Linköping Electronic Articles in
Computer and Information Science
Vol. 2(1997): nr 16

Tackling the Qualification Problem using Fluent Dependency Constraints: Preliminary Report

Patrick Doherty
Department of Computer
and Information Science
Linköping University
S-58183 Linköping, Sweden
patdo@ida.liu.se

Jonas Kvarnström
Department of Computer
and Information Science
Linköping University
S-58183 Linköping, Sweden
jonkv@ida.liu.se

Linköping University Electronic Press
Linköping, Sweden

http://www.ep.liu.se/ea/cis/1997/016/

Published on December 19, 1997 by
Linköping University Electronic Press
581 83 Linköping, Sweden

Linköping Electronic Articles in
Computer and Information Science
ISSN 1401-9841
Series editor: Erik Sandewall

Typeset by the author using LaTeX
Formatted using étendu *style*

Recommended citation:
<Author>. <Title>. Linköping Electronic Articles in
Computer and Information Science, Vol. 2(1997): nr 16.
http://www.ep.liu.se/ea/cis/1997/016/. *December 19, 1997.*

This URL will also contain a link to the author's home page.

Abstract

Recently, a great deal of progress has been made using nonmonotonic temporal logics to formalize reasoning about action and change. In particular, much focus has been placed on the proper representation of non-deterministic actions and the indirect effects of actions. One popular approach to representing the indirect effects of actions has been via the use of *causal* rules which in a more general sense can be viewed as fluent dependency constraints. Although fluent dependency constraints have been used primarily under a loose causal interpretation, we show that when interpreted in a broader sense they provide a flexible means for dealing with a number of other representational problems such as the qualification problem and the ramification constraints as qualification constraints problem, in addition to the standard ramification problem. More importantly, the use of fluent dependency constraints for different purposes does not involve additions to the base nonmonotonic temporal logic, TAL, used here, but simply the addition of several macro operators to an action language used to represent action scenarios or narratives. The payoff is that TAL has already been shown to offer a robust approach to representing action scenarios which permit incomplete specifications of both state and the timing of actions, non-deterministic actions, actions with duration, concurrent actions, use of both boolean and non-boolean fluents, and solutions to the frame and ramification problems for a wide class of action scenarios. In addition, all circumscribed action scenarios in these classes and the more general class involving qualification considered in this paper can be shown to be reducible to the first-order case. Finally, a restricted entailment method for this new class of scenarios is fully implemented. In the paper, we present a challenge example which incorporates all these features, propose a distinction between *weak* and *strong* qualification with a representation of both, and provide a visualization of the preferred entailments using a research tool VITAL for querying and visualizing action scenarios.

At the time of publication of this report, a similar shorter version lacking Appendices 3-6 had been submitted to TIME'98, The Fifth International Workshop on Temporal Representation and Reasoning. This version also contains some minor corrections in addition to comments added in Appendix 1. A longer paper is planned.

TIME'98 submission date: December 8, 1997.

1 Introduction

In this paper, we provide a challenge in the form of a complex action scenario description, the *Russian Airplane Hijack Scenario* (RAH) which requires robust solutions to the frame, ramification and qualification problems. We say robust because the scenario requires the representation of concurrent actions, incomplete specification of states, ramification with chaining, the use of non-boolean fluents, fine-grained dependencies among objects in different fluent value domains, actions with duration, ramification constraints as qualifications to actions, two types of qualification, *weak* and *strong*, and the use of explicit time, in addition to other features. Although some of these features have been reported in previous work, the approach to qualification is new, the use of a research tool VITAL [8], which provides an implementation and visualization of action scenarios is new, and the action scenario, to our knowledge, is one of the more challenging and complex scenarios proposed as a test example. We will use Temporal Action Logic (TAL) to reason about action scenarios.

Temporal Action Logics (TAL) have their origin in the Features and Fluents framework proposed by Sandewall [11], where both a variety of logics of preferential entailment for reasoning about action and change and a framework for assessing the correctness of these and future logics were proposed. One of the definitions of preferential entailment, PMON, was proposed by Sandewall and assessed correct for the $\mathcal{K}$-IA class of action scenarios, a broad class of scenarios which dealt with nondeterministic actions, incomplete specification of state and the timing of actions, and observations at arbitrary states in a scenario. PMON solved the frame problem for the $\mathcal{K}$-IA class. Later, Doherty [1, 2] translated and generalized PMON into an order-sorted first-order logic with a circumscription axiom capturing the PMON definition of preferential entailment.

Recently, a number of additional extensions and generalizations have been added to the original PMON and the logics generated belong to what we call the TAL family. Although the logics belong to the TAL family, each is essentially an incremental addition to the base logic PMON. TAL-RC, proposed by Gustafsson and Doherty [6], provides a solution to the ramification problem for a broad, but as yet unassessed class of action scenarios. The main idea is the addition of a specialization of fluent dependency constraints which we called causal constraints. The solution was based on the insight that the *Occlude* predicate used to solve the frame problem for PMON was all that was needed to define causal rules which turn out to be very similar to action effect axioms. The solution is also extremely fine-grained in the sense that one can easily encode dependencies between individual objects in the domain, work with both boolean and non-boolean fluents and represent both Markovian and non-Markovian dependencies [5]. TAL-C, recently proposed by Gustafsson and Karlsson [7], uses fluent dependency constraints as a basis for representing concurrent actions. A number of phenomena related to action concurrency such as interference between one action's effects and another's execution, bounds on concurrency and conflicting, synergistic, and cumulative effects of concurrent actions are studied.

In this paper, we will consider two problems: the qualification problem and the ramification constraints as qualification constraints problem [4, 9]. We call the logic used, TAL-Q, and note that it is an incremental extension of TAL-C, just as TAL-C is an incremental extension of TAL-RC. In fact, the logical language and minimization policy is roughly the same for TAL-RC, TAL-C, and TAL-Q. The novelty of the solution to these problems is in the combined use of fluent dependency constraints and an additional idea introduced in TAL-C, durational fluents. The advantage of leaving the logic and minimization policy intact is that the new class of action scenarios representable in TAL-Q subsumes previous classes and any circumscribed scenario in TAL-Q is provably and automatically reducible to a compact first-order theory efficiently implemented in a research tool called VITAL [8]. VITAL is an on-line tool which permits both the visualization and querying of action scenarios.

The paper is structured as follows. In Section 2, we briefly describe TAL-Q. We then provide a brief description of the surface language for action scenarios and their translation into an order-sorted first-order language with a circumscription axiom. In Section 3, we describe a complex action scenario, the *Russian Airplane Hijack Scenario* (RAH), and provide a formal description in Appendix 1 which will be used throughout the paper. In Section 4, we consider *weak* and *strong* qualification and provide a preliminary proposal for representing both types in TAL-Q. Appendix 1 contains the RAH action scenario in the language $\mathcal{L}(SD)$ (described below) and Appendix 2-6 contain visualizations of the preferred entailments of the RAH scenario and various extensions to it. The visualizations have been generated automatically from the VITAL tool. Due to page limitations, we have to be very brief with our description of action scenario translation macros and the underlying logic, referring the reader to the following sources ([11, 1, 2, 6, 7, 8]).

2 TAL-Q: Temporal Action Logic with Qualification

In this section, we introduce TAL-Q which will be used as a basis for a preliminary proposal for dealing with qualification of actions. The basic approach we use for reasoning about action and change is as follows. First, represent an action scenario in the surface language $\mathcal{L}(SD)$ which is a high-level language for representing observations, action types and action occurrences, dependency constraints, domain constraints, and timing constraints about actions and their duration.[1] Second, translate $\mathcal{L}(SD)$ into the base language $\mathcal{L}(FL)$ which is an order-sorted first-order language with four predicates $Occlude(t, f)$, $Holds(t, f, v)$, $Per(f)$, and $Dur(f, v)$, where t,f, and v are variables for timepoint, fluent, and value expressions, respectively. *Holds* expresses what value a fluent has at each timepoint. *Occlude* expresses that a fluent at a timepoint is allowed to change value at that timepoint. Each fluent has to be characterized as either a durational fluent, $Dur(f, v)$, with default value v, or a persistent fluent $Per(f)$, but not both. The idea is that unless a durational fluent is occluded at a timepoint, it will retain its default value, while a persistent fluent at $t + 1$ retains whatever value it has at t unless it is occluded.

A linear discrete time structure is used in TAL-Q. The minimization policy is based on the use of filtered preferential entailment [10] where action occurrences (**occ-**) and dependency constraints (**dep-**) are circumscribed with *Occlude* minimized and *Holds* fixed. The result is then filtered with two nochange axioms, the observations, and some foundational axioms such as unique names and temporal structure axioms.

Let $,_{obs}$, $,_{occ}$, $,_{dep}$, and $,_{acc}$ denote the translations into $\mathcal{L}(FL)$ of the observation, occurrence, dependency and domain constraints in an action scenario, respectively. In addition, let $,_{fnd}$ denote the foundational axioms which include axioms for the time structure, unique names and the *Dur*/*Per* specification of fluents. The nochange axioms in $,_{ncg}$ are

$$\forall t, f, v[Per(f) \rightarrow (\neg Occlude(t + 1, f) \rightarrow (Holds(t, f, v) \leftrightarrow Holds(t + 1, f, v)))],$$

which states that persistent fluents that are not occluded at time $t + 1$ retain their value from t, and

$$\forall t, f, v[Dur(f, v) \rightarrow (\neg Occlude(t, f) \rightarrow Holds(t, f, v))],$$

which states that durational fluents have a default value of v, but when occluded can take on arbitrary values. Since each fluent is either durational or persistent, inertia of fluent values or default behavior is dependent on the extension of *Occlude* which is minimized relative to $,_{dep}$ and $,_{occ}$.

If Υ is an action scenario in $\mathcal{L}(SD)$, then $Trans(\Upsilon)$ is its translation into $\mathcal{L}(FL)$ which includes all of the sets of formulas $,_{x}$.

The following definition of preferential entailment applies:

[1] Appendix 1 lists the RAH action scenario in language $\mathcal{L}(SD)$ which is described informally in Section 3.

Definition 1 *The formula α is entailed by $Trans(\Upsilon)$ iff*

$$, _{fnd} \wedge \, , _{acc} \wedge \, , _{obs} \wedge \, , _{ncg} \wedge Circ_{so}(, _{occ} \wedge \, , _{dep}; Occlude) \models \alpha$$

Since there are only positive occurrences of the *Occlude* predicate in the circumscription context, $Circ_{so}(--)$ is reducible to a logically equivalent first-order formula.

The translation from $\mathcal{L}(SD)$ to $\mathcal{L}(FL)$ is straightforward and the reader is referred to [3, 7] for details concerning translation and the logic used. We simply translate one of each type of statement in the scenario in Appendix 1 to provide some understanding, but first we discuss the macro operators C_T, R, I, and X.

The C_T operator stands for *becomes true.* For example, its use in dependency constraint **cc3**, $C_T([t]\ \mathsf{loc}(airplane) \hat{=} loc3)$, would be translated as follows:

$$Holds(t, \mathsf{loc}(airplane), loc3) \wedge \forall u[t = u + 1 \rightarrow \neg Holds(u, \mathsf{loc}(airplane), loc3)]$$

Translations of the next operators are shown after they are discussed. The R operator stands for *fluent reassignment* and where an interval is used, occludes the fluent in the interval and gives it a new value at the last timepoint. For example the action occurrence **occ3** uses the R operator as follows: $R([4]\ \mathsf{loc(dimiter)} \hat{=} \mathsf{office})$. Here, because of the use of a single timepoint, it is only changing at 4.

The I operator stands for *exceptional assignment* and is often used in combination with durational fluents. It states that a fluent will have a particular value which holds throughout the interval or at the timepoint. For example, the dependency constraint **cc1** contains the following:

$$I([t]\ \forall airplane[\neg \mathsf{poss\_board}(person, airplane)]),$$

which states that the default value for the durational fluent poss_board does not apply and the fluent is false at t.

The X operator stands for *occlude assignment.* Its purpose is simply to allow a fluent's value to vary at a timepoint or interval. For example, the dependency constraint **cc2** contains the following:

$$X([t]\ \forall airplane[\neg \mathsf{poss\_board}(person, airplane)]),$$

which states that poss_board may be true or false at t.

The observation statement **obs2**, the action occurrence statement **occ3** which uses the R operator, the dependency statement **cc1** which uses the I operator and the dependency statement **cc2** which uses the X operator are translated into $\mathcal{L}(FL)$ as follows:

obs2 $Holds(0, \mathsf{loc(erik)}, \mathsf{home2}) \wedge Holds(0, \mathsf{loc(comb2)}, \mathsf{home2}) \wedge \neg Holds(0, \mathsf{drunk(erik)}, \mathsf{true})$

occ3 $Holds(2, \mathsf{loc(dimiter)}, \mathsf{home3}) \rightarrow Holds(4, \mathsf{loc(dimiter)}, \mathsf{office}) \wedge Occlude(4, \mathsf{loc(dimiter)})$

cc1 $\forall t, person[Holds(t, \mathsf{inpocket}(person, \mathsf{gun}), \mathsf{true}) \rightarrow$
$\forall airplane[\neg Holds(t, \mathsf{poss\_board}(person, airplane), \mathsf{true}) \wedge$
$Occlude(t, \mathsf{poss\_board}(person, airplane))]]$

cc2 $\forall t, person[Holds(t, \mathsf{drunk}(person), \mathsf{true}) \rightarrow Occlude(t, \mathsf{poss\_board}(person, airplane))]$

3 The Russian Airplane Hijack Scenario

In this section, we will use the methodology of representative examples as a means of considering and proposing a preliminary solution to the qualification and ramification as qualification problems. The proposal, while conveyed via a specific action scenario, can easily be presented in a more generic, but less intuitive manner. We leave that for a longer paper. We will use a new action scenario, the Russian Airplane Hijack scenario, as a representative example.[2] The scenario is described informally below and the formal action scenario can be found in Appendix 1.

A Russian businessman, Vladimir, travels a lot and is concerned about both his hair and safety. Consequently, when traveling, he places both a comb and a gun in his pocket. A Bulgarian businessman, Dimiter, is less concerned about his hair, but when traveling by air, has a tendency to drink large amounts of vodka before boarding a flight to subdue his fear of flying. A Swedish businessman, Erik, travels a lot, likes combing his hair, but is generally law abiding. Now, one ramification of putting an object in your pocket is that it will follow with you as you travel from location to location. Generally, when boarding a plane, the only preconditions are that you are at the gate and you have a ticket. One *possible* qualification to the boarding action is if you arrive at the gate in a sufficiently inebriated condition, as will be the case for Dimiter. A ramification that may in some cases play a dual role as a qualification to the boarding action is if you try to board a plane with a gun in your pocket, which may be the case for Vladimir. Now, Vladimir, Erik and Dimiter, start from home, stop by the office, go to the airport, and try to board flight SAS609 to Stockholm. Both Erik and Vladimir put combs in their pockets at home, Vladimir picks up a gun at the office, while Dimiter is already drunk at home. Who will successfully board the plane? What are their final locations? What is in their pockets after attempting to board the plane and after the plane has arrived at its destination?

If the scenario is encoded properly and our intuitions about the frame, ramification and qualification problems are correct then we should be able to entail the following from the scenario in Appendix 1:

1. Erik will board the plane with comb2 in his pocket and eventually board the plane successfully ending up at his destination.
2. Vladimir will get as far as the airport with a gun and comb1 in his pocket. He will be unable to board the plane.
3. Dimiter will get as far as the airport and may or may not have boarded the plane. He may or may not have comb3 in his pocket. A weaker form of qualification (use of the X operator) is used here. For example, if he is observed to be on the plane then he successfully slipped by security and there is no inconsistency. If he is observed not to be on the plane, then the action failed and there is also no inconsistency.
4. An indirect effect of flying is that the person ends up at the same location as the airplane. In addition, because items in pockets follow the person, a transitive effect results where the items in the pocket are at the same location as the plane. Consequently, erik and comb2 end up at run609b, the final destination of flight sas609.

In fact, we do entail this and more. The facts true in all preferred models for this scenario can be viewed in Appendix 2.[3]

4 Representing the Qualification Problem in TAL-Q

In comparison with the frame and ramification problems, the qualification problem is still one of the least understood and with few satisfactory solutions. This is most probably due to the fact that there are many different types of qualification problem, or reasons for qualifying an action description. The main problem is that in general, it is computationally, ontologically and epistemologically unfeasible to represent complete specifications of all the preconditions to actions which would include all possible qualifications. In this section, we will propose a default-based

[2] This scenario is an elaboration and concretization of a sketch for a scenario proposed by Vladimir Lifschitz in recent on-line discussions in the European Transactions on Artificial Intelligence (ETAI/ENAI).

[3] *red* and *green* stand for false and true values for boolean fluents, *gray* stands for true or false, while *black* after *gray* stands for "do not know the value, but will take the value *gray*" ends up being. "*2*" means 2 possible values. They are not shown in the diagram due to lack of space. The diagram is automatically generated using the VITAL tool. You should have a colored copy of Appendix 2.

solution within the TAL-Q framework that has a number of novelties. We can express both *strong* and *weak* forms of qualification, the representation is efficient, and one can model ramification constraints as qualifications in a number of ways. We consider both forms of qualification using the RAH scenario from Appendix 1 to describe the approach.

Boarding an airplane might have a number of different qualifications, for example, being drunk or carrying a gun in your pocket. In other contexts, these facts may be perfectly natural. The difficulty, especially when such facts are inferred indirectly as ramifications, is in providing a flexible enough representation to allow the facts to play dual roles as either ramifications, or ramifications that may qualify an action. One ramification of traveling or boarding a plane, is that everything in my pocket travels with me, including guns and combs. So if I have a gun in my pocket before boarding a plane, one ramification of the boarding action is that the gun is on the plane. Now, one could argue that having a gun in my pocket plays the role of a qualification to the boarding action and the action should fail. On the other hand, it is perfectly possible that one could slip by with a gun if it is made of a special type of plastic. So, how do we know that an action has been qualified? Well, after the action is executed, we may observe the results of the action's effects. For example, if vladimir is observed to be on the plane even though he has a gun, then he has slipped by and the action was possible after all. If we observe that he isn't on the plane, then possession of the gun actually qualified the action. This weaker form of qualification which we call *weak* qualification can also be represented.

Let's start with *strong* qualification. A qualified action contains a fluent *enabling* the action in its precondition with the same number of arguments as its associated action type. For example, the boarding action contains a fluent $\mathsf{poss\_board}(person, airplane)$:

acs4 $[t_1, t_2]\ \mathsf{board}(person, airplane) \rightarrow [t_1]\ \mathsf{poss\_board}(person, airplane) \wedge \mathsf{loc}(person) \hat{=} \mathsf{airport} \rightarrow R([t_2]\ \mathsf{loc}(person) \hat{=} value(t_2, \mathsf{loc}(airplane)) \wedge \mathsf{onplane}(airplane, person))$

Note that the fluent poss_board is a durational fluent $Dur(\mathsf{poss\_board})$ with default value true which means it can only take on another value (false) at a timepoint if it is occluded at that timepoint for some reason. Such reasons are described as dependency constraints:

cc1 $\forall t[\forall person[[t]\ \mathsf{inpocket}(person, \mathsf{gun}) \rightarrow I([t]\ \forall airplane[\neg \mathsf{poss\_board}(person, airplane)])]]$

This dependency constraint together with the assertion that poss_board is durational implies that as long as a person has a gun in his pocket, poss_board will be false for that person on all airplanes. So far, this is similar to a standard default solution to the qualification problem, not unlike other solutions, but with some subtle differences. For example, it permits more control of the enabling precondition, even allowing it to change during the execution of an action. Strong qualification has its uses, but is not fully adequate for other types of qualification. A visualization of the preferred entailments for the scenario which is listed in Appendix 1, is provided in Appendix 2.[4]

Instead of the above, let's consider the use of a *weak* qualification. Using the current example, the idea here is that `inpocket(person, gun)` should not be an absolute qualification, it should make it less likely that the person can successfully board the plane, but that would depend on additional information derived from other domain and dependency constraints. We first relax the previous dependency **cc1** by replacing it with **cc1.1** where the I operator is replaced with the X operator.

cc1.1 $\forall t[\forall person[[t]\ \mathsf{inpocket}(person, \mathsf{gun}) \rightarrow X([t]\ \forall airplane[\neg \mathsf{poss\_board}(person, airplane)])]]$

This change would generate two preferred models for people with guns. In one, they successfully board the plane with a gun, in the other they do not. In the RAH scenario, this would then be the case for vladimir, but erik's and dimiter's situation would remain unchanged. We can not infer $\mathsf{poss\_board}(\mathsf{vladimir}, \mathsf{sas609})$ or its negation from timepoint 7 to infinity, where 7 is the end of the action where vladimir places a gun in his pocket. In the case with **cc1**, we could infer the negation

[4] Note that the scenario in Appendix 1 shows the use of strong qualification (guns) for vladimir and weak qualification (drunk) for dimiter described next.

from 7 to infinity. A visualization of the preferred entailments for the scenario which is listed in Appendix 1, but with **cc1** replaced with **cc1.1**, is provided in Appendix 3.

Now one of the advantages of this approach is its naturalness. Given a scenario, we can check whether action occurrences are successful or not by adding observations to the scenario after the action occurrence. Adding **obs5** [13] $\mathsf{onplane}(\mathsf{sas609}, \mathsf{vladimir})$ to the scenario would allow us to infer that he did in fact board the plane and $\mathsf{poss\_board}(\mathsf{vladimir}, \mathsf{sas609})$ was in fact true. He would then end up at his destination. A visualization of the preferred entailments for the scenario which is listed in Appendix 1, but with **cc1** replaced with **cc1.1** and with **obs5** added, is provided in Appendix 4. If instead we added **obs6** [13] $\neg\mathsf{onplane}(\mathsf{sas609}, \mathsf{vladimir})$, then we could infer that he was unable to board the plane and he did not end up at his destination. A visualization of the preferred entailments for the scenario which is listed in Appendix 1, but with **cc1** replaced with **cc1.1** and with **obs6** added, is provided in Appendix 5.

We are now very close to one solution to the problem of ramifications as qualification constraints. Observations are similar to domain constraints with the exception that observations generally assert facts about one timepoint while domain constraints (**acc**) assert facts generally true about one or more timepoints. Now the idea is simply that domain constraints, or other dependency constraints for that matter, when used in ramification chains may implicitly qualify actions. For example, let's extend our scenario (using **cc1.1**) with the domain constraint that it is absolutely forbidden for guns to be on planes,

acc3 $\forall t[\forall \mathit{airplane}[[t]\ \neg(\mathsf{loc}(\mathsf{gun}) \hat{=} \mathsf{loc}(\mathit{airplane}))]]$.

The location of the gun is a ramification of traveling from place to place. We have not changed **cc1.1**, but in combination with the constraints we can infer from the scenario that vladimir was unable to board the plane, not simply due to the fact that he had a gun in his pocket, but due to that and some additional domain constraints of a more generic nature that sometimes function as ramification constraints and in this case as qualification constraints. A visualization of the preferred entailments for the scenario which is listed in Appendix 1, but with cc1 replaced with **cc1.1** and **acc3** added, is provided in Appendix 6.

In conclusion, there is currently some basic understanding of the qualification problem, but we believe the techniques proposed here provide a reasonable basis for dealing with more sophisticated types of qualification than those found in the literature. We also demonstrated how our research tool VITAL [8] can be used to generate the six appendices describing the RAH scenario and visualization of preferred entailments.

Acknowledgments

This research is supported in part by the Knut and Alice Wallenberg Foundation, the Swedish Research Council for Engineering Sciences and the ECSEL graduate studies program.

References

[1] P. Doherty. Reasoning about Action and Change Using Occlusion. In: *Proceedings of the 11th European Conference on Artificial Intelligence*, 1994, 401-405.

[2] P. Doherty, W. Łukaszewicz. Circumscribing Features and Fluents. In: *Proceedings of the 1st International Conference on Temporal Logic*, Lecture Notes on Artificial Intelligence, vol. **827**, Springer-Verlag, 1994, 82-100.

[3] P. Doherty. PMON$^+$: A Fluent Logic for Action and Change, Formal Specification, Version 1.0. Department of Computer and Information Science, Linköping University. Technical report LITH-IDA-96-33. 1996. `http://www.ida.liu.se/publications/techrep/96/tr96.html`

[4] M. Ginsburg, D. E. Smith. Reasoning about Action II: the Qualification Problem, *Artificial Intelligence J.*, **35**, 1988, 311-342.

[5] E Giunchiglia and V. Lifschitz. Dependent fluents. In *Proc. of the 14th Int'l Conf. on Artificial Intelligence*, 1995.

[6] J. Gustafsson, P. Doherty. Embracing Occlusion in Specifying the Indirect Effects of Actions. In: *Proceedings of the 5th International Conference on Principles of Knowledge Representation and Reasoning*, Morgan Kaufmann Publishers, San Francisko, 1996, 87-98.

[7] J. Gustafsson, L. Karlsson. Reasoning about Actions in a Multi-agent Environment. *Linköping Electronic Articles in Computer and Information Science*, 1997. `http://www.ep.liu.se/ea/cis/1997/014`. Also submitted for journal publication.

[8] J. Kvarnström, P. Doherty. VITAL: A Research Tool for Visualizing and Querying Action Scenarios in TAL. `http://anton.ida.liu.se/vital/vital.html`.

[9] F. Lin, R. Reiter. State Constraints Revisited, *Journal of Logic and Computation*, special issue on actions and processes, **4**, 1994, 655-678.

[10] E. Sandewall. Filter Preferential Entailment for the Logic of Action and Change. In: *Proc. of the 11th Int'l Joint Conf. on Artificial Intelligence, (IJCAI-89)*, Morgan Kaufmann Publishers, 1989.

[11] E. Sandewall. *Features and Fluents. A Systematic Approach to the Representation of Knowledge about Dynamical Systems.* Volume 1. Oxford University Press, 1994.

Appendix1

DOMAIN SPECIFICATION

```
domain thing = {gun, comb1, comb2, comb3, vladimir, dimiter, erik, sas609}
domain location = {home1, home2, home3, office, airport, run609, run609b, air}
domain runway = location [run609, run609b]
domain airplane = thing [sas609]
domain person = thing [vladimir - erik]
domain pthing = thing [gun, comb1 - comb3]
domain pocket = {pocket1, pocket2, pocket3}
feature loc(thing): location showname
feature inpocket(person,pthing): boolean
durational poss_board = true
feature poss_board(person,airplane): boolean
feature drunk(person): boolean
feature onplane(airplane,person): boolean
action put(person, pthing, pocket)
action travel(person, location, location)
action fly(airplane, runway, runway)
action board(person, airplane)
```

THE NARRATIVE: OBSERVATIONS, ACTION OCCURRENCES AND TIMING

obs1 $[0]\ \mathsf{loc(vladimir)} \hat{=} \mathsf{home1} \wedge \mathsf{loc(gun)} \hat{=} \mathsf{office} \wedge \mathsf{loc(comb1)} \hat{=} \mathsf{home1} \wedge \neg\mathsf{drunk(vladimir)}$

obs2 $[0]\ \mathsf{loc(erik)} \hat{=} \mathsf{home2} \wedge \mathsf{loc(comb2)} \hat{=} \mathsf{home2} \wedge \neg\mathsf{drunk(erik)}$

obs3 $[0]\ \mathsf{loc(dimiter)} \hat{=} \mathsf{home3} \wedge \mathsf{loc(comb3)} \hat{=} \mathsf{home3} \wedge \mathsf{drunk(dimiter)}$

obs4 $[0]\ \mathsf{loc(sas609)} \hat{=} \mathsf{run609}$

occ1 $[1,2]$ put(vladimir, comb1, pocket1)

occ2 $[1,2]$ put(erik, comb2, pocket2)

occ3 $[2,4]$ travel(dimiter, home3, office)

occ4 $[3,5]$ travel(vladimir, home1, office)

occ5 $[4,6]$ travel(erik, home2, office)

occ6 $[6,7]$ put(vladimir, gun, pocket1)

occ7 $[5,7]$ travel(dimiter, office, airport)

occ8 $[7,9]$ travel(erik, office, airport)

occ9 $[8,10]$ travel(vladimir, office, airport)

occ10 $[9,10]$ board(dimiter, sas609)

occ11 $[10,11]$ board(vladimir, sas609)

occ12 $[11,12]$ board(erik, sas609)

occ13 $[13,16]$ fly(sas609, run609, run609b)

ACTION TYPES

acs1 $[t_1,t_2]\ \mathsf{fly}(\mathit{airplane}, \mathit{runway1}, \mathit{runway2}) \to [t_1]\ \mathsf{loc}(\mathit{airplane}) \hat{=} \mathit{runway1} \to$
$I((t_1,t_2)\ \mathsf{loc}(\mathit{airplane}) \hat{=} \mathsf{air}) \wedge R([t_2]\ \mathsf{loc}(\mathit{airplane}) \hat{=} \mathit{runway2})$

acs2 $[t_1,t_2]\ \mathsf{put}(\mathit{person}, \mathit{pthing}, \mathit{pocket}) \to [t_1]\ \mathsf{loc}(\mathit{person}) \hat{=} \mathsf{loc}(\mathit{pthing}) \to$
$R((t_1,t_2]\ \mathsf{inpocket}(\mathit{person}, \mathit{pthing}))$

acs3 $[t_1,t_2]\ \mathsf{travel}(\mathit{person}, \mathit{loc1}, \mathit{loc2}) \to [t_1]\ \mathsf{loc}(\mathit{person}) \hat{=} \mathit{loc1} \to R([t_2]\ \mathsf{loc}(\mathit{person}) \hat{=} \mathit{loc2})$

acs4 $[t_1,t_2]\ \mathsf{board}(\mathit{person}, \mathit{airplane}) \to [t_1]\ \mathsf{poss\_board}(\mathit{person}, \mathit{airplane}) \wedge$
$\mathsf{loc}(\mathit{person}) \hat{=} \mathsf{airport} \to R([t_2]\ \mathsf{loc}(\mathit{person}) \hat{=} \mathit{value}(t_2, \mathsf{loc}(\mathit{airplane})) \wedge$
$\mathsf{onplane}(\mathit{airplane}, \mathit{person}))$

DOMAIN CONSTRAINTS

//A pthing cannot be in two pockets at the same time.

acc1 $\forall t[\forall \mathit{pthing1}[\forall \mathit{person1}[\forall \mathit{person2}[\neg(\mathit{person1} = \mathit{person2}) \wedge$
$[t]\ \mathsf{inpocket}(\mathit{person1}, \mathit{pthing1}) \to [t]\ \neg\mathsf{inpocket}(\mathit{person2}, \mathit{pthing1})]]]]$

//A person cannot be on board two airplanes at the same time.

acc2 $\forall t[\forall \mathit{person1}[\forall \mathit{airplane1}[\forall \mathit{airplane2}[\neg(\mathit{airplane1} = \mathit{airplane2}) \wedge$
$[t]\ \mathsf{onplane}(\mathit{airplane1}, \mathit{person1}) \rightarrow [t]\ \neg\mathsf{onplane}(\mathit{airplane2}, \mathit{person1})]]]]$

DEPENDENCY CONSTRAINTS

//A person who has a gun cannot board any airplane.

cc1 $\forall t[\forall \mathit{person}[[t]\ \mathsf{inpocket}(\mathit{person}, \mathsf{gun}) \rightarrow I([t]\ \forall \mathit{airplane}[\neg \mathsf{poss\_board}(\mathit{person}, \mathit{airplane})])]]$

//A person who is drunk may not be able to board an airplane.

cc2 $\forall t[\forall \mathit{person}[[t]\ \mathsf{drunk}(\mathit{person}) \rightarrow X([t]\ \forall \mathit{airplane}[\neg \mathsf{poss\_board}(\mathit{person}, \mathit{airplane})])]]$

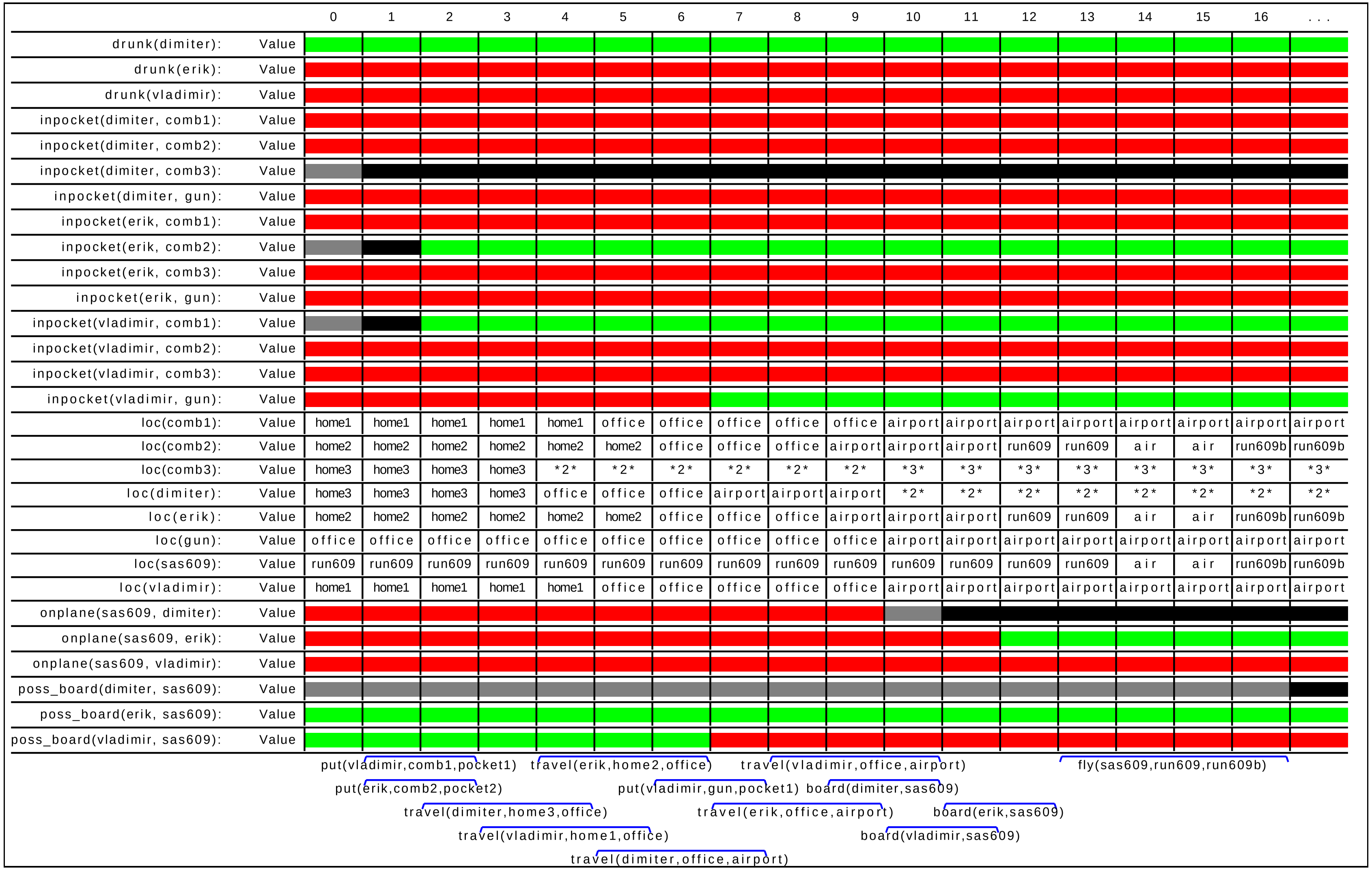
//When an airplane moves, persons on board the airplane also move.

cc3 $\forall t[\forall \mathit{airplane}[\forall \mathit{person}[\forall \mathit{loc3}[[t]\ \mathsf{onplane}(\mathit{airplane}, \mathit{person}) \wedge$
$C_T([t]\ \mathsf{loc}(\mathit{airplane}) \hat{=} \mathit{loc3}) \rightarrow R([t]\ \mathsf{loc}(\mathit{person}) \hat{=} \mathit{value}(t, \mathsf{loc}(\mathit{airplane})))]]]]$

//When persons move, things in their pockets also move.

cc4 $\forall t[\forall \mathit{person}[\forall \mathit{pthing}[\forall \mathit{loc3}[[t]\ \mathsf{inpocket}(\mathit{person}, \mathit{pthing}) \wedge$
$C_T([t]\ \mathsf{loc}(\mathit{person}) \hat{=} \mathit{loc3}) \rightarrow R([t]\ \mathsf{loc}(\mathit{pthing}) \hat{=} \mathit{value}(t, \mathsf{loc}(\mathit{person})))]]]]$

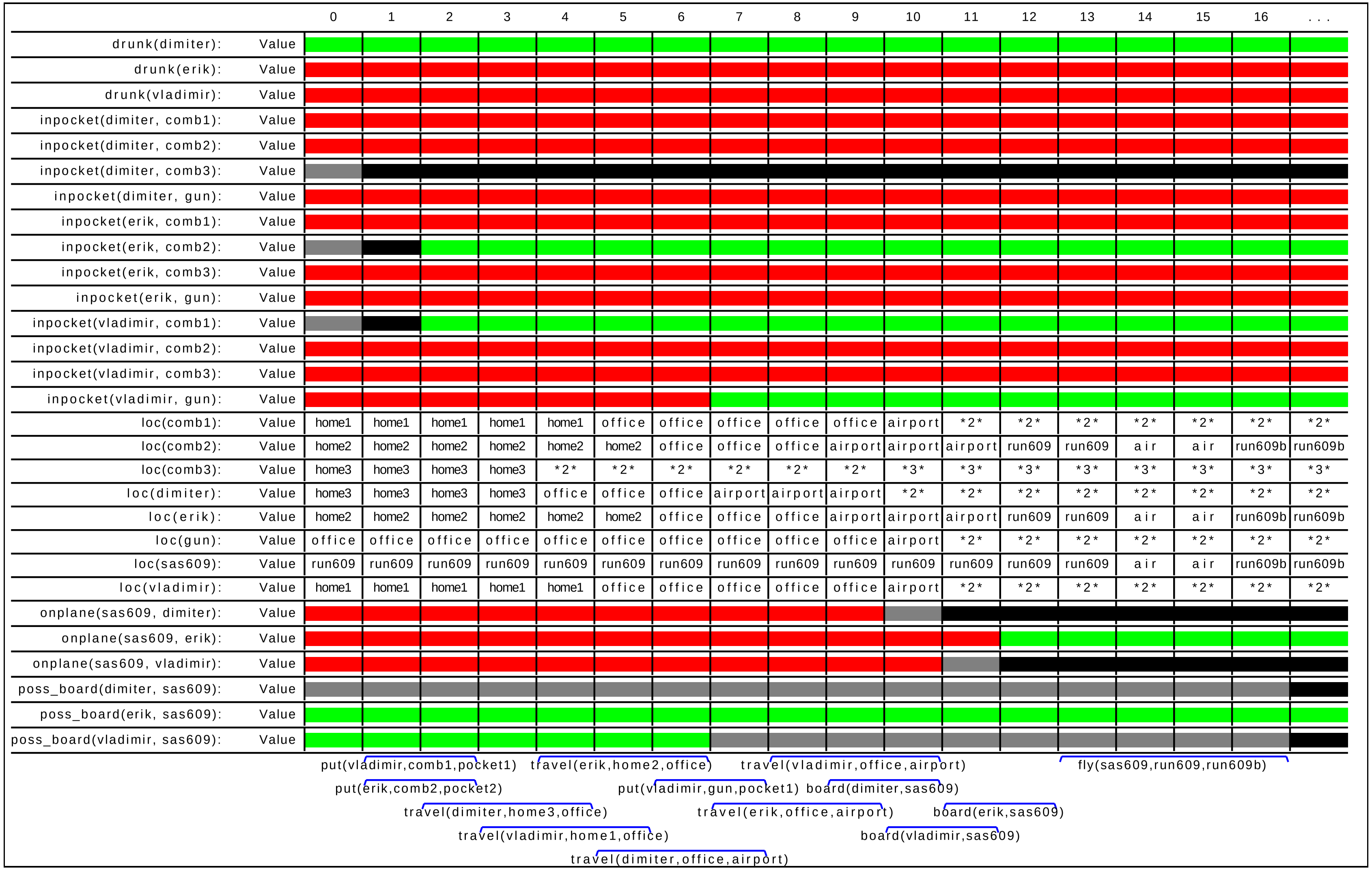
Appendix2



		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	...
drunk(dimiter):	Value	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
drunk(erik):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
drunk(vladimir):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
inpocket(dimiter, comb1):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
inpocket(dimiter, comb2):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
inpocket(dimiter, comb3):	Value	▒	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛
inpocket(dimiter, gun):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
inpocket(erik, comb1):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
inpocket(erik, comb2):	Value	▒	⬛	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
inpocket(erik, comb3):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
inpocket(erik, gun):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
inpocket(vladimir, comb1):	Value	▒	⬛	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
inpocket(vladimir, comb2):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
inpocket(vladimir, comb3):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
inpocket(vladimir, gun):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
loc(comb1):	Value	home1	home1	home1	home1	home1	office	office	office	office	office	airport	airport	airport	airport	airport	airport	airport	airport
loc(comb2):	Value	home2	home2	home2	home2	home2	home2	office	office	office	airport	airport	airport	run609	run609	air	air	run609b	run609b
loc(comb3):	Value	home3	home3	home3	home3	*2*	*2*	*2*	*2*	*2*	*2*	*3*	*3*	*3*	*3*	*3*	*3*	*3*	*3*
loc(dimiter):	Value	home3	home3	home3	home3	office	office	office	airport	airport	airport	*2*	*2*	*2*	*2*	*2*	*2*	*2*	*2*
loc(erik):	Value	home2	home2	home2	home2	home2	home2	office	office	office	airport	airport	airport	run609	run609	air	air	run609b	run609b
loc(gun):	Value	office	office	office	office	office	office	office	office	office	office	airport	airport	airport	airport	airport	airport	airport	airport
loc(sas609):	Value	run609	run609	run609	run609	run609	run609	run609	run609	run609	run609	run609	run609	run609	run609	air	air	run609b	run609b
loc(vladimir):	Value	home1	home1	home1	home1	home1	office	office	office	office	office	airport	airport	airport	airport	airport	airport	airport	airport
onplane(sas609, dimiter):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	▒	⬛	⬛	⬛	⬛	⬛	⬛	⬛
onplane(sas609, erik):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟩	🟩
onplane(sas609, vladimir):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
poss_board(dimiter, sas609):	Value	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	▒	⬛
poss_board(erik, sas609):	Value	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
poss_board(vladimir, sas609):	Value	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥

put(vladimir,comb1,pocket1)
travel(erik,home2,office)
travel(vladimir,office,airport)
fly(sas609,run609,run609b)
put(erik,comb2,pocket2)
put(vladimir,gun,pocket1)
board(dimiter,sas609)
travel(dimiter,home3,office)
travel(erik,office,airport)
board(erik,sas609)
travel(vladimir,home1,office)
board(vladimir,sas609)
travel(dimiter,office,airport)

Appendix3



		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	. . .
drunk(dimiter):	Value	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
drunk(erik):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
drunk(vladimir):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
inpocket(dimiter, comb1):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
inpocket(dimiter, comb2):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
inpocket(dimiter, comb3):	Value	▨	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛
inpocket(dimiter, gun):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
inpocket(erik, comb1):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
inpocket(erik, comb2):	Value	▨	⬛	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
inpocket(erik, comb3):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
inpocket(erik, gun):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
inpocket(vladimir, comb1):	Value	▨	⬛	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
inpocket(vladimir, comb2):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
inpocket(vladimir, comb3):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
inpocket(vladimir, gun):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
loc(comb1):	Value	home1	home1	home1	home1	home1	office	office	office	office	office	airport	*2*	*2*	*2*	*2*	*2*	*2*	*2*
loc(comb2):	Value	home2	home2	home2	home2	home2	home2	office	office	office	airport	airport	airport	run609	run609	air	air	run609b	run609b
loc(comb3):	Value	home3	home3	home3	home3	*2*	*2*	*2*	*2*	*2*	*2*	*3*	*3*	*3*	*3*	*3*	*3*	*3*	*3*
loc(dimiter):	Value	home3	home3	home3	home3	office	office	office	airport	airport	airport	*2*	*2*	*2*	*2*	*2*	*2*	*2*	*2*
loc(erik):	Value	home2	home2	home2	home2	home2	home2	office	office	office	airport	airport	airport	run609	run609	air	air	run609b	run609b
loc(gun):	Value	office	office	office	office	office	office	office	office	office	office	airport	*2*	*2*	*2*	*2*	*2*	*2*	*2*
loc(sas609):	Value	run609	run609	run609	run609	run609	run609	run609	run609	run609	run609	run609	run609	run609	run609	air	air	run609b	run609b
loc(vladimir):	Value	home1	home1	home1	home1	home1	office	office	office	office	office	airport	*2*	*2*	*2*	*2*	*2*	*2*	*2*
onplane(sas609, dimiter):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	▨	⬛	⬛	⬛	⬛	⬛	⬛	⬛
onplane(sas609, erik):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟩	🟩
onplane(sas609, vladimir):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	▨	⬛	⬛	⬛	⬛	⬛	⬛
poss_board(dimiter, sas609):	Value	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	⬛
poss_board(erik, sas609):	Value	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
poss_board(vladimir, sas609):	Value	🟩	🟩	🟩	🟩	🟩	🟩	🟩	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	⬛

put(vladimir,comb1,pocket1) travel(erik,home2,office) travel(vladimir,office,airport) fly(sas609,run609,run609b)

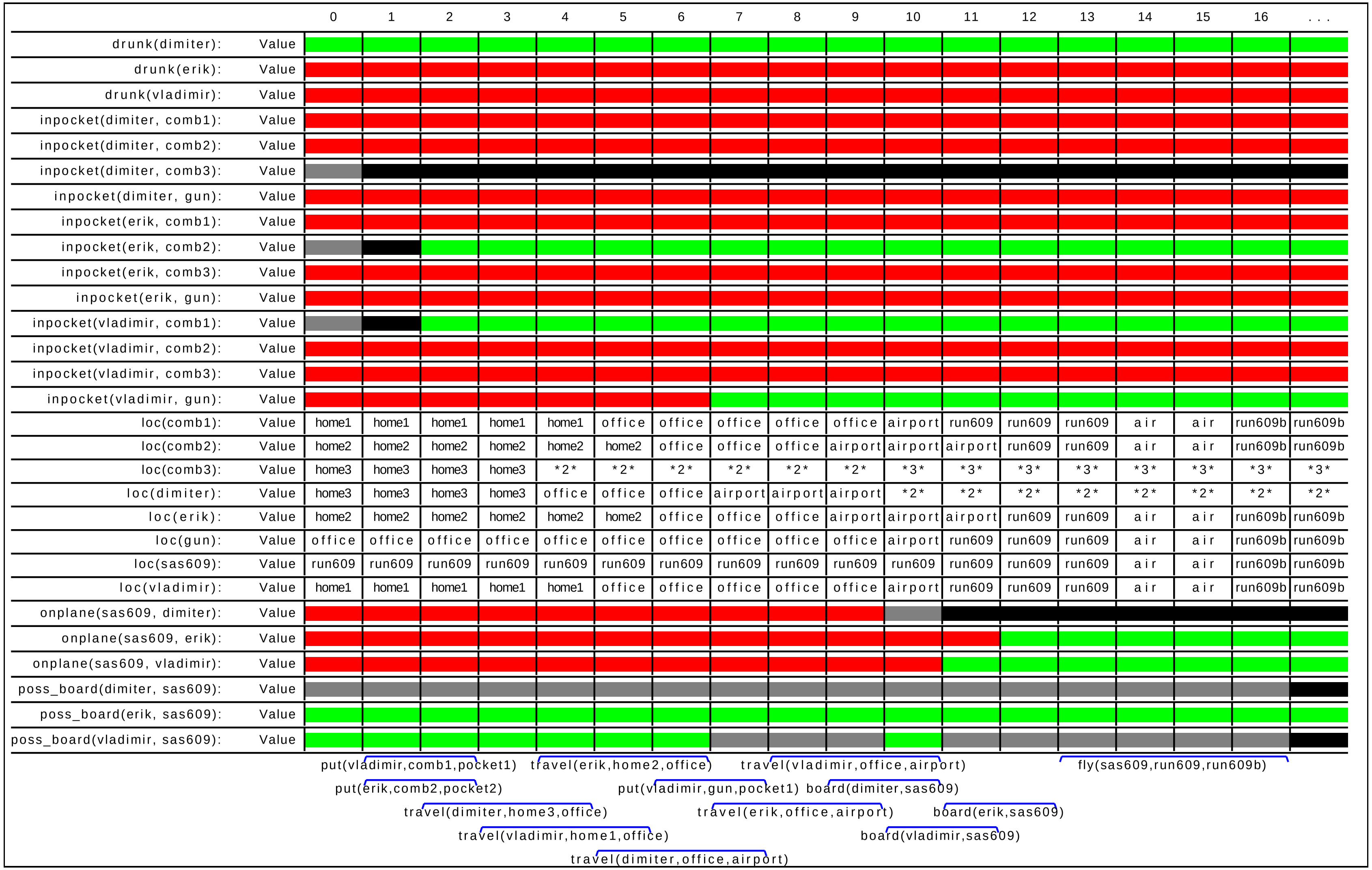
put(erik,comb2,pocket2) put(vladimir,gun,pocket1) board(dimiter,sas609)

travel(dimiter,home3,office) travel(erik,office,airport) board(erik,sas609)

travel(vladimir,home1,office) board(vladimir,sas609)

travel(dimiter,office,airport)

Appendix4



		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	...
drunk(dimiter):	Value	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
drunk(erik):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
drunk(vladimir):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
inpocket(dimiter, comb1):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
inpocket(dimiter, comb2):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
inpocket(dimiter, comb3):	Value	▨	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛
inpocket(dimiter, gun):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
inpocket(erik, comb1):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
inpocket(erik, comb2):	Value	▨	⬛	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
inpocket(erik, comb3):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
inpocket(erik, gun):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
inpocket(vladimir, comb1):	Value	▨	⬛	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
inpocket(vladimir, comb2):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
inpocket(vladimir, comb3):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
inpocket(vladimir, gun):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
loc(comb1):	Value	home1	home1	home1	home1	home1	office	office	office	office	office	airport	run609	run609	run609	air	air	run609b	run609b
loc(comb2):	Value	home2	home2	home2	home2	home2	home2	office	office	office	airport	airport	airport	run609	run609	air	air	run609b	run609b
loc(comb3):	Value	home3	home3	home3	home3	*2*	*2*	*2*	*2*	*2*	*2*	*3*	*3*	*3*	*3*	*3*	*3*	*3*	*3*
loc(dimiter):	Value	home3	home3	home3	home3	office	office	office	airport	airport	airport	*2*	*2*	*2*	*2*	*2*	*2*	*2*	*2*
loc(erik):	Value	home2	home2	home2	home2	home2	home2	office	office	office	airport	airport	airport	run609	run609	air	air	run609b	run609b
loc(gun):	Value	office	office	office	office	office	office	office	office	office	office	airport	run609	run609	run609	air	air	run609b	run609b
loc(sas609):	Value	run609	run609	run609	run609	run609	run609	run609	run609	run609	run609	run609	run609	run609	run609	air	air	run609b	run609b
loc(vladimir):	Value	home1	home1	home1	home1	home1	office	office	office	office	office	airport	run609	run609	run609	air	air	run609b	run609b
onplane(sas609, dimiter):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	▨	⬛	⬛	⬛	⬛	⬛	⬛	⬛
onplane(sas609, erik):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟩	🟩
onplane(sas609, vladimir):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟩	🟩	🟩
poss_board(dimiter, sas609):	Value	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	⬛
poss_board(erik, sas609):	Value	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
poss_board(vladimir, sas609):	Value	🟩	🟩	🟩	🟩	🟩	🟩	🟩	▨	▨	▨	🟩	▨	▨	▨	▨	▨	▨	⬛

put(vladimir,comb1,pocket1)
travel(erik,home2,office)
travel(vladimir,office,airport)
fly(sas609,run609,run609b)
put(erik,comb2,pocket2)
put(vladimir,gun,pocket1)
board(dimiter,sas609)
travel(dimiter,home3,office)
travel(erik,office,airport)
board(erik,sas609)
travel(vladimir,home1,office)
board(vladimir,sas609)
travel(dimiter,office,airport)

Appendix5

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	. . .
drunk(dimiter):	Value	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
drunk(erik):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
drunk(vladimir):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
inpocket(dimiter, comb1):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
inpocket(dimiter, comb2):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
inpocket(dimiter, comb3):	Value	▨	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛
inpocket(dimiter, gun):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
inpocket(erik, comb1):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
inpocket(erik, comb2):	Value	▨	⬛	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
inpocket(erik, comb3):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
inpocket(erik, gun):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
inpocket(vladimir, comb1):	Value	▨	⬛	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
inpocket(vladimir, comb2):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
inpocket(vladimir, comb3):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
inpocket(vladimir, gun):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
loc(comb1):	Value	home1	home1	home1	home1	home1	office	office	office	office	office	airport	airport	airport	airport	airport	airport	airport	airport
loc(comb2):	Value	home2	home2	home2	home2	home2	home2	office	office	office	airport	airport	airport	run609	run609	air	air	run609b	run609b
loc(comb3):	Value	home3	home3	home3	home3	*2*	*2*	*2*	*2*	*2*	*2*	*3*	*3*	*3*	*3*	*3*	*3*	*3*	*3*
loc(dimiter):	Value	home3	home3	home3	home3	office	office	office	airport	airport	airport	*2*	*2*	*2*	*2*	*2*	*2*	*2*	*2*
loc(erik):	Value	home2	home2	home2	home2	home2	home2	office	office	office	airport	airport	airport	run609	run609	air	air	run609b	run609b
loc(gun):	Value	office	office	office	office	office	office	office	office	office	office	airport	airport	airport	airport	airport	airport	airport	airport
loc(sas609):	Value	run609	run609	run609	run609	run609	run609	run609	run609	run609	run609	run609	run609	run609	run609	air	air	run609b	run609b
loc(vladimir):	Value	home1	home1	home1	home1	home1	office	office	office	office	office	airport	airport	airport	airport	airport	airport	airport	airport
onplane(sas609, dimiter):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	▨	⬛	⬛	⬛	⬛	⬛	⬛	⬛
onplane(sas609, erik):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟩	🟩
onplane(sas609, vladimir):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
poss_board(dimiter, sas609):	Value	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	⬛
poss_board(erik, sas609):	Value	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
poss_board(vladimir, sas609):	Value	🟩	🟩	🟩	🟩	🟩	🟩	🟩	▨	▨	▨	🟥	▨	▨	▨	▨	▨	▨	⬛

put(vladimir,comb1,pocket1) travel(erik,home2,office) travel(vladimir,office,airport) fly(sas609,run609,run609b)

put(erik,comb2,pocket2) put(vladimir,gun,pocket1) board(dimiter,sas609)

travel(dimiter,home3,office) travel(erik,office,airport) board(erik,sas609)

travel(vladimir,home1,office) board(vladimir,sas609)

travel(dimiter,office,airport)

Appendix6

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	. . .
drunk(dimiter):	Value	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
drunk(erik):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
drunk(vladimir):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
inpocket(dimiter, comb1):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
inpocket(dimiter, comb2):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
inpocket(dimiter, comb3):	Value	▨	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛
inpocket(dimiter, gun):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
inpocket(erik, comb1):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
inpocket(erik, comb2):	Value	▨	⬛	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
inpocket(erik, comb3):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
inpocket(erik, gun):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
inpocket(vladimir, comb1):	Value	▨	⬛	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
inpocket(vladimir, comb2):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
inpocket(vladimir, comb3):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
inpocket(vladimir, gun):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
loc(comb1):	Value	home1	home1	home1	home1	home1	office	office	office	office	office	airport	airport	airport	airport	airport	airport	airport	airport
loc(comb2):	Value	home2	home2	home2	home2	home2	home2	office	office	office	airport	airport	airport	run609	run609	air	air	run609b	run609b
loc(comb3):	Value	home3	home3	home3	home3	*2*	*2*	*2*	*2*	*2*	*2*	*3*	*3*	*3*	*3*	*3*	*3*	*3*	*3*
loc(dimiter):	Value	home3	home3	home3	home3	office	office	office	airport	airport	airport	*2*	*2*	*2*	*2*	*2*	*2*	*2*	*2*
loc(erik):	Value	home2	home2	home2	home2	home2	home2	office	office	office	airport	airport	airport	run609	run609	air	air	run609b	run609b
loc(gun):	Value	office	office	office	office	office	office	office	office	office	office	airport	airport	airport	airport	airport	airport	airport	airport
loc(sas609):	Value	run609	run609	run609	run609	run609	run609	run609	run609	run609	run609	run609	run609	run609	run609	air	air	run609b	run609b
loc(vladimir):	Value	home1	home1	home1	home1	home1	office	office	office	office	office	airport	airport	airport	airport	airport	airport	airport	airport
onplane(sas609, dimiter):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	▨	⬛	⬛	⬛	⬛	⬛	⬛	⬛
onplane(sas609, erik):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟩	🟩	🟩	🟩	🟩	🟩
onplane(sas609, vladimir):	Value	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
poss_board(dimiter, sas609):	Value	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	⬛
poss_board(erik, sas609):	Value	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
poss_board(vladimir, sas609):	Value	🟩	🟩	🟩	🟩	🟩	🟩	🟩	▨	▨	▨	🟥	▨	▨	▨	▨	▨	▨	⬛

put(vladimir,comb1,pocket1) travel(erik,home2,office) travel(vladimir,office,airport) fly(sas609,run609,run609b)

put(erik,comb2,pocket2) put(vladimir,gun,pocket1) board(dimiter,sas609)

travel(dimiter,home3,office) travel(erik,office,airport) board(erik,sas609)

travel(vladimir,home1,office) board(vladimir,sas609)

travel(dimiter,office,airport)