IDA Homework - Analysis of Formal Verification Algorithms

Akos Hajdu

2016

Contents

1 Introduction

1.1 Background

In my research, I am working on the formal verification of software and hardware systems. I create a formal model (a formal representation) of the system and the desired property using first order logic formulas. Then, the set of possible states of the system (state space) can be explored to check whether it is safe, i.e., the desired property holds for each reachable state (e.g., a variable `x` should never be negative during the execution of a software). However, a major drawback of formal verification techniques is the so-called state space explosion problem, which means that the set of possible states of a system can be unmanageably or even infinitely large. To overcome this problem, I use abstraction-based techniques, which reduce complexity by hiding information that is not relevant for verification. However, finding the proper precision of abstraction is a difficult task. Counterexample-Guided Abstraction Refinement (CEGAR) is an automatic verification algorithm that starts with a coarse abstraction and refines it iteratively until the proper precision is obtained. CEGAR is a general concept, having numerous variants in the literature, with different variants performing better for different tasks. The main focus of my research is the development of a generic CEGAR framework that can incorporate different variants of the CEGAR algorithm. This way, the most appropriate variant can be choosed for a particular verification task. In this homework I run several variants of the algorithm on different models and analyze the results.

1.2 Variables of the problem

1.2.1 Input variables

- Parameters of the model (the formal representation of the system under verification)
  - **Type** of the model (hardware or plc)
  - Name of the **model**
  - Number of **variables** in the model
  - **Size** of the formulas describing the model
- Parameters of the algorithm (the algorithm configuration)
  - Abstract **domain** (predicate or explicit)
  - **Refinement** strategy (Craig interpolation, sequence interpolation, unsat core)
  - **Initial precision** of the abstraction (empty or property-based)
  - **Search** strategy (BFS or DFS)

Constraints: unsat core refinement cannot be used with predicate domain. Besides that, all combinations of the algorithm parameters are valid configurations.

1.2.2 Output variables

- Is the model **safe**, i.e., does the property hold
- Execution **time** of the verification
- Number of refinement **iterations**
- **Size** of the ARG (Abstract Reachability Graph, i.e., the abstract state space)
- **Depth** of the ARG
- **Length** of the counterexample (if the model is not safe)

Constraints: it is possible that the algorithm did not terminate within a specified time (see later). In this case all output variables are `NA` values. Furthermore, the length of the counterexample is `NA` if the model is safe.

2 Summary of the data

Common variables for the script:

```
timeout.ms <- 480000                         # Timeout used during the measurements
csv.path = "log_20161209_153530_hw_plc_vm.csv" # Path of the data
```

2.1 Load and clean data

```
d <- read.csv(csv.path, header = T, sep = ",", quote = "", na.strings = "")
# Formatting: trim the name of the model, determine new variable 'Type'
d$Model <- as.factor(gsub("models/", "", d$Model))
d$Model <- as.factor(substr(d$Model, 0, regexpr("\\.[^\\.]*$", d$Model) - 1))
d <- data.frame(Type = as.factor(substr(d$Model, 0, regexpr('/', d$Model) - 1)), d)
# Filter timeouts
d.no.to <- d %>% filter(!is.na(TimeMs))
```

2.2 Summary

There are **18** input models and **20** algorithm configurations. Each measurement was repeated **5** times, yielding a total number of **1800** measurement points with **1120** successful verifications (non-timeouts) (**62%**). Total time of the measurements: **4 days 6 hours**.

```
str(d, width = 80, strict.width = "cut") # Check column types
```

```
## 'data.frame':    1800 obs. of  14 variables:
##  $ Type       : Factor w/ 2 levels "hw","plc": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Model      : Factor w/ 18 levels "hw/bob2","hw/eijks208c",..: 1 1 1 1 1 1 1 ..
##  $ Vars       : int  159 159 159 159 159 159 159 159 159 159 ...
##  $ Size       : int  2968 2968 2968 2968 2968 2968 2968 2968 2968 2968 ...
##  $ Domain     : Factor w/ 2 levels "EXPL","PRED": 2 2 2 2 2 2 2 2 2 2 ...
##  $ Refinement: Factor w/ 3 levels "CRAIG_ITP","SEQ_ITP",..: 1 1 1 1 1 1 1 1 1 ..
##  $ InitPrec   : Factor w/ 2 levels "EMPTY","PROP": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Search     : Factor w/ 2 levels "BFS","DFS": 1 1 1 1 1 2 2 2 2 2 ...
##  $ Safe       : Factor w/ 2 levels "false","true": 2 2 2 2 2 2 2 2 2 2 ...
##  $ TimeMs     : int  4328 4326 4354 4263 4379 4026 4110 4020 4285 4017 ...
##  $ Iterations: int  12 12 12 12 12 14 14 14 14 14 ...
##  $ ARGsize    : int  23 23 23 23 23 17 17 17 17 17 ...
##  $ ARGdepth   : int  5 5 5 5 5 5 5 5 5 5 ...
##  $ CEXlen     : int  NA NA NA NA NA NA NA NA NA NA ...
```

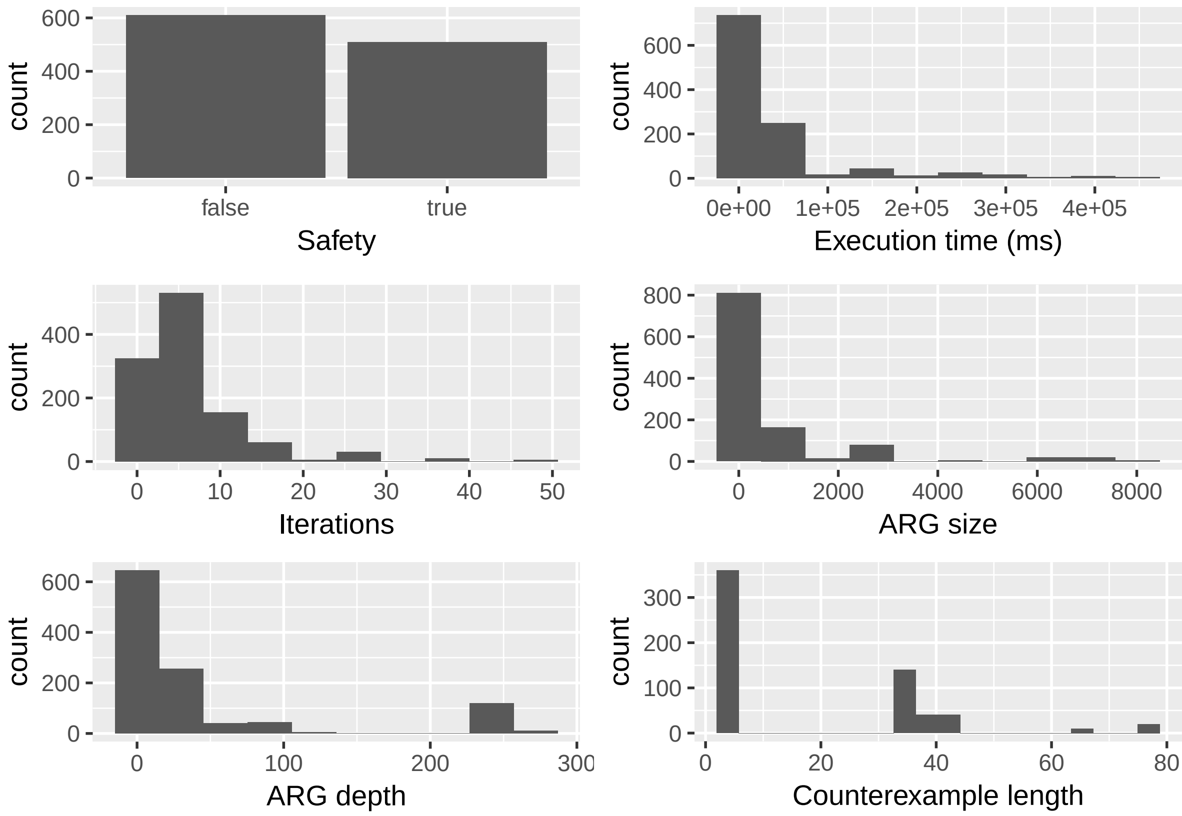
```
summary(d) # Check summary values
```

```
##   Type                          Model          Vars             Size
##  hw :1200   hw/bob2               : 100   Min.   : 30.0   Min.   :1226
##  plc: 600   hw/eijks208c          : 100   1st Qu.: 83.0   1st Qu.:1783
##             hw/eijks208o          : 100   Median : 99.0   Median :3280
##             hw/nusmv.syncarb10_2.B: 100   Mean   :159.5   Mean   :4046
##             hw/nusmv.syncarb5_2.B : 100   3rd Qu.:188.0   3rd Qu.:6730
##             hw/pdtpmsarbiter      : 100   Max.   :382.0   Max.   :7218
##             (Other)               :1200
##   Domain          Refinement    InitPrec    Search       Safe
##  EXPL:1080   CRAIG_ITP :720   EMPTY:900   BFS:900   false:610
##  PRED: 720   SEQ_ITP   :720   PROP :900   DFS:900   true :510
##              UNSAT_CORE:360                         NA's :680
##
##
##
##
##      TimeMs         Iterations        ARGsize          ARGdepth
##  Min.   :   515   Min.   : 1.000   Min.   :   2.0   Min.   :  1.00
##  1st Qu.:  1376   1st Qu.: 2.000   1st Qu.:  17.0   1st Qu.:  3.00
##  Median :  3571   Median : 4.000   Median :  37.0   Median :  7.00
##  Mean   : 38372   Mean   : 6.491   Mean   : 694.7   Mean   : 46.75
##  3rd Qu.: 33776   3rd Qu.: 8.000   3rd Qu.: 744.0   3rd Qu.: 37.25
##  Max.   :448544   Max.   :49.000   Max.   :8018.0   Max.   :273.00
##  NA's   :680      NA's   :680      NA's   :680      NA's   :680
##      CEXlen
##  Min.   : 2.00
##  1st Qu.: 2.00
##  Median : 3.00
##  Mean   :17.72
##  3rd Qu.:33.00
##  Max.   :75.00
##  NA's   :1190
```

2.3 Histograms of output variables

The following histograms show an overview of the distribution of the output variables.

```
plots = list()
plots[[1]] <- qplot(d.no.to$Safe,
                    xlab = "Safety")
plots[[2]] <- qplot(d.no.to$TimeMs, geom = "histogram", bins = 10,
                    xlab = "Execution time (ms)")
plots[[3]] <- qplot(d.no.to$Iterations, geom = "histogram", bins = 10,
                    xlab = "Iterations")
plots[[4]] <- qplot(d.no.to$ARGsize, geom = "histogram", bins = 10,
                    xlab = "ARG size")
plots[[5]] <- qplot(d.no.to$ARGdepth, geom = "histogram", bins = 10,
                    xlab = "ARG depth")
plots[[6]] <- qplot(filter(d.no.to, !is.na(d.no.to$CEXlen))$CEXlen,
                    geom = "histogram", bins = 20, xlab = "Counterexample length")
do.call(grid.arrange, plots)
```

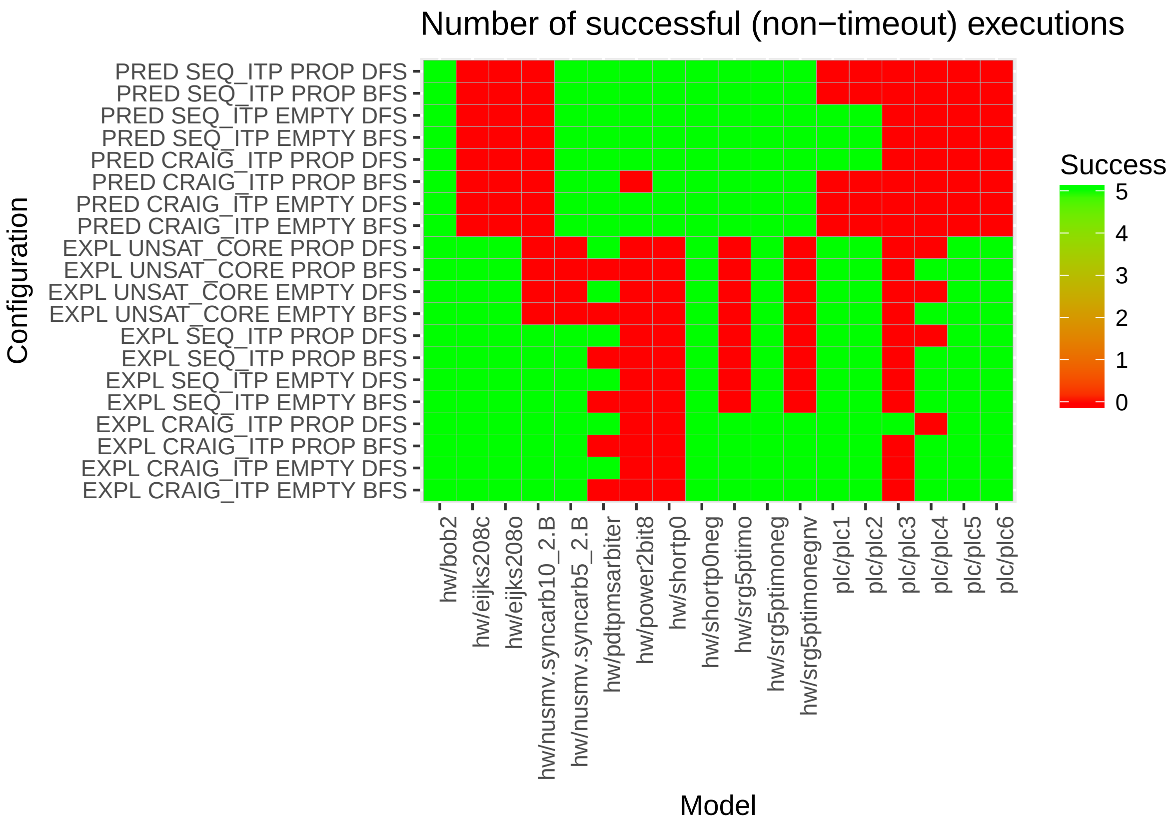


3 Basic analysis

3.1 Number of successful executions

The following plot shows that each model was verified by at least one configuration. It is interesting that there was a model (plc3), which could only be verified by a single configuration. It can also be observed, that either all repetitions of a measurement succeeded or all timed out. Regarding the configurations, it can be seen that some of them can verify almost every model, while others can only verify about half of them. However, none of the configurations could verify every model.

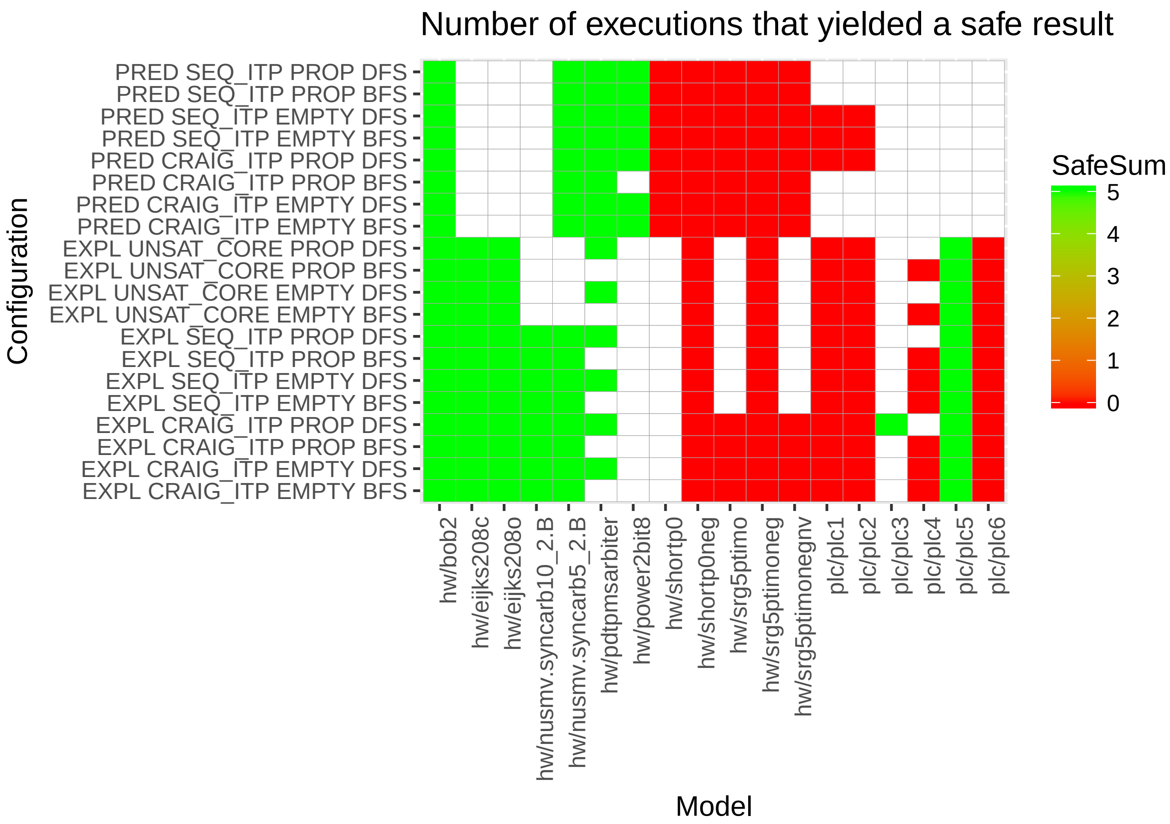
```r
# Combine parameter columns into a single string column
d.confs <- data.frame(d, Config = paste(d$Domain, d$Refinement,
                                        d$InitPrec, d$Search, sep = " "))
# Calculate number of successful runs for each configuration and model
model.conf.succ <- d.confs %>% group_by(Config, Model) %>%
  summarise(Success = sum(ifelse(is.na(Safe), 0, 1)))
# Plot results
ggplot(model.conf.succ, aes(Model, Config, fill = Success)) +
  geom_tile(colour = "darkgray") +
  scale_fill_gradient(low = "red", high = "green") +
  labs(title = "Number of successful (non-timeout) executions",
       x = "Model", y = "Configuration") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



3.2 Safety results

The following plot shows that (1) all configurations agree on the safety of the model (whether it meets the property or not) and (2) either all executions of a configuration yielded safe or all yielded unsafe. The lack of the previous properties would obviously mean that the algorithms are not sound.

```r
# Calculate number of safe results for each configuration and model
model.conf.safe <- d.confs %>% group_by(Config, Model) %>%
  summarise(SafeSum = sum(ifelse(Safe == "true", 1, 0)))
# Plot results
ggplot(model.conf.safe, aes(Model, Config, fill = SafeSum)) +
  geom_tile(colour = "darkgray") +
  scale_fill_gradient(low = "red", high = "green", na.value = "white") +
  labs(title = "Number of executions that yielded a safe result",
       x = "Model", y = "Configuration") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



3.3 Average execution times

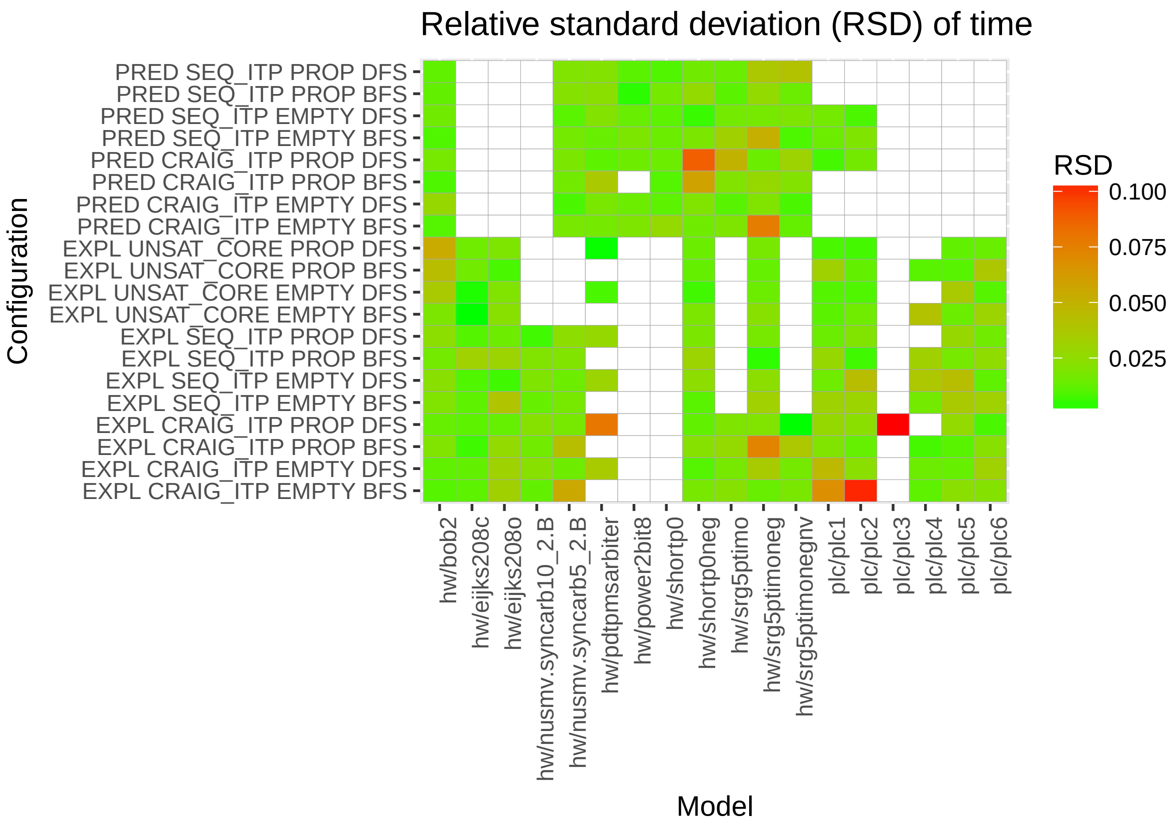
The following plot shows that there are 2-3 orders of magnitude difference between average execution times. It is not surprising that the same configuration performs differently for different models, but it is interesting that for certain models there is a great difference in performance with the change of a single parameter of the configuration. It is also quite surprising that there is a model, where only one configuration was successful, but with a short execution time. This case could be the subject of detailed analysis in the future.

```
# Calculate average execution time and RSD for each configuration and model
model.conf.time <- d.confs %>% group_by(Config, Model) %>%
  summarise(AvgTime = log10(mean(TimeMs)), RSD = sd(TimeMs)/mean(TimeMs))
# Plot average time
ggplot(model.conf.time, aes(Model, Config, fill = AvgTime)) +
  geom_tile(colour = "darkgray") +
  scale_fill_gradient(low = "green", high = "red", na.value = "white") +
  labs(title = "Average execution time (log10, milliseconds)",
       x = "Model", y = "Configuration") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```

3.4 Relative standard deviation of measurements

The following plot shows that the relative standard deviation of the repeated executions is low (maximum is 0.1044726). This indicates a robust measurement environment.

```
# Plot RSD
ggplot(model.conf.time, aes(Model, Config, fill = RSD)) +
  geom_tile(colour = "darkgray") +
  scale_fill_gradient(low = "green", high = "red", na.value = "white") +
  labs(title = "Relative standard deviation (RSD) of time",
       x = "Model", y = "Configuration") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



4 Correlations

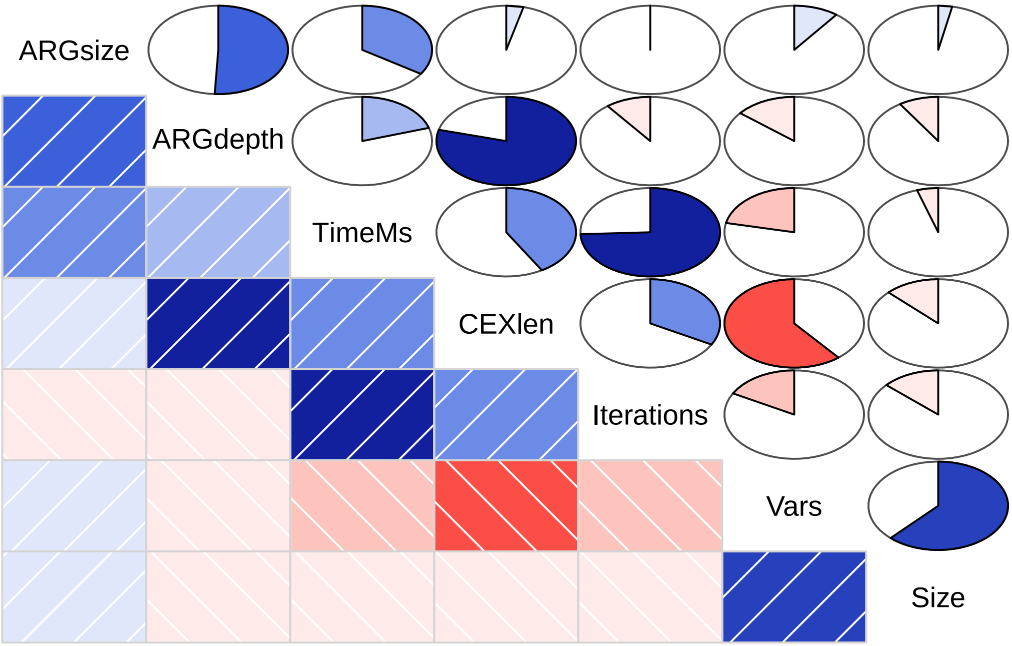
4.1 On the whole data set

The following correlation matrices and correlograms show some correlation between variables, but it is not convincing enough. Therefore, I split the results into two parts based on the type of the model (hardware or PLC).

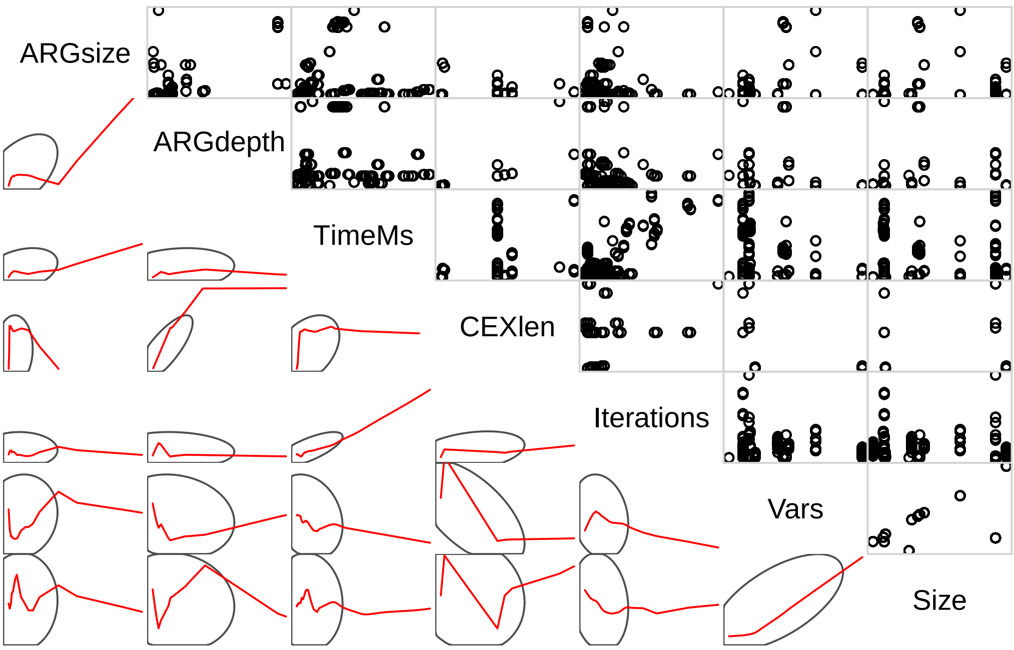
```
# Select numerical data
d.corr <- select(d, TimeMs, Vars, Size, Iterations, ARGsize, ARGdepth, CEXlen)
# Correlation matrix
round(cor(d.corr, use = "pairwise.complete.obs"), 4)
```

```
##             TimeMs    Vars    Size Iterations ARGsize ARGdepth  CEXlen
## TimeMs      1.0000 -0.2163 -0.0492     0.7430  0.3414   0.2018  0.4148
## Vars       -0.2163  1.0000  0.6197    -0.1704  0.1033  -0.1409 -0.6093
## Size       -0.0492  0.6197  1.0000    -0.1335  0.0322  -0.0921 -0.1262
## Iterations  0.7430 -0.1704 -0.1335     1.0000 -0.0231  -0.1045  0.3294
## ARGsize     0.3414  0.1033  0.0322    -0.0231  1.0000   0.5082  0.0398
## ARGdepth    0.2018 -0.1409 -0.0921    -0.1045  0.5082   1.0000  0.7912
## CEXlen      0.4148 -0.6093 -0.1262     0.3294  0.0398   0.7912  1.0000
```

```
# Correlograms
corrgram(d.corr, order = TRUE, lower.panel = panel.shade, upper.panel = panel.pie)
```




```
corrgram(d.corr, order = TRUE, lower.panel = panel.ellipse, upper.panel = panel.pts)
```



4.2 On hardware models

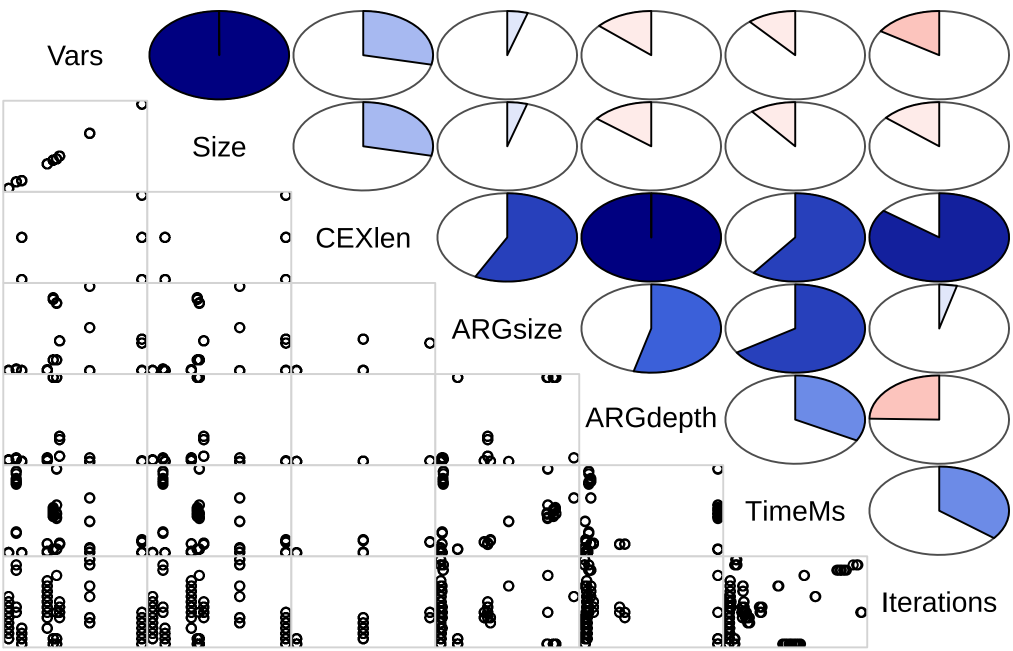
The following matrices and correlograms show some strong correlations.

- Correlation between the number of variables and the size of the model is due to the formal representation of hardware circuits (e.g., for each gate output, a unique variable is assigned).
- The correlation between the depth of the ARG and the length of the counterexample may indicate that counterexamples are always on the deepest level of the explored state space.
- Correlation between the number of iterations and the counterexample length indicates that for each step towards the target state, a refinement was required.

```
# Filter to hardware
d.corr.hw <- select(filter(d, Type == "hw"), TimeMs, Vars, Size, Iterations, ARGsize,
                    ARGdepth, CEXlen)
# Correlation matrix
round(cor(d.corr.hw, use = "pairwise.complete.obs"), 4)
```

```
##              TimeMs    Vars    Size Iterations ARGsize ARGdepth CEXlen
## TimeMs       1.0000 -0.1131 -0.1061     0.3565  0.6584   0.3283 0.6018
## Vars        -0.1131  1.0000  0.9992    -0.1586  0.0475  -0.1344 0.2845
## Size        -0.1061  0.9992  1.0000    -0.1384  0.0461  -0.1425 0.2845
## Iterations   0.3565 -0.1586 -0.1384     1.0000  0.0409  -0.2466 0.8531
## ARGsize      0.6584  0.0475  0.0461     0.0409  1.0000   0.5405 0.5755
## ARGdepth     0.3283 -0.1344 -0.1425    -0.2466  0.5405   1.0000 1.0000
## CEXlen       0.6018  0.2845  0.2845     0.8531  0.5755   1.0000 1.0000
```

```
# Correlogram
corrgram(d.corr.hw, order = TRUE, lower.panel = panel.pts, upper.panel = panel.pie)
```



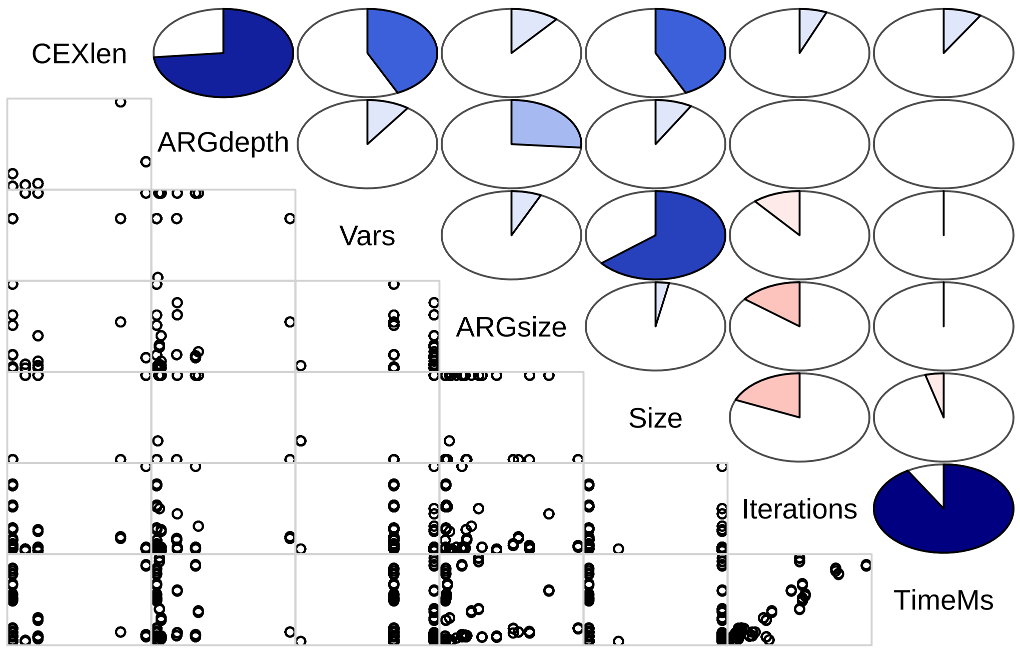
4.3 On PLC models

Correlations for PLC models are less significant. This may be due to the high diversity of PLC models.

```
# Filter to PLC
d.corr.plc <- select(filter(d, Type == "plc"), TimeMs, Vars, Size, Iterations, ARGsize,
                     ARGdepth, CEXlen)
# Correlation matrix
round(cor(d.corr.plc, use = "pairwise.complete.obs"), 4)
```

```
##              TimeMs    Vars    Size Iterations ARGsize ARGdepth CEXlen
## TimeMs       1.0000  0.0093 -0.0425     0.9139  0.0243   0.0017 0.0883
## Vars         0.0093  1.0000  0.6409    -0.1104  0.0691   0.0978 0.4287
## Size        -0.0425  0.6409  1.0000    -0.1847  0.0311   0.0847 0.4287
## Iterations   0.9139 -0.1104 -0.1847     1.0000 -0.1442   0.0028 0.0632
## ARGsize      0.0243  0.0691  0.0311    -0.1442  1.0000   0.2625 0.1114
## ARGdepth     0.0017  0.0978  0.0847     0.0028  0.2625   1.0000 0.7347
## CEXlen       0.0883  0.4287  0.4287     0.0632  0.1114   0.7347 1.0000
```

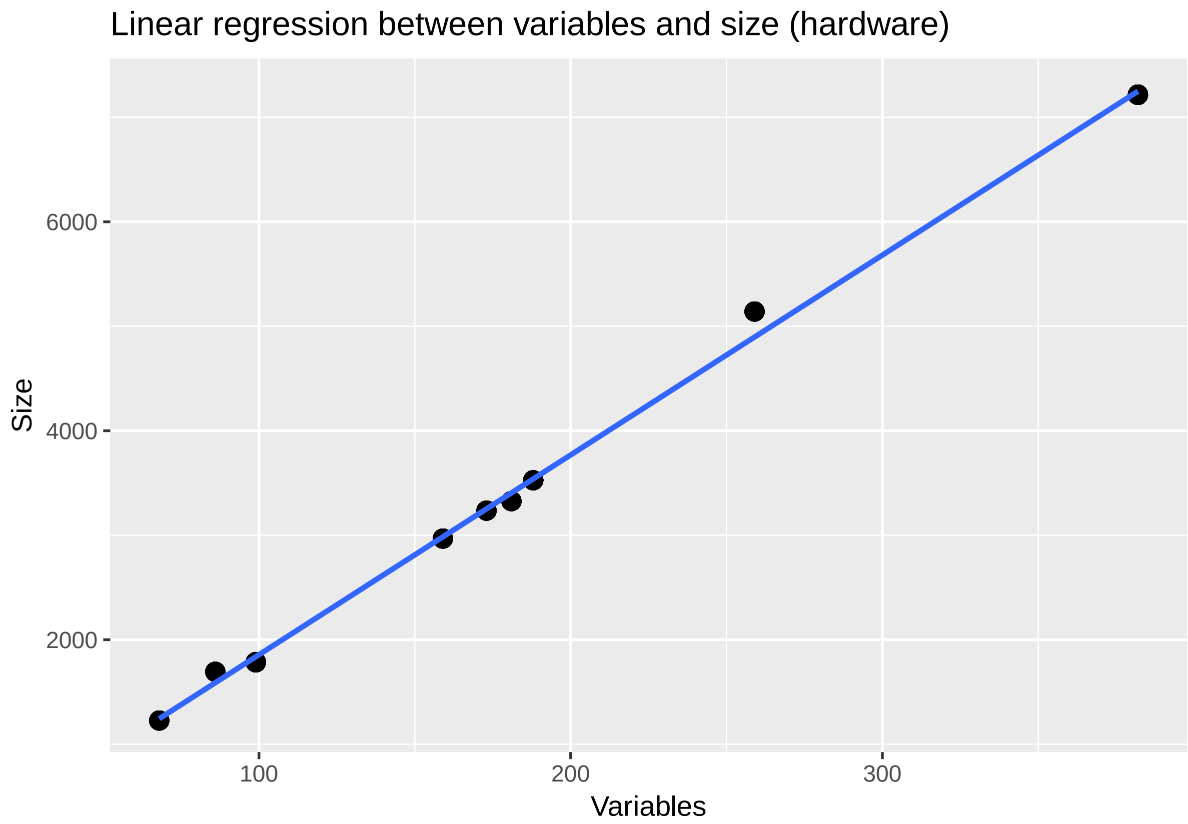
```
# Correlogram
corrgram(d.corr.plc, order = TRUE, lower.panel = panel.pts, upper.panel = panel.pie)
```



4.4 Linear regressions

Based on the correlations above, some linear regressions were also calculated. The R-squared values show a strong correlation (R-squared > 0.8).

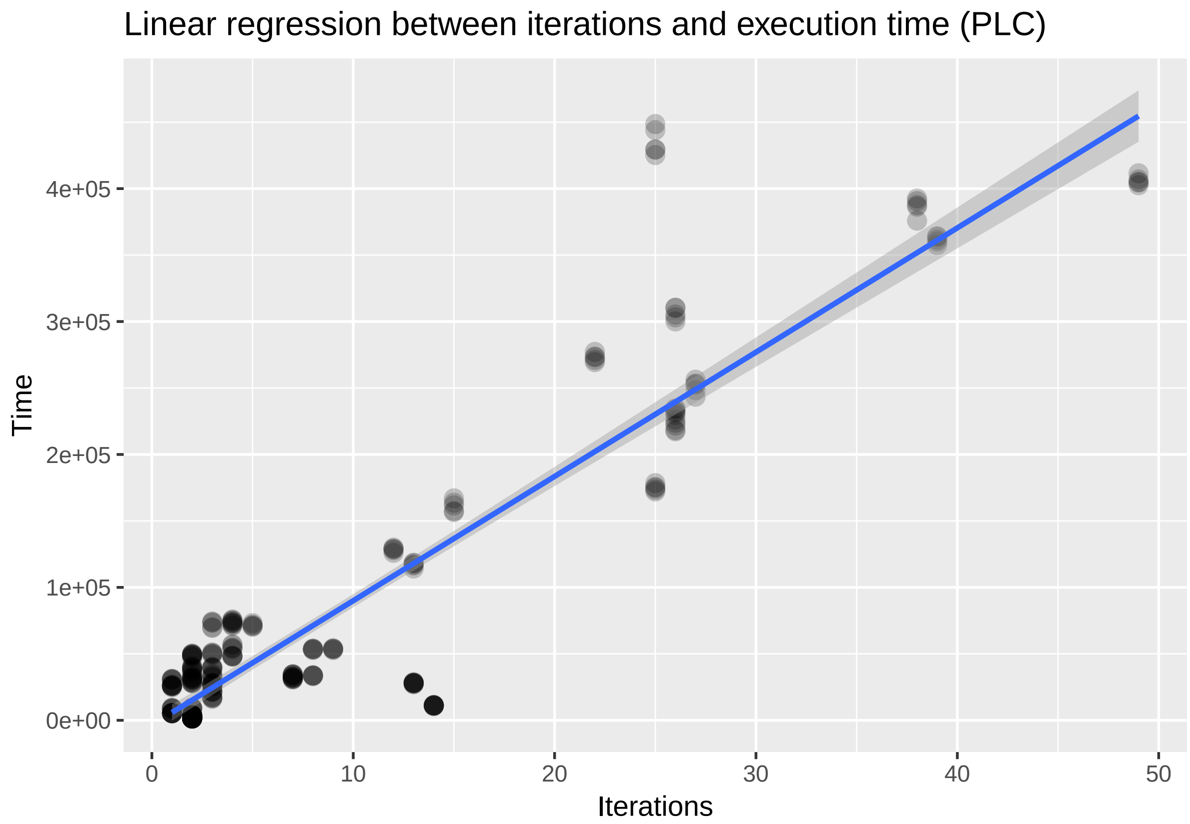
```
# Variables ~ Size (hardware)
ggplot(d.corr.hw, aes(Vars, Size)) +
  geom_point(alpha = 1/5, size = 3) +
  labs(title = "Linear regression between variables and size (hardware)",
       x = "Variables", y = "Size") +
  geom_smooth(method = "lm")
```




```r
summary(lm(d.corr.hw$Size ~ d.corr.hw$Vars))
```

```
##
## Call:
## lm(formula = d.corr.hw$Size ~ d.corr.hw$Vars)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -79.18 -38.51 -24.69 -15.58 244.04
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)    -56.50646    5.04540   -11.2   <2e-16 ***
## d.corr.hw$Vars  19.12534    0.02152   888.8   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 85.07 on 1198 degrees of freedom
## Multiple R-squared:  0.9985, Adjusted R-squared:  0.9985
## F-statistic: 7.9e+05 on 1 and 1198 DF,  p-value: < 2.2e-16
```

```r
# Iterations ~ Time (PLC)
d.corr.plc.no.to <- filter(d.corr.plc, !is.na(TimeMs))
ggplot(d.corr.plc.no.to, aes(Iterations, TimeMs)) +
  geom_point(alpha = 1/5, size = 3) +
  labs(title = "Linear regression between iterations and execution time (PLC)",
       x = "Iterations", y = "Time") +
  geom_smooth(method = "lm")
```




```
summary(lm(d.corr.plc.no.to$TimeMs ~ d.corr.plc.no.to$Iterations))
```

```
##
## Call:
## lm(formula = d.corr.plc.no.to$TimeMs ~ d.corr.plc.no.to$Iterations)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -116675  -13763    -563   20493  218329
##
## Coefficients:
##                              Estimate Std. Error t value Pr(>|t|)
## (Intercept)                   -3509.5     3202.7  -1.096    0.274
## d.corr.plc.no.to$Iterations   9349.0      234.7  39.835   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 44440 on 313 degrees of freedom
## Multiple R-squared:  0.8352, Adjusted R-squared:  0.8347
## F-statistic:  1587 on 1 and 313 DF,  p-value: < 2.2e-16
```

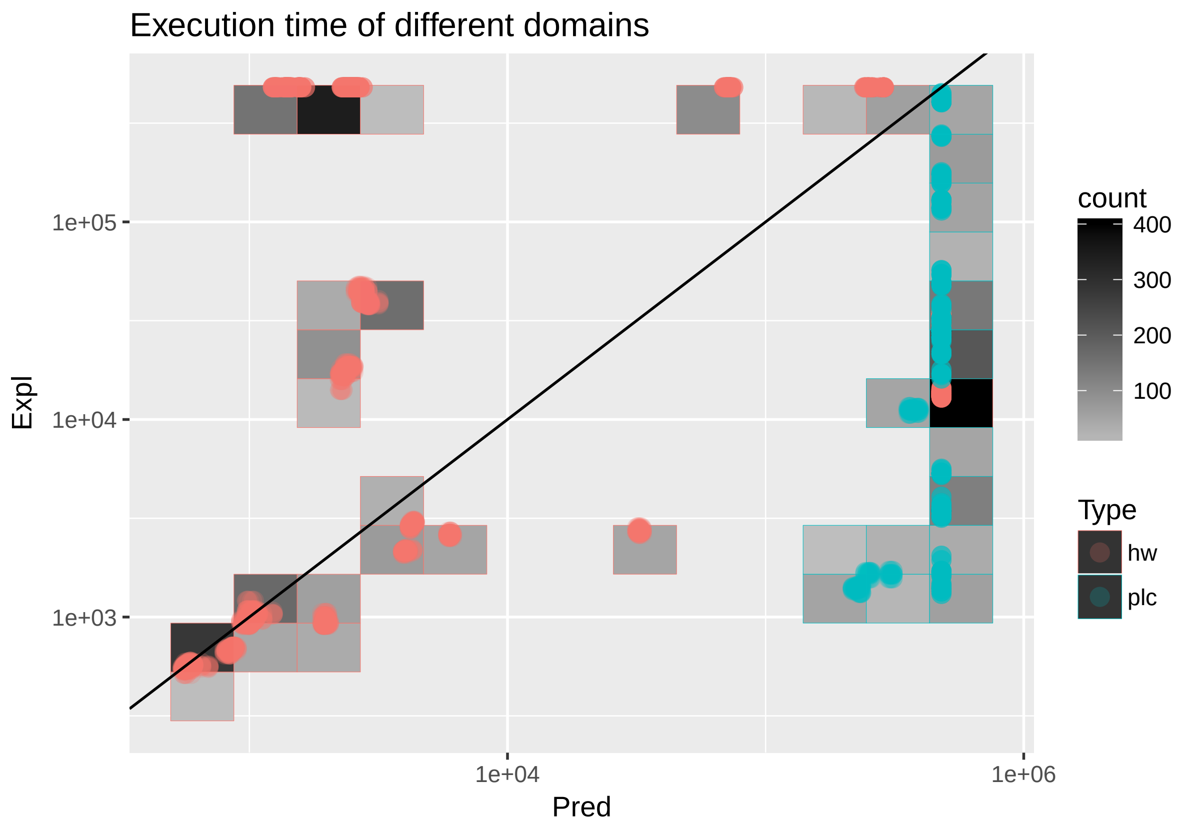
5 Pairwise comparisons

In the following, I analyze the effect of individual parameters of the algorithm. This is done by forming pairs from the measurements (using the join operation known from databases), where each parameter is the same, except the examined one. Then, a scatterplot is drawn, where the `x` and `y` values represent the two possible parameter values. Note, that this only works for parameters with 2 factors, but in my case, most of the parameters fulfill this constraint (e.g., the domain can be predicate or explicit).

5.1 Comparison of execution time for predicate and explicit domains

Each point in the following plot represents two executions for the same model: one with predicate domain and one with explicit domain. A point above the line means that predicate domain was faster, while a point below the line means the opposite. Points at the right and top edges of the plot correspond to timeouts. It can be seen that verification of PLC models is more efficient in the explicit domain. Hardware models show some diversity, each domain has good results. Some clusters can also be observed, therefore I also performed cluster analysis.

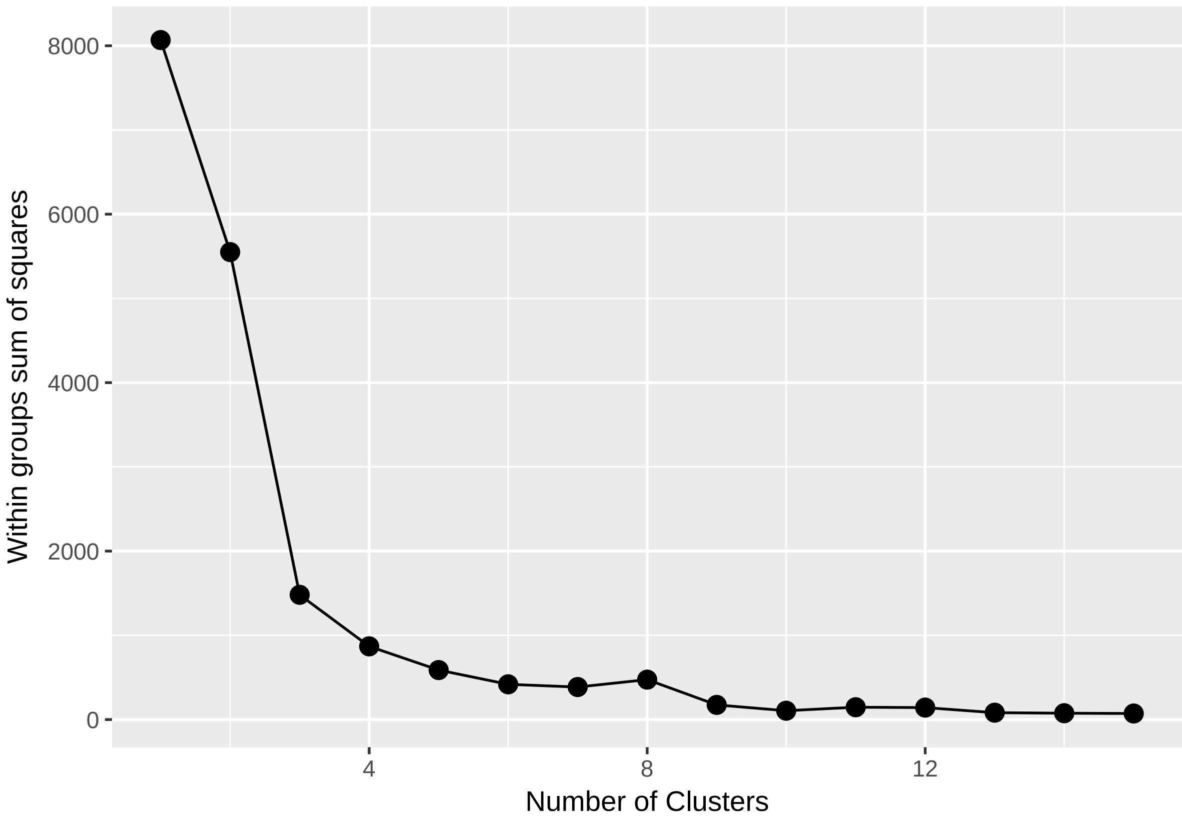
```
# Select only input variables and time
d.inputs.time <- select(d, Type, Model, Domain, Refinement, InitPrec, Search, TimeMs)
# Fill NAs with timeout value
d.inputs.time[is.na(d.inputs.time)] <- timeout.ms
# Filter for the different domains and join by the other columns
d.domain.time <- inner_join(
  filter(d.inputs.time, Domain == "PRED"),
  filter(d.inputs.time, Domain == "EXPL"),
  by = c("Type", "Model", "Refinement", "InitPrec", "Search"))
# Filter where both times are timeout
d.domain.time <- filter(d.domain.time,
                        TimeMs.x !=  timeout.ms | TimeMs.y !=  timeout.ms)

# Plot
ggplot(d.domain.time, aes(TimeMs.x, TimeMs.y, color = Type)) +
  scale_y_log10() + scale_x_log10() +
  geom_bin2d(bins = 12) +
  scale_fill_gradient(low = "gray", high = "black") +
  geom_point(alpha = 1/5, size = 3) +
  geom_abline() +
  labs(title = "Execution time of different domains", x = "Pred", y = "Expl")
```



5.1.1 Clustering

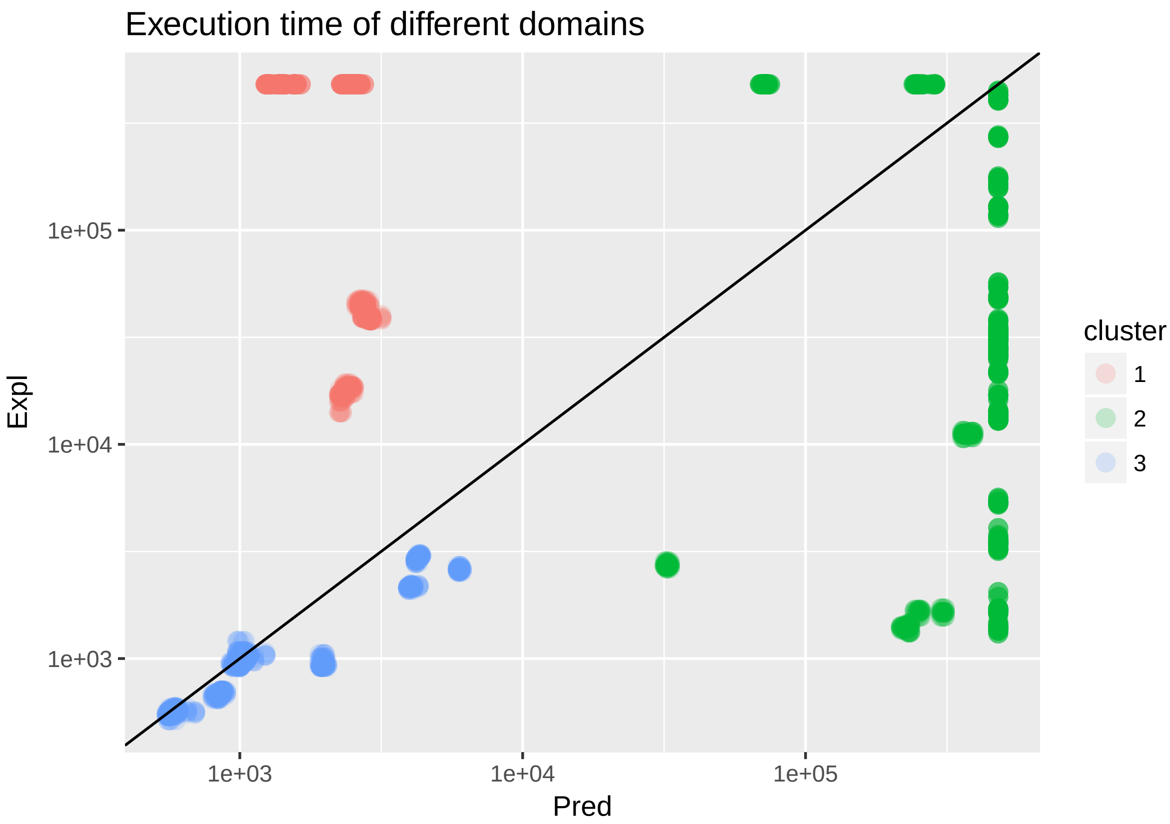
Usually, the first task in cluster analysis is to determine the number of clusters. The following plot shows that the increase in the quality of clustering slows after 3 clusters. Therefore, I ran a k-means cluster analysis with `k = 3`.

```
# Extract data for clustering
d.dom.time.clust <- log10(select(d.domain.time, TimeMs.x, TimeMs.y))
# Determine number of clusters
set.seed(1)
wss <- (nrow(d.dom.time.clust) - 1) * sum(apply(d.dom.time.clust, 2, var))
for (i in 2:15) wss[i] <- sum(kmeans(d.dom.time.clust, centers = i)$withinss)
ggplot(data.frame(x = 1:15, wss), aes(x, wss)) +
  geom_point(size = 3) + geom_line() +
  labs(x = "Number of Clusters", y = "Within groups sum of squares")
```



The following plot shows a convincing result.

```
# K-Means Cluster Analysis
fit <- kmeans(d.dom.time.clust, 3)
# Append clusters
d.dom.time.clust <- data.frame(d.domain.time, cluster = as.factor(fit$cluster))
# Plot
ggplot(d.dom.time.clust, aes(TimeMs.x, TimeMs.y, color = cluster)) +
  scale_y_log10() + scale_x_log10() +
  geom_point(alpha = 1/5, size = 3) +
  geom_abline() +
  labs(title = "Execution time of different domains", x = "Pred", y = "Expl")
```



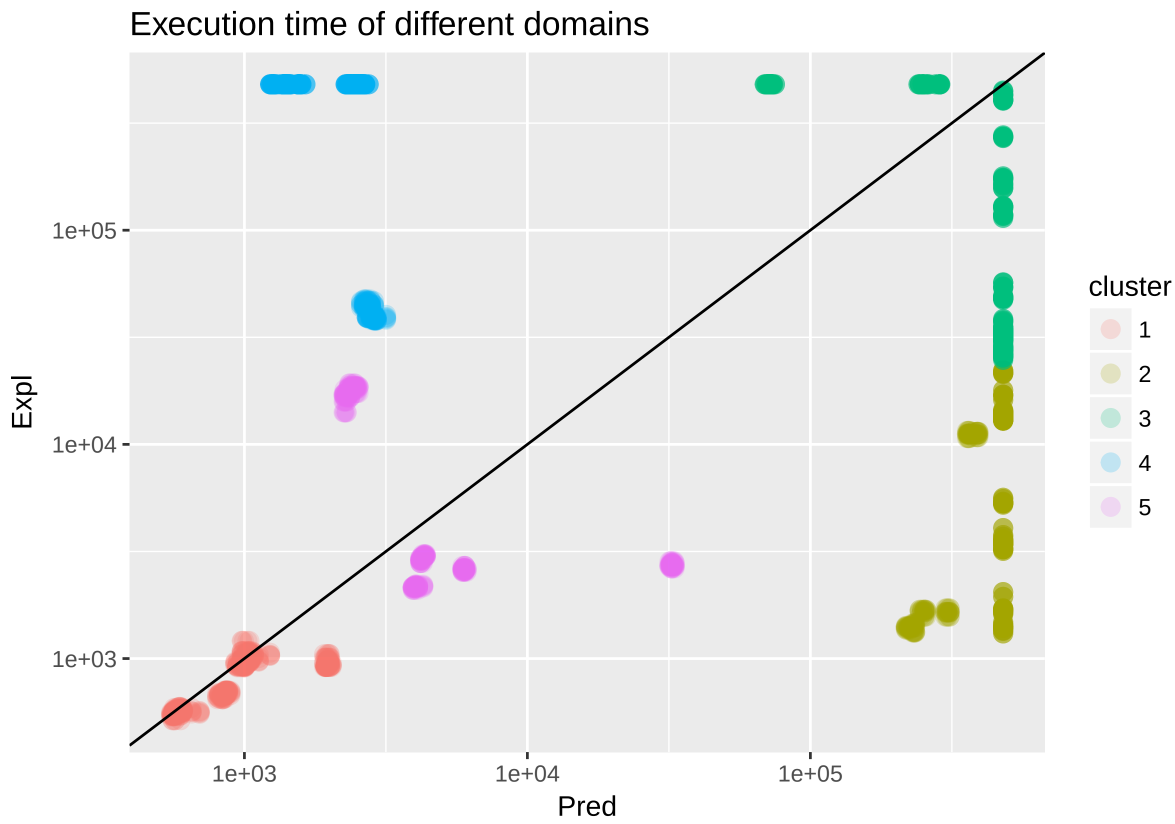
The table shows that all PLC models are in the same cluster (mixed with some hardware) and there are two clusters containing the majority of hardware models.

```
table(d.dom.time.clust$cluster, d.dom.time.clust$Type)
```

```
##
##       hw plc
##    1 800   0
##    2 825 975
##    3 750   0
```

As it can be seen below, running clustering with `k = 5` only yields a slightly better classification (there are a bit less hardware mixed with PLC).

```
d.dom.time.clust <- log10(select(d.domain.time, TimeMs.x, TimeMs.y))
# K-Means Cluster Analysis
fit <- kmeans(d.dom.time.clust, 5)
# Append clusters
d.dom.time.clust <- data.frame(d.domain.time, cluster = as.factor(fit$cluster))
# Plot
ggplot(d.dom.time.clust, aes(TimeMs.x, TimeMs.y, color = cluster)) +
  scale_y_log10() + scale_x_log10() +
  geom_point(alpha = 1/5, size = 3) +
  geom_abline() +
  labs(title = "Execution time of different domains", x = "Pred", y = "Expl")
```




```
table(d.dom.time.clust$cluster, d.dom.time.clust$Type)
```

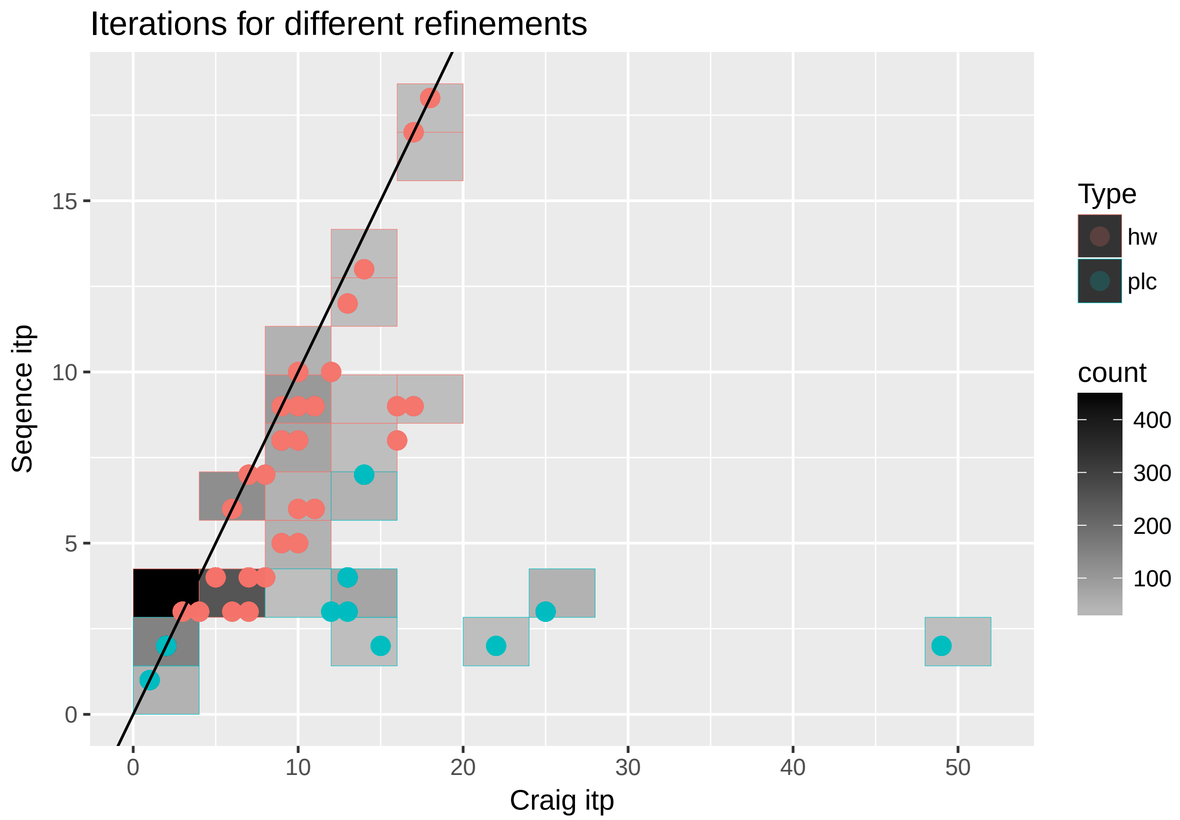
```
##
##      hw plc
##   1 600   0
##   2 400 475
##   3 375 500
##   4 700   0
##   5 300   0
```

5.2 Comparison of iterations for Craig and sequence itp refinements

Each point in the following plot represents two executions for the same model: one with Craig interpolation-based refinement and one with sequence interpolation. It can be observed, that for hardware models, sequence interpolation yields slightly less iterations. However, for PLC models, there are much less iterations with sequence interpolation.

```
# Select only input variables and iterations
d.inputs.iters <- select(d, Type, Model, Domain, Refinement,
                         InitPrec, Search, Iterations)
d.inputs.iters <- filter(d.inputs.iters, !is.na(Iterations))
# Filter for the different refinements and join by the other columns
d.refin.iters <- inner_join(
  filter(d.inputs.iters, Refinement == "CRAIG_ITP"),
  filter(d.inputs.iters, Refinement == "SEQ_ITP"),
  by = c("Type", "Model", "Domain", "InitPrec", "Search"))

# Plot
ggplot(d.refin.iters, aes(Iterations.x, Iterations.y, color = Type)) +
  geom_bin2d(bins = 12) +
  scale_fill_gradient(low = "gray", high = "black") +
```

```
  geom_point(alpha = 1/5, size = 3) +
  geom_abline() +
  labs(title = "Iterations for different refinements",
       x = "Craig itp", y = "Seqence itp")
```



The previous observation can also be confirmed with a Wilcoxon test: checking whether the number of iterations has identical distribution for different refinement strategies. The p-value is lower than 0.05, therefore the null hypothesis can be rejected, the distribution of the number of iterations is different for the refinement strategies.

```
wilcox.test(Iterations ~ Refinement ,data = filter(d.no.to, Refinement != "UNSAT_CORE"))
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  Iterations by Refinement
## W = 128550, p-value = 9.935e-09
## alternative hypothesis: true location shift is not equal to 0
```

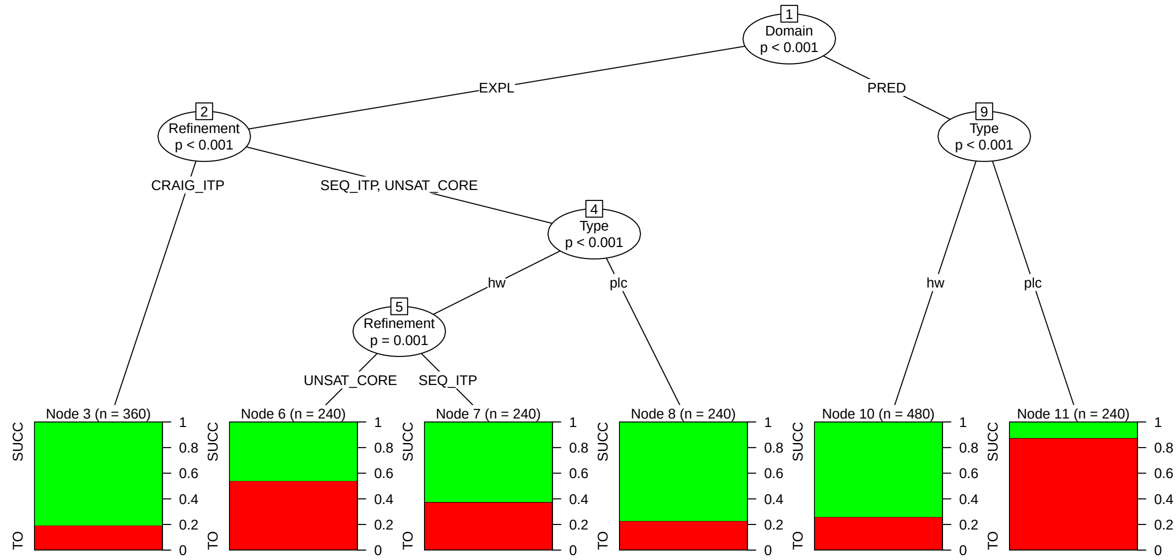
6 Decision tree

I also created some decision trees to check which input parameters influence certain output parameters.

6.1 Success of verification

Most of the leaves of the following tree categorize successful (SUCC) and timeout (TO) executions quite well. For example, choosing predicate abstraction for PLCs will most likely not succeed. On the other hand, it is likely to succeed for hardware models. It can also be seen that explicit abstraction with Craig interpolation refinement is likely to succeed regardless of the model type.

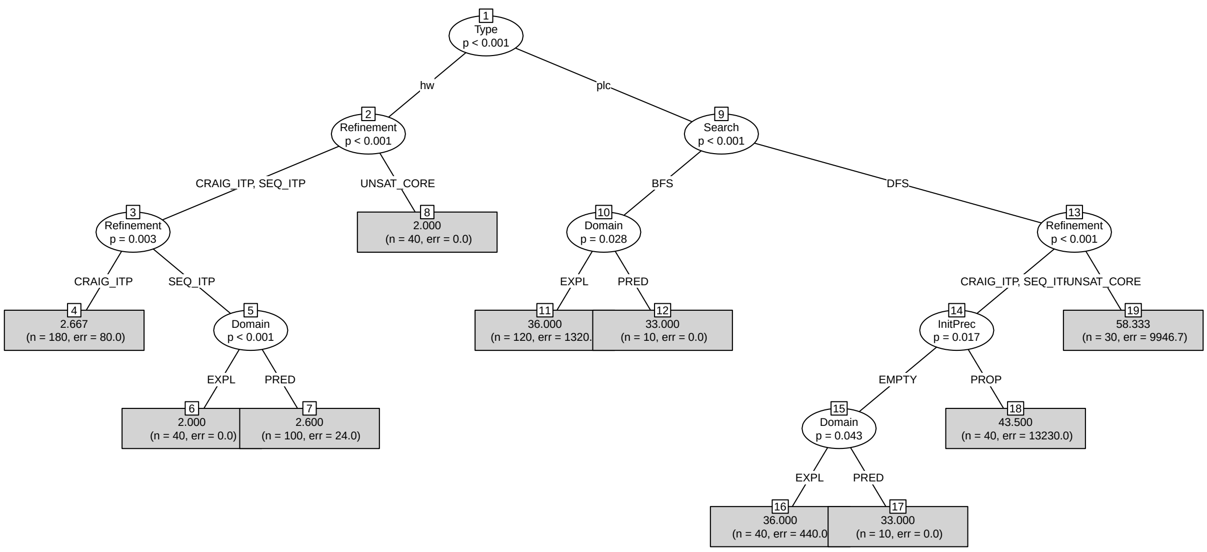
```
# Create success column
d.succ <- data.frame(d, Success = ifelse(!is.na(d$TimeMs), "SUCC", "TO") )
# Generate tree
tree <- ctree(Success ~ Type + Domain + Refinement + InitPrec + Search, data = d.succ)
plot(tree, gp = gpar(fontsize = 10), tp_args = list(fill = c("red", "green")))
```



6.2 Counterexample length

I also checked if the search strategy (BFS or DFS) has an effect on the length of the counterexample. However, as the following tree shows, for hardware models, counterexamples are short regardless of the search strategy and there is no significant difference for PLCs as well: with the other parameters adjusted, DFS can also produce shorter counterexamples.

```
# Filter for unsafe executions
d.cex <- d %>% filter(!is.na(CEXlen))
# Generate tree
tree <- ctree(CEXlen ~ Type + Domain + Refinement + InitPrec + Search, data = d.cex)
plot(tree, type = "simple", gp = gpar(fontsize = 8))
```



The previous observation can be checked by a Wilcoxon test: checking whether the counterexample length has identical distribution for different search strategies.

```
wilcox.test(CEXlen ~ Search, data = d.cex)
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  CEXlen by Search
## W = 46050, p-value = 0.8313
## alternative hypothesis: true location shift is not equal to 0
```

The p-value is large, which means that there is no strong evidence stating that DFS and BFS would yield counterexamples with different length.

7 Conclusion

In my homework I analyzed different configurations of a formal verification algorithm on various models. I checked basic properties of the configurations, examined correlations, performed pairwise comparisons and clustering, and I also generated decision trees. The results showed some interesting properties of the algorithm configurations. In the future I am planning to optimize my algorithms so that I can successfully run them on a larger number of models, yielding more robust results.