set(symbol)
   : set(symbol)
   = if res in l-effects
      then if defined?(res-map(res))
      then res-map(res) with a-name
      else {a-name}
      else if defined?(res-map(res))
      then res-map(res)
      else {}
first(schedule).effect-set,
first(schedule).lst,
first(schedule).est
+ first(schedule).duration,
init-st-hist))

function Construct-state-map-pos(schedule: sched)
  : st-hist-map
  = construct-state-map-pos-aux(schedule, *initial-state-map*)

function Construct-state-map-pos-aux
  (schedule: sched, init-st-hist: st-hist-map)
  : st-hist-map
  = if empty(schedule)
    then init-st-hist
    else if empty(first(schedule).effect-set)
      then construct-state-map-pos-aux(rest(schedule), init-st-hist)
      else construct-state-map-pos-aux(rest(schedule), init-st-hist
++ add-act-map(first(schedule),
  first(schedule).effect-set,
  first(schedule).est,
  first(schedule).lst
+ first(schedule).duration, init-st-hist))

function longest-highest-poss-excess-interval
  (poss-state-map: map(time, state-type))
  : tuple(time, time) =
  let (var best-dom-val = undefined,
    var best-ran-val = undefined,
    var best-after-dom-val = undefined)
& ti-val > 1
& (defined?(best-dom-val)
=> ti-val > best-ran-val
or (ti-val = best-ran-val
  & time-after-in-domain(ti, ti-val, poss-state-map) - ti
> best-after-dom-val - best-dom-val))
  -- (best-dom-val <- (best-dom-val; ti);
  best-ran-val <- (best-ran-val; ti-val);
  best-after-dom-val
  <- time-after-in-domain
  ( best-dom-val, best-ran-val, poss-state-map));
if defined?(best-dom-val) & defined?(best-after-dom-val)
then <best-dom-val, best-after-dom-val - 1>
else undefined

-----------------------------

function maximally-poss-act-in-poss-interval
    (sched: seq(activity), i-ti: time, e-ti: time,
     unav-res: map(symbol, set(symbol)))
    : integer =
let (var max-poss-act = undefined,
    var max-poss-time = undefined,
    var max-poss-indx = undefined,
    sel-resource = get-least-committed-res(unav-res))
(enumerate act over unav-res(sel-resource) do
    let ( ti: integer = get-activity-index(sched, act))
    let (act-poss-time = poss-time-in-interval(sched(ti), i-ti, e-ti))
    if (defined?(max-poss-time) => act-poss-time > max-poss-time)
        then max-poss-act <- act;
            max-poss-indx <- ti;
            max-poss-time <- act-poss-time);
% format(true, "~% selected resource :~S~% selected activity :~S",
% sel-resource, max-poss-act);
max-poss-indx

function get-least-committed-res(unav-res : map(symbol, set(symbol)))
    : symbol
    = first(sort([ x | (x) x in domain(unav-res)],
    lambda(a1: symbol, a2: symbol)
    (size(unav-res(a1)) < size(unav-res(a2)))))

function poss-time-in-interval
    (act: activity, i-ti: time, e-ti: time): time =
let (est = act.est,
    lst = act.lst,
    dur = act.duration)
let (eft = est + dur,
    lft = lst + dur)
if est = lst then 0
else max(min(lft, e-ti + 1) - max(eft, i-ti),
    min(lst, e-ti) + 1 - max(est, i-ti))
function poss-interval-split-time
  (act: activity, i-ti: time, e-ti: time): time =
let (est = act.est,
    lst = act.lst,
    dur = act.duration)
let (eft = est + dur,
    lft = lst + dur)
let (max-est = max(est, i-ti),
    min-lst = min(lst, e-ti),
    max-eft = max(eft - 1, i-ti),
    min-lft = min(lft - 1, e-ti))
if est = lst then undefined
else
  if min-lst - max-est >= min-lft - max-eft
     then (max-est + min-lst) div 2
     - (if max-est = min-lst & est < max-est
         then 1 else 0)
  else (max-eft + min-lft) div 2 - dur
       + (if max-eft = min-lft & eft <= max-eft
           then 0 else 1)

function All-activities-scheduled
  (activities: seq(activity),
   schedule : seq(activity))
  : boolean
  = included-activities(activities) = included-activities(schedule)

function Included-activities(activities : seq(activity))
  : set(symbol)
  = seq-to-set(image(lambda (act : activity)
                        act.act-name,
                        activities))

function sort-activities (activities : seq(activity)) : seq(activity)
= image(lambda(x : tuple(activity, seq(activity))) x.1,
         sort(image(lambda(y) <y, activities>, activities),
              sort-criteria))

function get-level(act : activity, acts : seq(activity)) : integer
  computed-using
empty(act.predecessors) => get-level(act, acts) = 0,
true => get-level(act, acts) = 1 + reduce(lambda(x,y) if y < x then x else y,
image(lambda(y) get-level(y, acts),
image(lambda(x) get-activity(acts, x),
act.predecessors)))

function sort-criteria (act1 : tuple(activity, seq(activity)),
act2 : tuple(activity, seq(activity))) : boolean
= let (key1-1 : integer = get-level(act1.1, act1.2),
key1-2 : integer = get-level(act2.1, act2.2),
key2-1 : integer = act1.1.1st - act1.1.est - act1.1.duration,
key2-2 : integer = act2.1.1st - act2.1.est - act2.1.duration)
if (key1-1 < key1-2) then true
else (key1-1 = key1-2) & (key2-1 < key2-2)

function get-activity
(activities : seq(activity), act-s : pred-succ-act)
: activity
= first(filter(lambda(x: activity) x.act-name = act-s.id, activities))

function get-activity-index
(activities : seq(activity), name : symbol)
: integer
= some (indx0 : integer)
(indx0 in domain(activities) & name = activities(indx0).1)

% Printing functions %

function print-problem-gannt(prob-acts: seq(activity)) =
(enumerate act: activity over prob-acts do
  print-gannt-line(act.act-name, act.est, act.lst, act.duration));
format(true, "~9@a~10@a~10@a~10@a~10@a~10@a~%",
0, 10, 20, 30, 40, 50);
values()

function print-ascii-gannt(sched: seq(activity)) =
let (width = reduce(max,
  image(lambda (act: activity) act.lst + act.duration, sched)))
let (scale-factor = if width < 100 then 1 else width div 100 + 1)
(enumerate act: activity over sched do
  print-gannt-line
  (act.act-name,
   act.est div scale-factor,
   act.lst div scale-factor,
   max(1, (act.duration - 1) div scale-factor + 1)));
format(true, "-9@a", 0);
(enumerate sn over [1 .. width div scale-factor div 10] do
  format(true, "-10@a", sn * 10 * scale-factor));
format(true, "-%");
values()

function print-gannt-line(nm: symbol, est: time, lst: time, dur: time) =
  let (est = est + 1, % because zero-based
       lst = lst + 1)
  let (eft = est + dur - 1,
       lft = lst + dur - 1)
  if est = lft then format(true, "-8a-v@a-%",
                          nm, est, "*")
  elseif lst = est then format(true, "-8a-v@a-v,,,,,*@a-%",
                               nm, est, ",", lft, ",", est, ")"
  elseif lst < eft then format(true, "-8a-v@a-v,,,,,-@a-v,,,,,*@a-%",
                                nm, est, ",", lst, ",", est, ",", eft, ",", lft, ",", est, ")"
  elseif lst = eft then format(true, "-8a-v@a-v,,,,,-@a-v,,,,,-@a-%",
                               nm, est, ",", lst, ",", est, ",", est, ",", lft, ",", eft, ",", est, ")"
  else format(true, "-8a-v@a-v,,,,,-@a-v,,,,,-@a-v,,,,,-@a-%",
              nm, est, ",", eft, ",", est, ",", lst, ",", est, ",", lst, ",", est, ")"

function print-ascii-hist
  (hist: map(time, state-type), sched: seq(activity), scale?: boolean) =
  let (height = reduce(max, image(lambda (st: state-type)
                          st.num-unav-ress - st.acploss?,
                          range(hist))),
       width = reduce(max,
                      image(lambda (act: activity) act.lst + act.duration,
                             sched)))
  let (scale-factor = if width < 100 then 1 else width div 100 + 1)
  (enumerate j over [0 .. height] do
let (i = height - j,
var this-h = 0,
var ign-ctr = 0)
    format(true, "-70a ", i);
(enumerate k over [0 .. width] do
    let (st: state-type = hist(k))
    this-h <- (if defined?(st)
        then st.num-unav-ress - st.acploss?
    else this-h);
ign-ctr <- ign-ctr + 1;
    if ign-ctr = scale-factor
    then prin(n(if this-h = i then "-" else " ");
ign-ctr <- 0);
    format(true, "-%");
(if scale? then
    format(true, "-90a", 0);
(enumerate sn over [1 .. width div scale-factor div 10] do
    format(true, "-100a", sn * 10 * scale-factor));
    format(true, "-%");
values()

function print-ascii-gannt-and-poss-hist(sched: seq(activity)) =
    print-ascii-gannt(sched);
    print-ascii-hist(construct-state-map-pos(sched), sched, false)

%% Gs Operators & functions %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% p-sched refines to qsched ; p-state-hist refines to q-state-hist
% p-state-evs refines to q-state-evs
% drs 20ct95: reversed the order of lst inequality

function REFINES-TO
    (p-sched : sched, q-sched : sched) : boolean
= ( size(p-sched) <= size(q-sched)
    & fa(i : integer)
        ( i in [1 .. size(p-sched)]
        => (p-sched(i).est <= q-sched(i).est &
            q-sched(i).lst <= p-sched(i).lst
        ))
)

function SEQ-SATISFIES
    (p-sched : seq(activity), sched : seq(activity)) :boolean
= refines-to(p-sched, sched)
function Consistent-ac-power-filter( p-schedule: sched,
def-state-map: st-hist-map)
  : boolean
  = fa (t1:integer, act: integer)
    (t1 in domain(def-state-map) & act in domain(p-schedule) &
      ~empty(p-schedule(act).effect-set)
      & ~empty(intersect(p-schedule(act).effect-set,
                     def-state-map(t1).list-av-ress))
    )
  => def-state-map(t1).num-unav-ress <=
      (1 + def-state-map(t1).acploss?)

function Consistent-ac-power-prop1( p-schedule: sched,
def-state-map: st-hist-map)
  : boolean
  = fa (t1:integer, t2:integer, act: integer)
    (t1 in domain(def-state-map) & act in domain(p-schedule)
      & ~empty(p-schedule(act).effect-set)
      & t2 = find-time-after(t1, def-state-map(t1), def-state-map)
      & defined?(t2) &
    )
  =>
    def-state-map(t1).num-unav-ress = (1 + def-state-map(t1).acploss?)&
      ~empty(intersect(p-schedule(act).effect-set,
                     def-state-map(t1).list-av-ress))
  & (p-schedule(act).est > (t1 - p-schedule(act).duration))

function Consistent-ac-power-prop2( p-schedule: sched,
def-state-map: st-hist-map)
  : boolean
  = fa (t1:integer, t2:integer, act: integer)
    (t1 in domain(def-state-map) & act in domain(p-schedule)
      & ~empty(p-schedule(act).effect-set) &
    )
  =>
    t2 = find-time-after(t1, def-state-map(t1), def-state-map)
      & defined?(t2) &
def-state-map(t1).num-unav-ress = (1 + def-state-map(t1).acploss) &
  ~empty(intersect(p-schedule(act).effect-set,
           def-state-map(t1).list-av-ress))
& t2 > p-schedule(act).lst
   => (p-schedule(act).lst
       <= t1 - p-schedule(act).duration)

function find-time-after(t1: time, t1-val: state-type,
                        def-state-hist-map: st-hist-map)
  : time
  = let (val = undefined)
     t1-val = def-state-hist-map(t1)
     & t1 > t1
     & (defined?(val) => ti < val)
     --> val <- (val; ti);
  val

%-------------------- Extract -------------------------------

function new-activity
  (act-i : activity) : activity
  = act-i.ft <- act-i.est + act-i.duration;
  act-i.st <- act-i.est;
  act-i

function Extract-schedule
  (ps : seq(activity)) : sched
  = [ new-activity(ps(i))
     | (i:integer)
     i in domain(ps)]

function excess-poss?
  (int : tuple(time, time)) : boolean
  = defined?(int)

%-----------------------------------------------------------------------------------
assert DISTRIBUT-const-def-STATE-map-OVER-EMPTY
  fa ()
    (construct-state-map-def([]) = *initial-state-map*)

assert DISTRIBUT-const-def-STATE-map-OVER-APPEND
  fa (ps: sched, qs:sched, act: activity)
    ( construct-state-map-def(append(ps, act))
      = construct-state-map-def(ps) ++
      fd-def-app(act, ps, act.effect-set, act.lst, act.est + act.duration,
      construct-state-map-def(ps)))

assert DISTRIBUT-CONST-def-STATE-map-OVER-EXTEND-EST
  fa (ps: sched, qs:sched, i:integer, n-est: time)
    ( construct-state-map-def(seq-shadow1(ps, i, tuple-shadow(ps(i).est, n-est)))
      = construct-state-map-def(ps)
      ++ fd-def-ext-est(i, ps, ps(i).effect-set, ps(i).est + ps(i).duration, n-est + ps(i).duration, construct-state-map-def(ps)))

assert DISTRIBUT-CONST-def-STATE-map-OVER-EXTEND-1st
  fa (ps: sched, qs:sched, i:integer, n-1st: time)
    (construct-state-map-def(seq-shadow1(ps, i, tuple-shadow(ps(i).1st, n-1st)))
      = construct-state-map-def(ps) ++
      fd-def-ext-1st(i, ps, ps(i).effect-set, n-1st, ps(i).1st, construct-state-map-def(ps)))

%%%%% POSSIBLE
assert DISTRIBUT-CONST-pos-STATE-map-OVER-EXTEND-EST
  fa (ps: sched, qs:sched, i:integer, n-est)
    (construct-state-map-pos
      (seq-shadow1(ps, i, tuple-shadow(ps(i).est,n-est)) )
      = construct-state-map-pos(ps)
++ fd-pos-ext-est(i, ps, 
ps(i).effect-set, 
ps(i).est, 
n-est, 
construct-state-map-pos(ps))

assert DISTRIBUTE-CONST-pos-STATE-map-OVER-EXTEND-LST
fa (ps: sched, qs:sched, i:integer, n-lst)
  (construct-state-map-pos
   (seq-shadowl(ps, i, tuple-shadow(ps(i).lst, n-lst)))
   = construct-state-map-pos(ps)
   ++ fd-pos-ext-lst(i, ps, ps(i).effect-set, 
n-lst + ps(i).duration, 
ps(i).lst + ps(i).duration, 
construct-state-map-pos(ps)))

assert DISTRIBUTE-ALL-ACTIVITIES-SCHEDULED
fa (Acts: seq(activity), Sched: seq(activity))
  (ALL-ACTIVITIES-SCHEDULED(Acts, Sched)
   = (Included-activities(Acts) = Included-activities(Sched)))

assert DISTRIBUTE-ALL-ACTIVITIES-SCHEDULED-OVER-EMPTY-SEQ
fa (A: seq(activity))
  (all-activities-scheduled(A,[]) = false)

assert DISTRIBUTE-INCLUDED-ACTIVITIES-OVER-EMPTY-SEQ
fa ()
  (Included-activities([]) = {})

assert DISTRIBUTE-INCLUDED-ACTIVITIES-OVER-CONCATENATE
fa (S1: seq(activity), S2: seq(activity))
  (Included-activities(S1 union S2) = Included-activities(S1) union
   Included-activities(S2))

assert DISTRIBUTE-INCLUDED-ACTIVITIES-OVER-APPEND
fa (S: seq(activity), A: activity)
  (Included-activities(append(S,A)) = Included-activities(S) union
   Included-activities([A]))
assert
CONSISTENT-ACTIVITY-SEPARATION-EST-to-CONSISTENT-ACTIVITY-SEPARATION-EST
fa(PS)( fa(S)(REFINES-TO(S, PS) => CONSISTENT-ACTIVITY-SEPARATION-EST(S))
=> CONSISTENT-ACTIVITY-SEPARATION-EST(PS))

assert
CONSISTENT-ACTIVITY-SEPARATION-LST-to-CONSISTENT-ACTIVITY-SEPARATION-LST
fa(PS)( fa(S)(REFINES-TO(S, PS) => CONSISTENT-ACTIVITY-SEPARATION-LST(S))
=> CONSISTENT-ACTIVITY-SEPARATION-LST(PS))

assert get-activity-over-append-predecessors
fa(q,e,j) get-activity(append(q, e), e.predecessors(j))
  = get-activity(q, e.predecessors(j))

%-----------------------------------------------------------------------
% THEORY-RULES

function RULE-DISTRIBUTE-const-def-STATE-map-OVER-EMPTY
  () rb-compile-simplification-equality
  DISTRIBUTED const-def-STATE-map-OVER-EMPTY

function RULE-DISTRIBUTE-const-def-state-map-OVER-append
  () rb-compile-simplification-equality
  DISTRIBUTED const-def-STATE-map-OVER-append

function RULE-DISTRIBUTE-const-def-STATE-map-OVER-Extend-est
  () rb-compile-simplification-equality
  DISTRIBUTED const-def-STATE-map-OVER-Extend-est

function RULE-DISTRIBUTE-const-def-STATE-map-OVER-Extend-lst
  () rb-compile-simplification-equality
  DISTRIBUTED const-def-STATE-map-OVER-Extend-lst
function RULE-DISTRIBUTE-DISTRIBUTE-CONST-pos-STATE-map-OVER-EXTEND-EST
  () rb-compile-simplification-equality
  DISTRIBUTED-const-pos-state-map-OVER-extend-est

function RULE-DISTRIBUTE-DISTRIBUTE-CONST-pos-STATE-map-OVER-EXTEND-1ST
  () rb-compile-simplification-equality
  DISTRIBUTED-const-pos-state-map-OVER-extend-1st

function RULE-DISTRIBUTE-ALL-ACTIVITIES-SCHEDULED-OVER-EMPTY-SEQ-REWRITE
  () rb-compile-simplification-equality
  DISTRIBUTED-ALL-ACTIVITIES-SCHEDULED-OVER-EMPTY-SEQ

function RULE-DISTRIBUTE-INCLUDED-ACTIVITIES-OVER-EMPTY-SEQ-REWRITE
  () rb-compile-simplification-equality
  DISTRIBUTED-INCLUDED-ACTIVITIES-OVER-EMPTY-SEQ

function RULE-DISTRIBUTE-INCLUDED-ACTIVITIES-OVER-CONCATENATE-REWRITE
  () rb-compile-simplification-equality
  DISTRIBUTED-INCLUDED-ACTIVITIES-OVER-CONCATENATE

function RULE-DISTRIBUTE-INCLUDED-ACTIVITIES-OVER-APPEND-REWRITE
  () rb-compile-simplification-equality
  DISTRIBUTED-INCLUDED-ACTIVITIES-OVER-APPEND

function rule-get-activity-over-append-predecessors ()
  rb-compile-simplification-equality get-activity-over-append-predecessors

%-----------------------------------------------------------------------
THEORY-MISC-LAWS

%-----------------------------------------------------------------------
function
OUTAGE-RULE-CONSISTENT-ACTIVITY-SEPARATION-EST-TO-CONSISTENT-ACTIVITY
-SEPARATION-EST (a)
computed-using
a = 'fa(S0)(refines-to(@PS0,S0) => CONSISTENT-ACTIVITY-SEPARATION-EST(S0))'
& new-a = make-structure(
  '##r RB-GRAMMAR
    (rule-instance-make UNDEFINED,
      CONSISTENT-ACTIVITY-SEPARATION-EST(@c-t(PS0))),
      ${}, 
      + 1, OUTAGE-RULE-CONSISTENT-ACTIVITY-SEPARATION-EST-TO-
      CONSISTENT-ACTIVITY-SEPARATION-EST)
  )
=> OUTAGE-RULE-CONSISTENT-ACTIVITY-SEPARATION-EST-TO-CONSISTENT-
ACTIVITY-SEPARATION-EST(a)
  = new-a

function
OUTAGE-RULE-CONSISTENT-ACTIVITY-SEPARATION-LST-TO-CONSISTENT-ACTIVITY-
SEPARATION-LST (a)
computed-using
a = 'fa(S0)(refines-to(@PS0,S0) => CONSISTENT-ACTIVITY-SEPARATION-LST(S0))'
& new-a = make-structure(
  '##r RB-GRAMMAR
    (rule-instance-make UNDEFINED,
      CONSISTENT-ACTIVITY-SEPARATION-LST(@c-t(PS0))),
      ${}, 
      1, OUTAGE-RULE-CONSISTENT-ACTIVITY-SEPARATION-LST-TO-CONSISTENT-
      ACTIVITY-SEPARATION-LST)
  )
=> OUTAGE-RULE-CONSISTENT-ACTIVITY-SEPARATION-LST-TO-CONSISTENT-ACTIVITY
-SEPARATION-LST(a)
  = new-a

function
OUTAGE-RULE-Consistent-ac-power-to-Consistent-ac-power-prop1 (a)
computed-using
a = 'fa(S0)(refines-to(@PS0,S0) => Consistent-ac-power(S0))'
& new-a = make-structure(
  '##r RB-GRAMMAR
    (rule-instance-make UNDEFINED,
Consistent-ac-power-prop1(\(c-t(PSO)\)),
construct-state-map-def(@c-t(PSO))),
  $\{\},$\{\},
  1, OUTAGE-RULE-Consistent-ac-power-to-Consistent-ac-power-prop1)')
=> OUTAGE-RULE-Consistent-ac-power-to-Consistent-ac-power-prop1(a)
  = new-a

function
OUTAGE-RULE-Consistent-ac-power-to-Consistent-ac-power-filter (a)
computed-using
a = 'fa(SO)(refines-to(@PSO,SO) => Consistent-ac-power(SO))'
& new-a = make-structure(
  '##r RB-GRAMMAR
    (rule-instance-make UNDEFINED,
     Consistent-ac-power-filter(@c-t(PSO)),
    construct-state-map-def(@c-t(PSO))),
    $\{\},$\{\},
    1, OUTAGE-RULE-Consistent-ac-power-to-Consistent-ac-power-filter)')
=> OUTAGE-RULE-Consistent-ac-power-to-Consistent-ac-power-filter(a)
  = new-a

function
OUTAGE-RULE-Consistent-ac-power-to-Consistent-ac-power-prop2 (a)
computed-using
a = 'fa(SO)(refines-to(@PSO,SO) => Consistent-ac-power(SO))'
& new-a = make-structure(
  '##r RB-GRAMMAR
    (rule-instance-make UNDEFINED,
     Consistent-ac-power-prop2(@c-t(PSO)),
    construct-state-map-def(@c-t(PSO))),
    $\{\},$\{\},
    1, OUTAGE-RULE-Consistent-ac-power-to-Consistent-ac-power-prop2)')
=> OUTAGE-RULE-Consistent-ac-power-to-Consistent-ac-power-prop2(a)
  = new-a

%---------------------------------------------------------------------------------------------------------------------
THEORY-MISC-RULES

rule tuple-deref-seq-appl-over-get-activity-index-seq-shadow1-tuple-shadow(a)
also {{ index-on -> 'rb-simplification-rules, 'get-field> |}}
a = 'q0(q(get-activity-index
  (seq-shadow1(q1, qip1, tuple-shadow(q2(qip2),@fi, @q)),
  @k))
  .@fo'
& term-equal?(q0, q1) & term-equal?(q1, q2) & term-equal?(ip1, ip2)
& fi ^= fo
--> a = 'q0(get-activity-index(q1, qk)).fo'

rule tuple-deref-over-get-activity-seq-shadow1-tuple-shadow(a)
  also {l index-on -> '<rb-simplification-rules, 'get-field> |}
a = 'get-activity
  (seq-shadow1(q1, Gip1, tuple-shadow(qq2(ip2).fi, qq)),
   qk)
   .fo'
& term-equal?(q1, q2) & term-equal?(ip1, ip2)
& fi ^= fo
--> a = 'get-activity(q1, qk).fo'

rule defined?-over-get-activity-seq-shadow1-tuple-shadow(a)
  also {l index-on -> '<rb-simplification-rules, 'defined?> |}
a = 'defined?(get-activity
(seq-shadow1(q1, Gip1, tuple-shadow(qq2(ip2).qq, qq)),
  qk))'
& term-equal?(q1, q2) & term-equal?(ip1, ip2)
--> a = 'defined?(get-activity(q1, qk))'

%%% Not certain that this is valid
rule get-activity-over-append-predecessors(a)
  also {l index-on -> '<rb-simplification-rules, 'get-activity> |}
a = 'get-activity(append(qq, Gq1, Gq2.predecessors(qj))'
& term-equal?(qj, q2)
--> a = 'get-activity(qq, Gq2.predecessors(qj))'

%-------------------------------------------------------------
THEORY-MISC-FORMS

form remove-SOME-OP-SIMPLIFICATION-RULES
  remove-rb-simplification-rules('some,
   {'basic-boolean-theory-distribute-some-over-ordered-or-into-ex-form})

form add-SOME-OP-SIMPLIFICATION-RULES
  add-rb-simplification-rules('some,
   {'basic-boolean-theory-rule-distribute-var-definitions-in-some-op})

%%% 'basic-boolean-theory-rule-distribute-var-definitions-in-some-op

form remove-member-SIMPLIFICATION-RULES
  remove-rb-simplification-rules('member,
form remove-defined?-SIMP rule-distribute-in-over-interval)
remove-rb-simplification-rules
(defined?, {regroup-type-rule-definedness-of-function-parameter})

form remove-seq-theory-rule-distribute-in-over-reverse
remove-rb-simplification-rules(member,
{seq-theory-rule-distribute-in-over-reverse})

form ADD-universal-FI-laws
ADD-RB-FORWARD-IMPLICATIONS
(forall,
{outage-rule-consistent-activity-separation-est-to-consistent-activity-separation-est
outage-rule-consistent-activity-separation-lst-to-consistent-activity-separation-lst
outage-rule-consistent-ac-power-to-consistent-ac-power-filter
outage-rule-consistent-ac-power-to-consistent-ac-power-prop1
outage-rule-consistent-ac-power-to-consistent-ac-power-prop2})

form ADD-AND-SIMPLIFICATION-RULES
% associative commutative idempotent identity fixpoint
add-simplification-rules-for-operator('and) %;
% add-rb-simplification-rules('and,
% {basic-boolean-theory-rule-distribute-and-over-or})
Appendix D - Global Search Theories for the Outage Problem

Appendix D contains the global search theory for scheduling considering the definite period of activities.

form index-outage-scheduling
  gs-activities = make-binding('gs-activities)
  & full-sched = make-binding('full-sched)
  & p-sched = make-binding('p-sched)
  & unsched-acts = make-binding('unsched-acts)
  & def-state-map = make-binding('def-state-map)
  & p-sched-new = make-binding('p-sched-new)
  & def-state-map-new = make-binding('def-state-map-new)
  & unsched-acts-new = make-binding('unsched-acts-new)
  & gs-act = make-binding('gs-act)

-->

'##r cypress-grammar
(Global-Search-Theory GS-OUTAGE-SCHEDULING

input-types seq(activity)
input-vars gs-activities
input-condition true

output-types seq(activity)
output-vars full-sched
output-condition
  all-activities-scheduled(gs-activities, full-sched)

subspace-types
  seq(activity),
  seq(activity),
  st-hist-map
subspace-vars
  p-sched,
  unsched-acts,
  def-state-map
subspace-split-vars
  p-sched-new,
  unsched-acts-new,
  def-state-map-new
subspace-vars-constraint
    def-state-map = construct-state-map-def(p-sched)
        & included-activities(gs-activities) = included-activities(p-sched)
        union included-activities(unsched-acts)
        & disjoint(included-activities(p-sched), included-activities(unsched-acts))
        & defined?(p-sched)
Constraint-Info-types
activity
Constraint-Info-vars
    gs-act
Constraint-Info-condition
    gs-act = first(unsched-acts)
Splitting-constraint
    p-sched-new = append(p-sched, gs-act)
    & unsched-acts-new = rest(unsched-acts)
    & def-state-map-new = construct-state-map-def(p-sched-new)
satisfies
    refines-to(p-sched,full-sched)
refines
    refines-to(p-sched, p-sched-new)

initial-space
    (<<[], sort-activities(gs-activities),
*initial-state-map*>)
extract
    full-sched = p-sched
Extractable
    unsched-acts = []
Splittable
    unsched-acts = []
)’ in gs-theories-prop(find-global(’powersequence))

form index-outage-poss-scheduling
    gs-ps = make-binding(’gs-ps)
    & full-sched = make-binding(’full-sched)
    & p-sched = make-binding(’p-sched)
    & def-state-map = make-binding(’def-state-map)
    & poss-state-map = make-binding(’poss-state-map)
    & p-sched-new = make-binding(’p-sched-new)
    & def-state-map-new = make-binding(’def-state-map-new)
    & poss-state-map-new = make-binding(’poss-state-map-new)
    & gs-high-poss-interval = make-binding(’gs-high-poss-interval)
    & gs-res-index = make-binding(’gs-res-index)
    & gs-split-time = make-binding(’gs-split-time)
    & gs-switch = make-binding(’gs-switch)
Global search theory for time window refinement that takes into consideration the potential period of activities.

```
'##r cypress-grammar
(Global-Search-Theory GS-OUTAGE-poss-SCHEDULING-1

input-types seq(activity)
input-vars gs-ps
input-condition true

output-types seq(activity)
output-vars full-sched
output-condition
    all-activities-scheduled(gs-ps, full-sched)

subspace-types
    seq(activity),
    st-hist-map,
    st-hist-map
subspace-vars
    p-sched,
    def-state-map,
    poss-state-map
subspace-split-vars
    p-sched-new,
    def-state-map-new,
    poss-state-map-new
subspace-vars-constraint
    def-state-map = construct-state-map-def(p-sched)
    & poss-state-map = construct-state-map-pos(p-sched)
Constraint-Info-types
    tuple(time, time), integer, time, integer
Constraint-Info-vars
    gs-high-poss-interval,
    % gs-res, computed
    gs-res-index, gs-split-time,
    gs-switch
Constraint-Info-condition
    gs-high-poss-interval
    = longest-highest-poss-excess-interval(poss-state-map)
```
& gs-res-index
  = maximally-poss-act-in-poss-interval
  (p-sched, gs-high-poss-interval.1, gs-high-poss-interval.2,
   poss-state-map(gs-high-poss-interval.1).unav-res-map)

& gs-split-time
  = poss-interval-split-time(p-sched(gs-res-index),
   gs-high-poss-interval.1,
   gs-high-poss-interval.2)
  & defined?(gs-split-time)
  & gs-switch in [0, 1] % 1 causes things to be scheduled early
  & (gs-switch = 0
  & gs-split-time = p-sched(gs-res-index).lst
  or
  gs-switch = 1
  & 1 + gs-split-time = p-sched(gs-res-index).est)

Splitting-constraint
  if gs-switch = 0
  then p-sched-new
    = seq-shadow1(p-sched,
     gs-res-index,
     tuple-shadow(p-sched(gs-res-index).lst,
      gs-split-time))
  & def-state-map-new = construct-state-map-def(p-sched-new)
  & poss-state-map-new = construct-state-map-pos(p-sched-new)
  else
  p-sched-new
    = seq-shadow1(p-sched,
     gs-res-index,
     tuple-shadow(p-sched(gs-res-index).est,
      1 + gs-split-time))
  & def-state-map-new = construct-state-map-def(p-sched-new)
  & poss-state-map-new = construct-state-map-pos(p-sched-new)

satisfies
  refines-to(p-sched,full-sched)
refines
  refines-to(p-sched, p-sched-new)

initial-space
  (&gs-ps, construct-state-map-def(gs-ps),
   construct-state-map-pos(gs-ps))

extract
  full-sched = extract-schedule(p-sched)

Extractable
  ~excess-poss?
    (longest-highest-poss-excess-interval(poss-state-map))

74
Splittable
excess-poss?
   (longest-highest-poss-excess-interval(poss-state-map))
' in gs-theories-prop(find-global('powersquence'))
MISSION
OF
ROME LABORATORY

Mission. The mission of Rome Laboratory is to advance the science and technologies of command, control, communications and intelligence and to transition them into systems to meet customer needs. To achieve this, Rome Lab:

a. Conducts vigorous research, development and test programs in all applicable technologies;

b. Transitions technology to current and future systems to improve operational capability, readiness, and supportability;

c. Provides a full range of technical support to Air Force Materiel Command product centers and other Air Force organizations;

d. Promotes transfer of technology to the private sector;

e. Maintains leading edge technological expertise in the areas of surveillance, communications, command and control, intelligence, reliability science, electro-magnetic technology, photonics, signal processing, and computational science.

The thrust areas of technical competence include: Surveillance, Communications, Command and Control, Intelligence, Signal Processing, Computer Science and Technology, Electromagnetic Technology, Photonics and Reliability Sciences.