Bill J. Ellis
June 9, 2005

SPADEase: The Good, the Bad and the Ugly

Contents

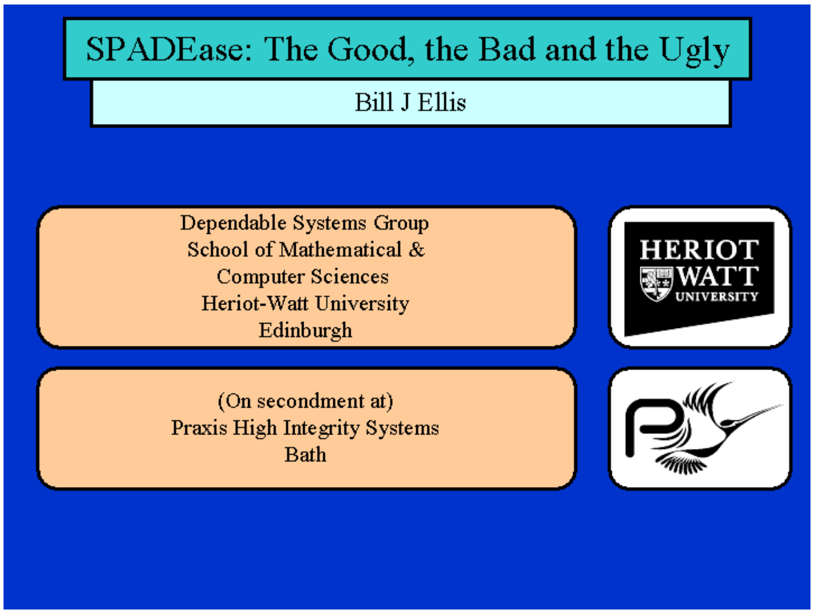
*Hums are short notes intended for distribution between those involved with EPSRC grant GR/T12289/01. Hums describe ϵ-baked ideas, where $1 \geq \epsilon \geq 0$. (Hum refers to both the Praxis Humming bird and Winnie-the-Pooh's indirect approach to writing: "Poetry and Hums aren't things which you get, they're things which get you.").

1 Introduction

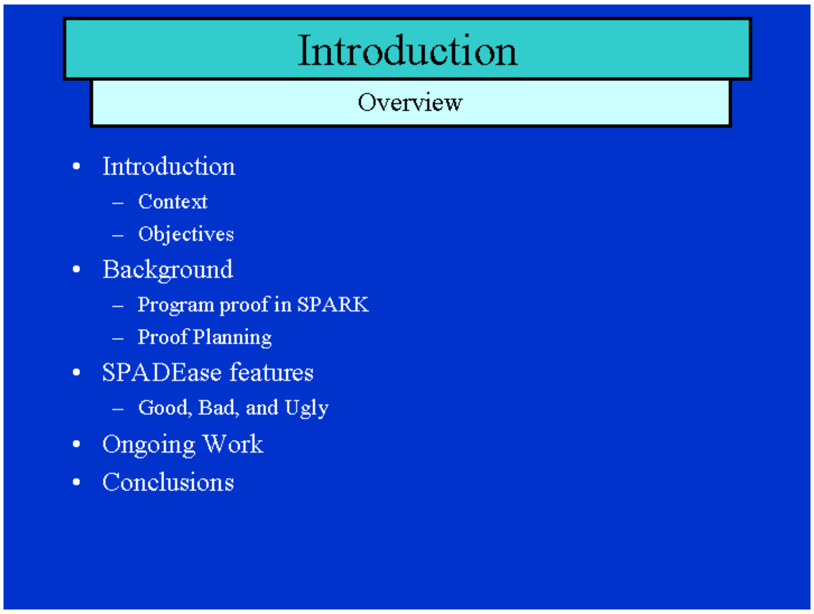
Approaching the closing five weeks of the SPADEase project, Andrew Ireland is visiting Praxis High Integrity Systems to see what has been achieved so far and guide activity in these closing weeks. To get Andrew, and Praxis, up to speed to on the project, a lunchtime seminar is being held. Here the slides and some notes from this seminar are collected.

2 Presentation

2.1 Slide 1



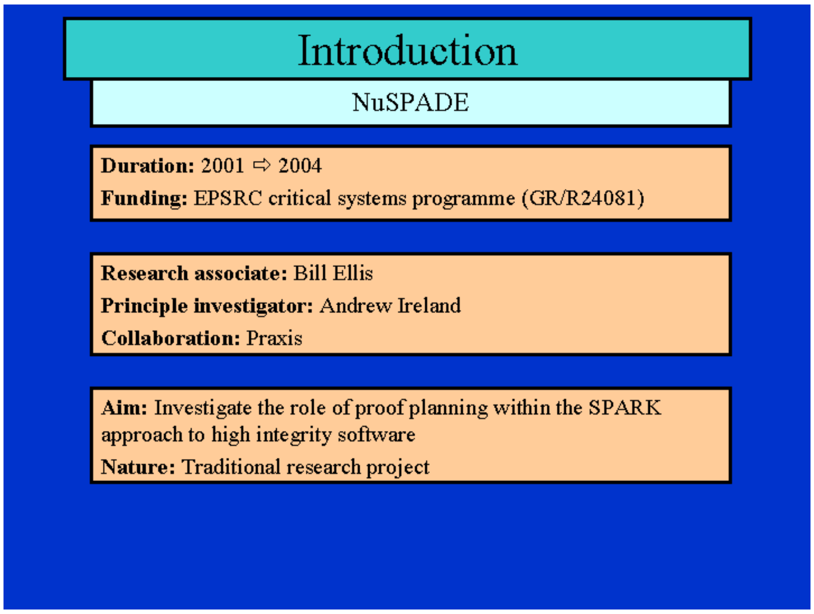
2.2 Slide 2



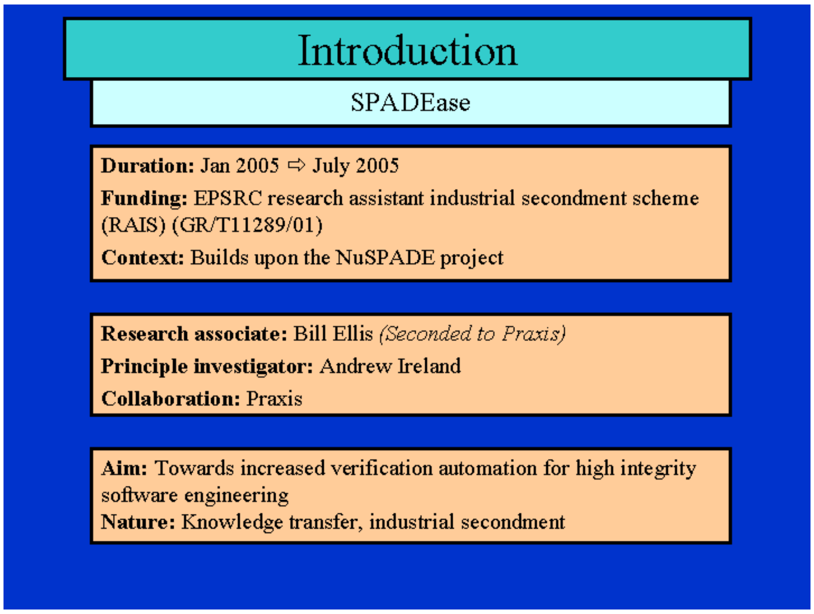
- Begin by explaining the context for this project.
- Outline the key objectives of this project.
  - And how we plan to meet these.
- Give high level background to:

  - Program proof in SPARK.
  - Proof Planning.
- Describe key features of the new system SPADEase:
  - Categorising these as good, bad or ugly.
- Outline work for the remainder of the project.
- Finish with some conclusions.

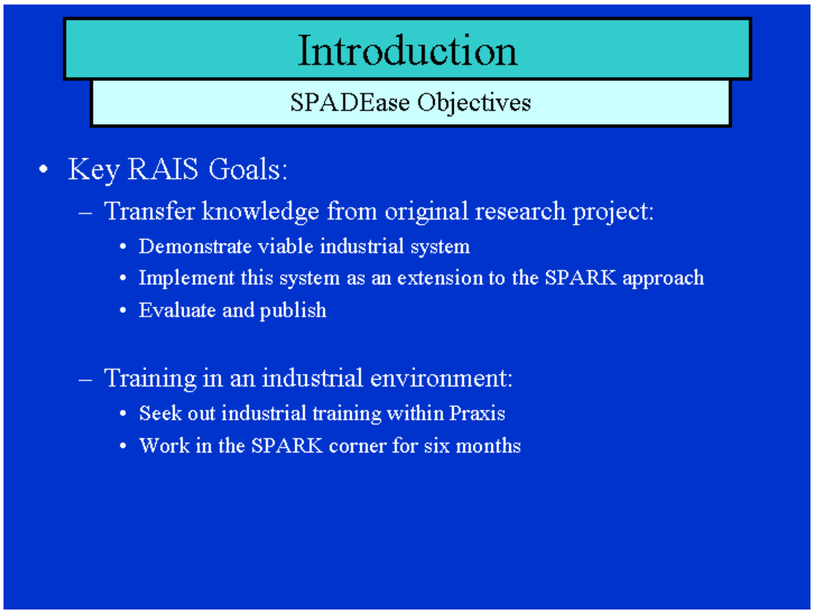
2.3 Slide 3



2.4 Slide 4

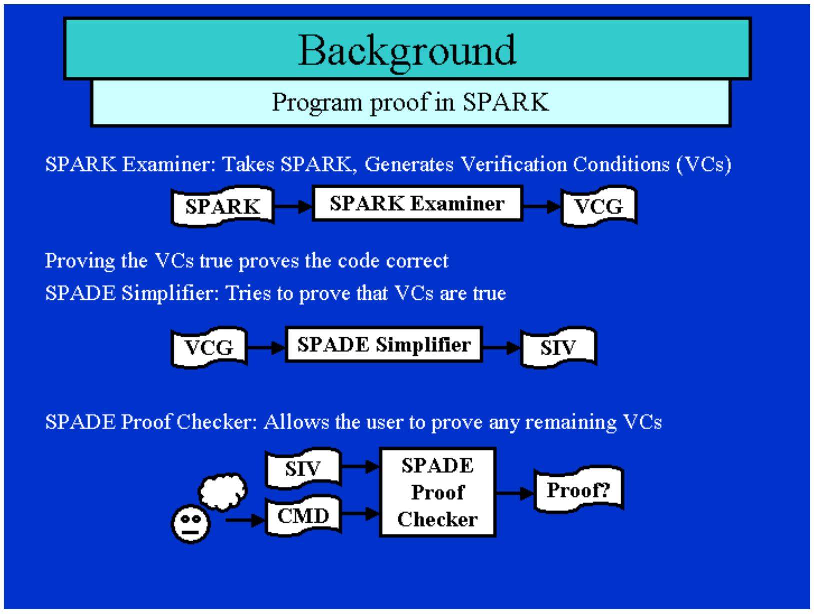


2.5 Slide 5



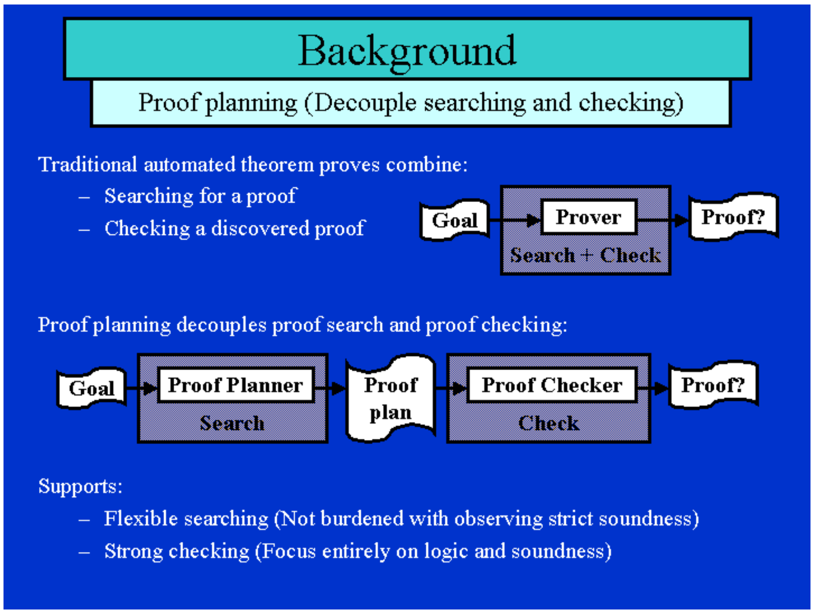
- There are two key goals that every RAIS project should try to reach.
- The first is to transfer knowledge from the original research project.
  - Demonstrate a viable industrial system.
    - To show this knowledge is useful.
  - Implement this system as an extension to the existing SPARK tools.
    - To show this knowledge is directly applicable.
  - Evaluate and publish.
    - To provide guidance in applying this knowledge.
- The second RAIS goal is to train the research associate in an industrial environment.
  - Seek out industrial training within Praxis.
    - Attend any courses, or Praxis meetings.
  - Primarily, just work in the SPARK corner for six months.

2.6 Slide 6



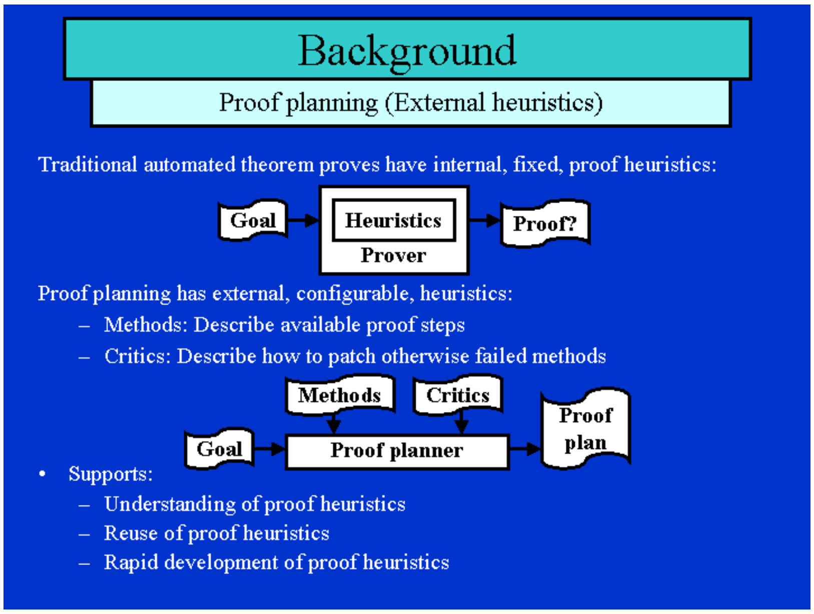
- SPARK Examiner takes as input a SPARK program.
  - Generates Verification Conditions (VCs) in a VCG file.
  - Proving every VC is true proves the code is correct.
- SPADE Simplifier takes these VCs as input.
  - Tries to automatically prove them as true.
  - And returns any it can not prove in an SIV file.
- SPADE Proof Checker supports the user to prove any remaining VCs.
  - The user interacts with the Proof Checker.
  - To construct a proof that convinces the Proof Checker that the VC is true.

2.7 Slide 7



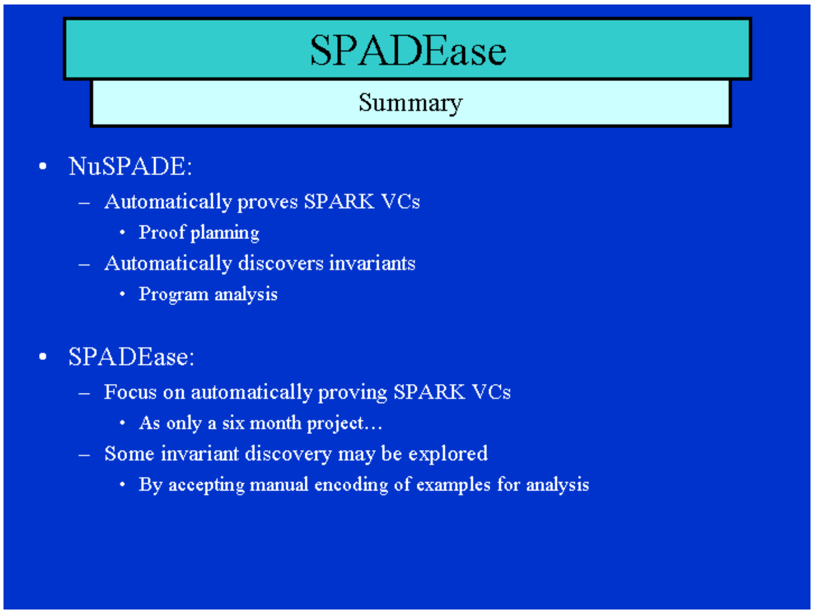
- Traditional automated theorem provers interleave the task of:
  - Searching for a proof.
  - Checking a discovered proof.
- Proof planning decouples these separate tasks:
  - Having an explicit searching component.
  - And an explicit proof checking component.
- This supports more flexible searching:
  - As the search is not burdened with demonstrating strict soundness.
- This supports stronger proof checking:
  - As the checking focuses entirely on logic and soundness.

2.8 Slide 8

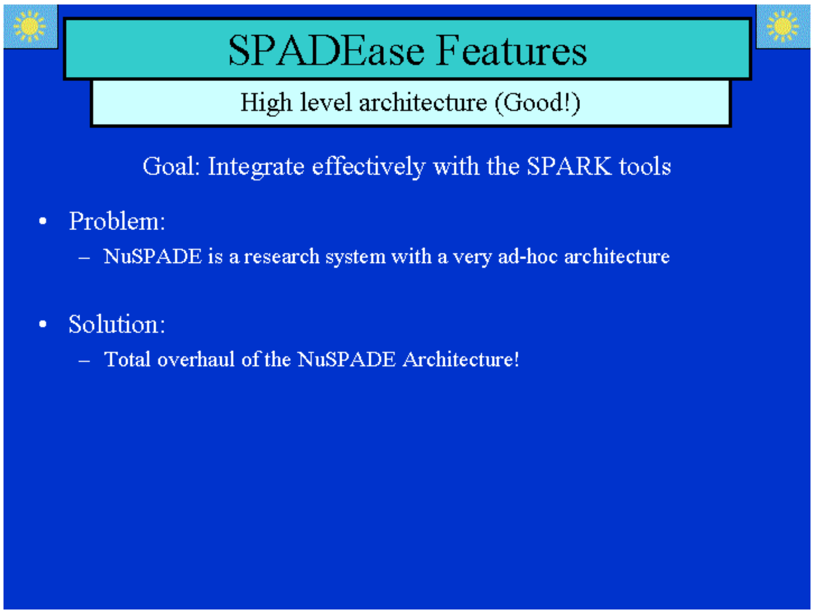


- Traditional automated theorem proves have internal, fixed, proof heuristics.
- While proof planning has external, configurable, heuristics.
  - Methods describe available proof steps.
  - Critics are expressed in the same manner as methods.
  - Only they describe how to patch the failure of methods.
  - Both are described using a high level proof description language.
- Doing this supports understanding of proof strategies.
  - As every heuristic is expressed in the same style.
  - Using a high level description language.
- Further, this eases the reuse of proof heuristics.
  - As presenting the heuristics as external components reduces them to their essence.
  - making them less implementation dependent.
  - And much more straight forward to transfer across different systems.
- And supports the rapid development of proof heuristics.

2.9 Slide 9

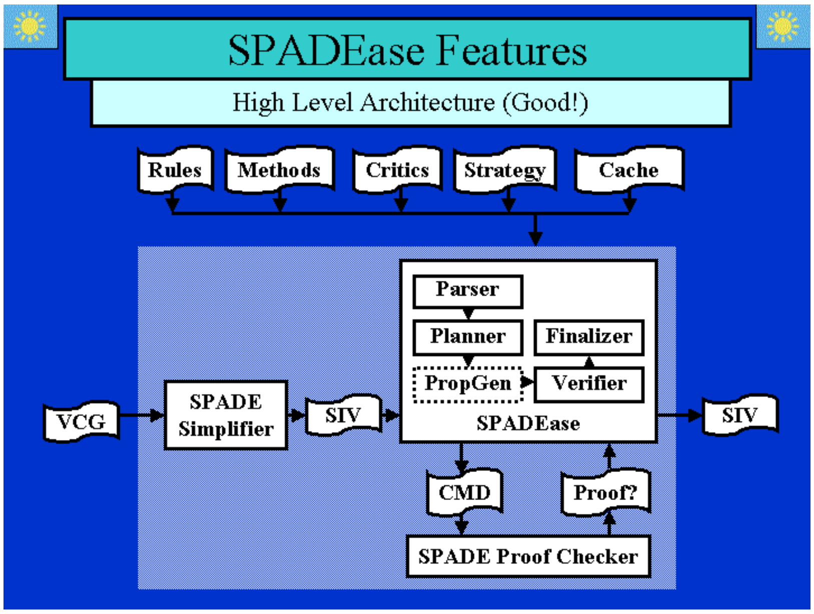


2.10 Slide 10



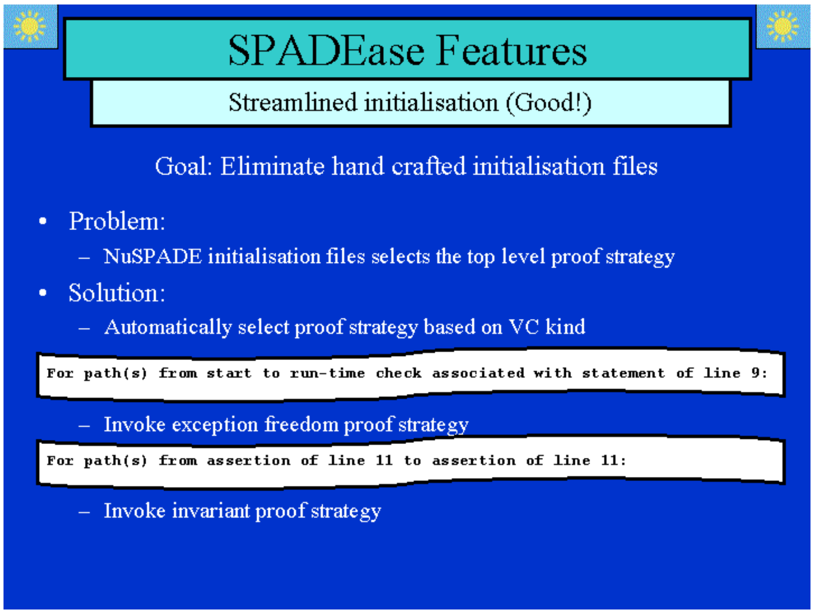
- Want to integrate effectively with the SPARK tools.
- However.
  - NuSPADE is a research system.
  - And has a very ad-hoc architecture.
- Solution.
  - Create a brand new architecture for SPADEase.

2.11 Slide 11



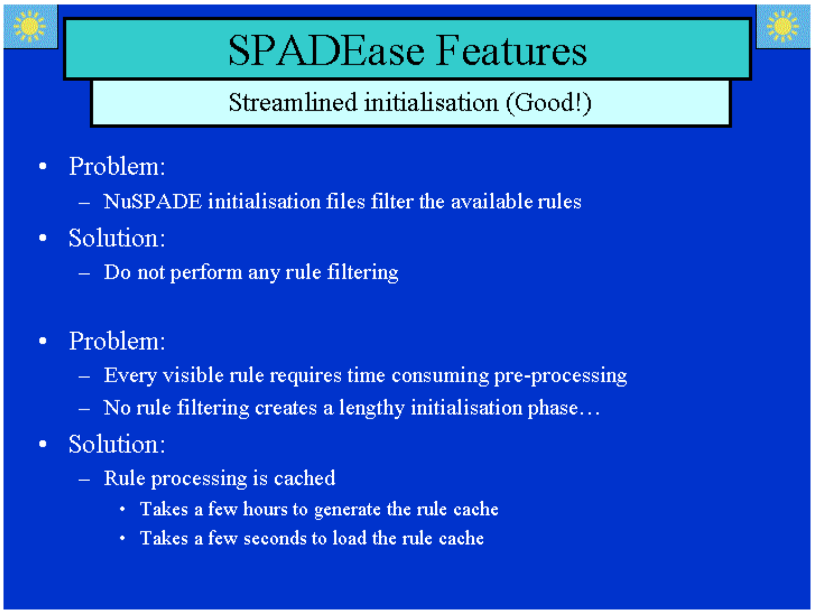
- This is the architecture for SPADEase.
- Note how SPADEase performs the same operation as the Simplifier.
  - A VCG file is provided as input.
  - And an SIV is provided as output.
- To begin, the Simplifier is called to prove as many VCs as possible.
- The core of SPADEase is then applied to any remaining VCs.
- SPADEase comprises of four discrete interacting components.
- Begins with the Parser.
  - Parse and process all data required for planning.
  - Includes all the rules. Built in rules, and RLS, and RUL rule.
  - All planning components: Methods, Critics and a Strategy for controlling these.
  - And a data cache.
- The planner takes the result from the parser.
  - And plans every remaining VC.
- In a complete version of SPADEase program analysis would be performed by PropGen.
  - This would try to discover program invariants in reaction to failed plans.
  - At best, a limited version of PropGen will be seen in SPADEase.
- The verifier takes the result from the planner.
  - Every discovered plan is converted into a CMD log.
  - And checked within the SPADE Proof checker.
- The finaliser inspects the results of the Proof Checker.
  - And generates the final SIV file.

2.12 Slide 12



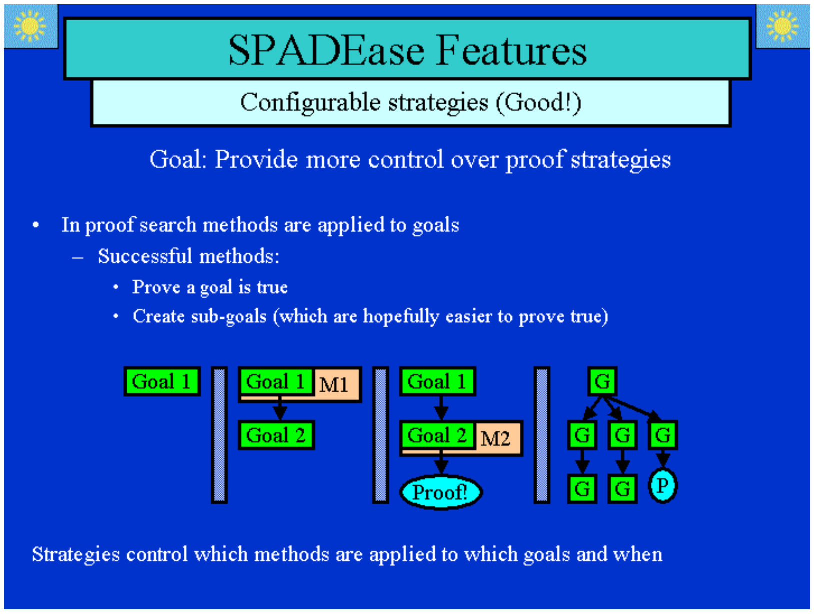
- Want to eliminate hand crafted initialisation files.
  - As these prevent the automatic application of SPADEase.
- A Problem.
  - NuSPADE requires an initialisation file to select the top level proof strategy.
- Solution.
  - Automatically select the proof strategy based on VC kind.
- So, for example, if the VC is declared as being:
  - For path(s) from start to run-time check associated with statement of line 9:
  - Then it is a run-time VC.
  - And the top level exception freedom strategy is executed.
- And if the VC is declared as being:
  - For path(s) from assertion of line 11 to assertion of line 11:
  - Then the invariant proof strategy is executed.

2.13 Slide 13

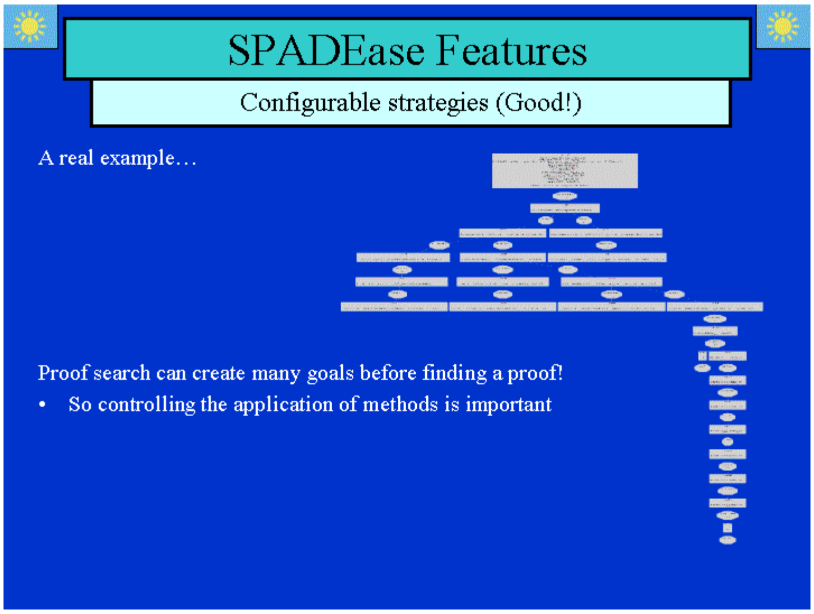


- Another problem.
  - The initialisation file is also used to filter the available rules.
- Solution.
  - Is straight forward.
  - Do not perform any rule filtering.
- However, this creates a new problem.
  - Every visible rule requires time consuming pre-processing.
- Fortunately, this processing always returns the same results for the same rule.
- Solution.
  - Cache all of this rule processing.
  - Although it takes a few hours to generate the rule cache.
  - It only takes a few seconds to load the rule cache.

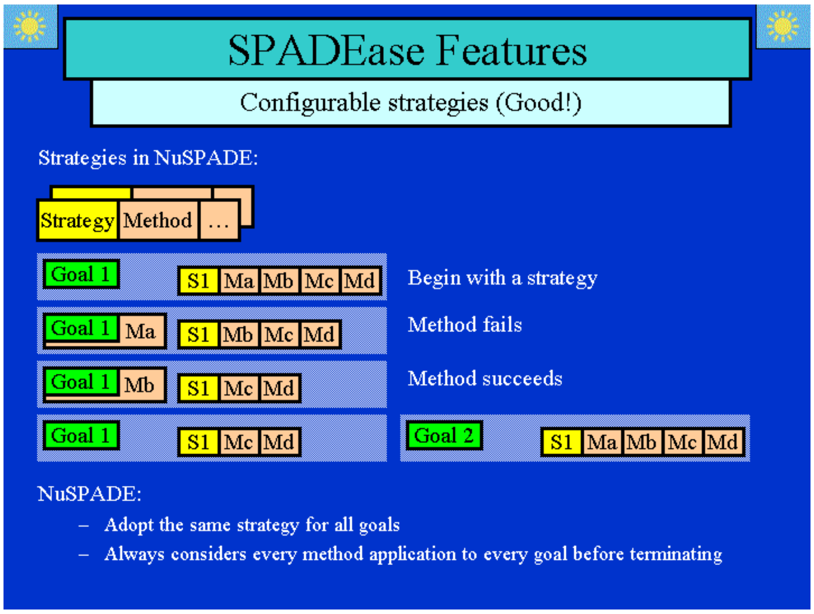
2.14 Slide 14



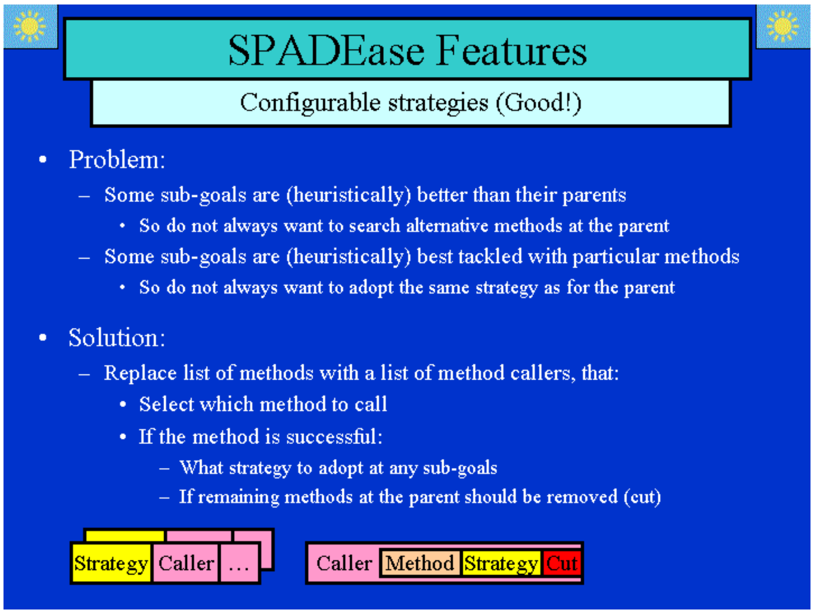
2.15 Slide 15



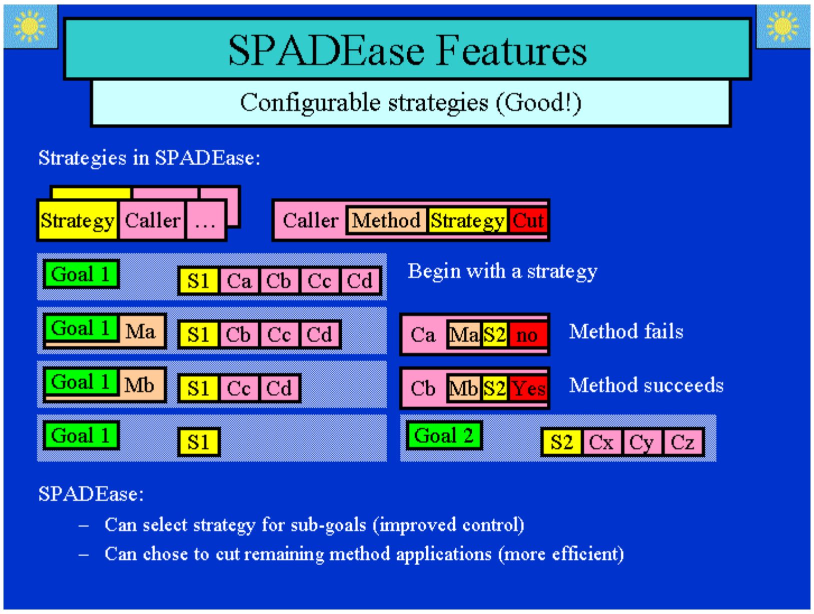
2.16 Slide 16



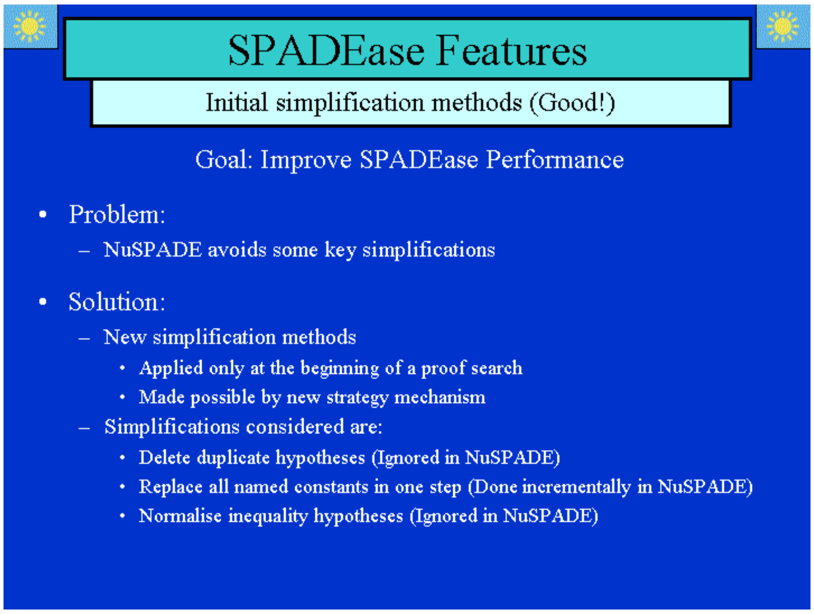
2.17 Slide 17



2.18 Slide 18

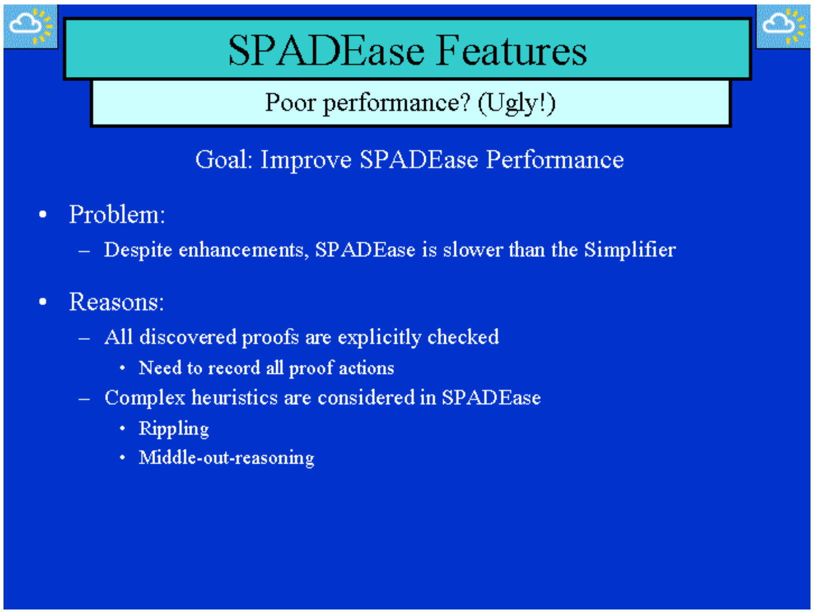


2.19 Slide 19

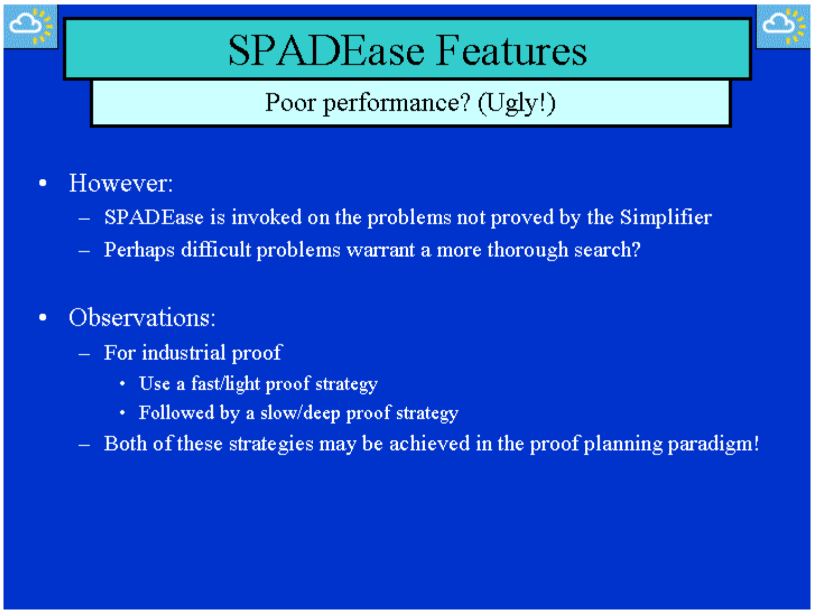


- Want to improve SPADEase Performance.
- Is particularly important to be industrially applicable.
- NuSPADE avoids some key simplifications.
  - Because this was less important in a research system.
  - And difficult to employ effectively using the NuSPADE strategy mechanism.
- Solution is new start up simplification methods.
- These apply only once, just after a goal is loaded.
  - Delete double hypotheses.
  - Which brings an obvious efficiency boost.
  - All named constants are replaced with their values in a single method call.
  - NuSPADE performed this incrementally via several method calls.
  - Which is much slower.
  - Place all inequality hypotheses into a standard form.
  - Eliminating the need for search within methods that process inequalities.

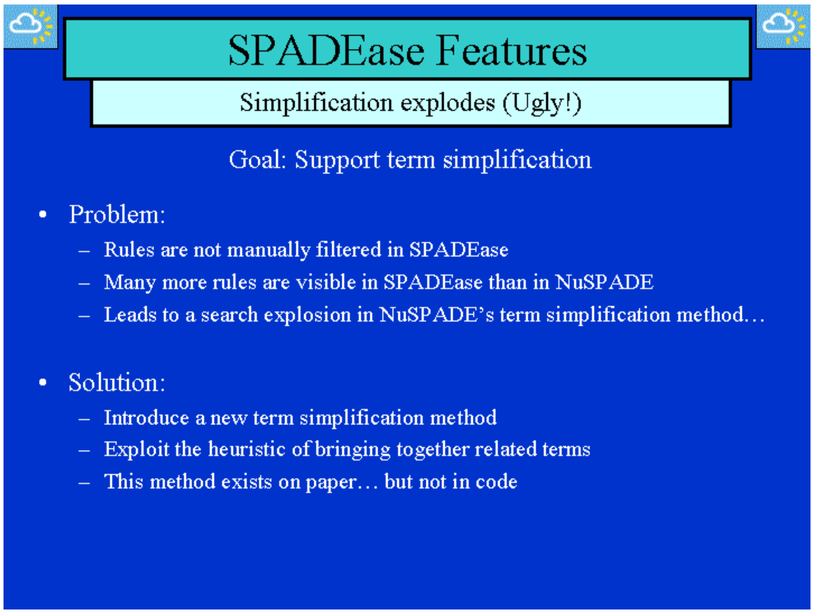
2.20 Slide 20



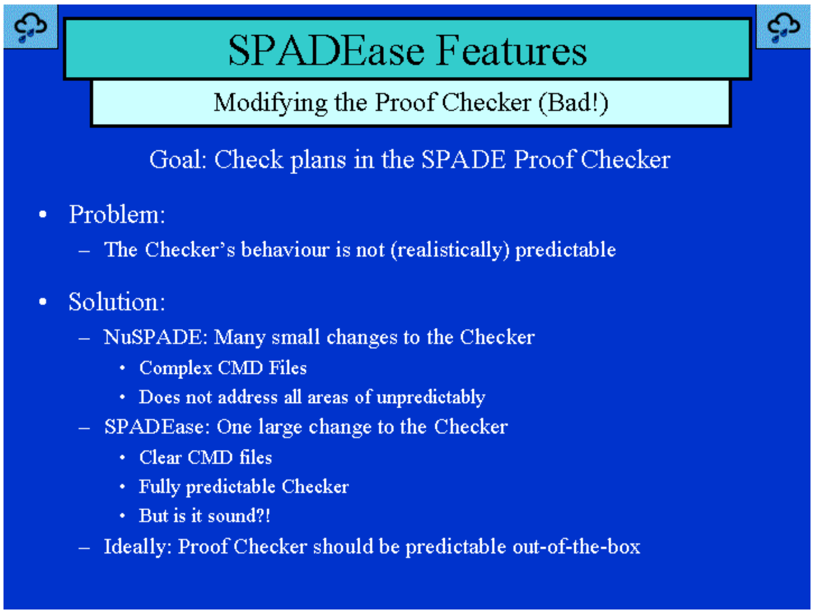
2.21 Slide 21



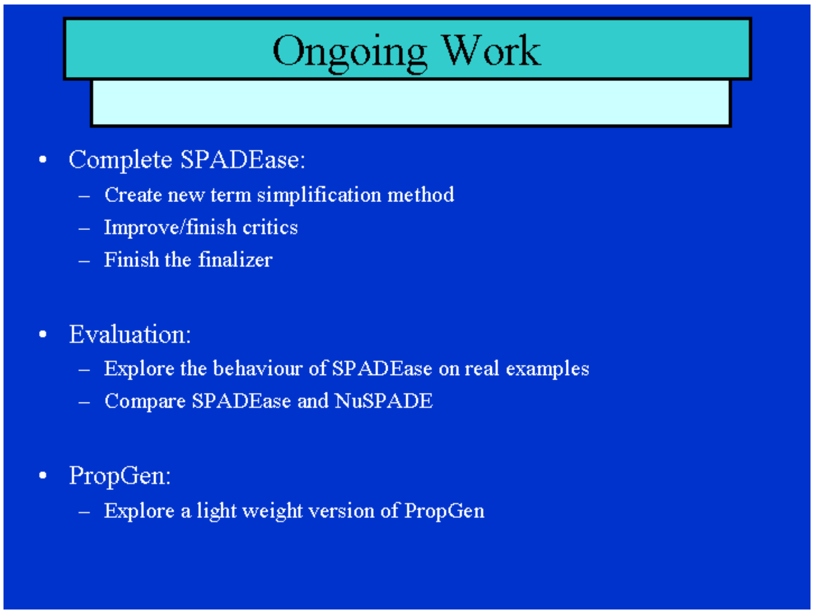
2.22 Slide 22



2.23 Slide 23



2.24 Slide 24



2.25 Slide 25