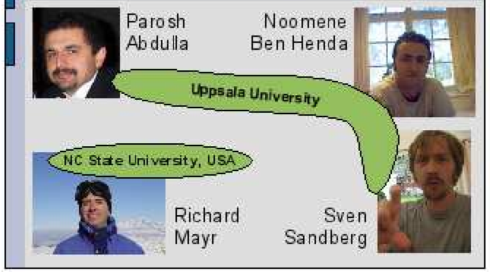
The Good, the Bad, and the Random

Stochastic Games with Lossy Channels



Overview

- Background
- Model
- Algorithm
- Conclusion

Overview

- Background
- Model
- Algorithm
- Conclusion

Context

- **Reactive System**
  - Never terminates
  - Input & output during execution
- **Program Verification**
  - Communication protocols
  - Hardware

One approach to verification:

- **Model checking**
  - Input: model + formula
  - Output: "yes, formula true" / "no, formula false"

Context

- **Reactive System**
  - Never terminates
  - Input & output during execution
- **Program Verification**
  - Communication protocols
  - Hardware

One approach to verification:

- **Model checking**
  - Input: model + formula
  - Output: "yes, formula true" / "no, formula false"

Stochastic Infinite-State Games

- **Stochastic**
  - Unreliable communication
  - Fault-Tolerant systems
  - Randomized algorithms
- **Infinite-State**
  - Unbounded queues, stacks, counters, ...
  - Time
  - Unbounded parallelism
- **Games**
  - Model interacts with environment
  - Alternation in logics
  - Abstraction

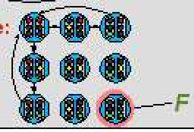
Stochastic Infinite-State Games

- **Stochastic**
  - Unreliable communication
  - Fault-Tolerant systems
  - Randomized algorithms
- **Infinite-State**
  - Unbounded queues, stacks, counters, ...
  - Time
  - Unbounded parallelism
- **Games**
  - Model interacts with environment
  - Alternation in logics
  - Abstraction

Reachability

- *F* – set of "target states"
- *F* reached?

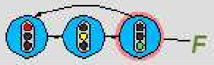
- **Example:**



Repeated Reachability

- *F* – set of "target states"
- *F* reached infinitely often?

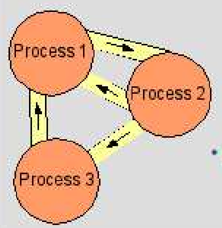
- **Example:**
  - Will it always return to the green state?



Overview

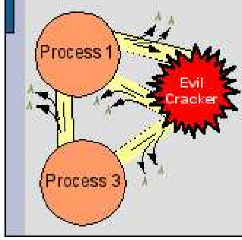
- Background
- Model
- Algorithm
- Conclusion

Lossy Channel Systems (LCS)



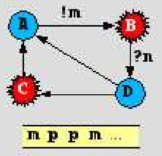
- **Model:**
  - Finite state processes
  - Unbounded lossy channels
  - Send & receive operations
- **Motivation:**
  - Models of communication protocols

Game Probabilistic Lossy Channel Systems (GPLCS)



- **Game:**
  Interaction with evil cracker
- **Probabilistic:**
  Messages lost randomly (probability λ)

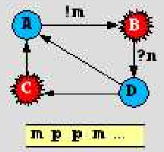
Game Probabilistic Lossy Channel Systems (GPLCS)



Sufficient:
- One channel
- One process
- Each state controlled by a player

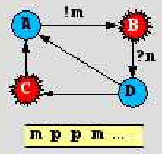
Properties:
- Infinite state space
- Perfect channel = Turing machine

Game Probabilistic Lossy Channel Systems (GPLCS)

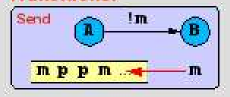


State: s = (A, mpnn)

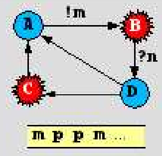
Game Probabilistic Lossy Channel Systems (GPLCS)



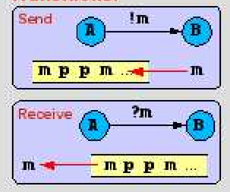
Transitions:



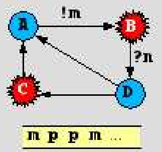
Game Probabilistic Lossy Channel Systems (GPLCS)



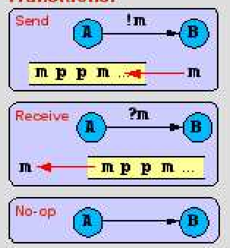
Transitions:



Game Probabilistic Lossy Channel Systems (GPLCS)

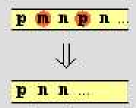


Transitions:



Game Probabilistic Lossy Channel Systems (GPLCS)

Probabilistic message loss:



- **Each transition:**

  each message lost with prob $\lambda > 0$, independently

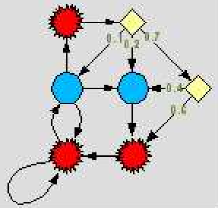
Stochastic Games

- **Every GPLCS induces a stochastic game**
- **3 types of states:**

Player Good

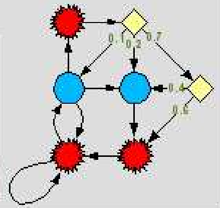
Player Bad

Player Random



Stochastic Games

- **Every GPLCS induces a stochastic game**
- **3 types of states:**



Stochastic Games

- **Strategy**

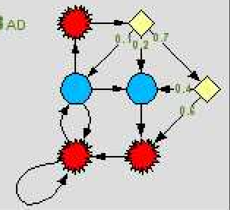
  = selection of outgoing transitions
- **Strategies for Good & Bad**

  ⇓

  Only probabilistic choices remain

  ⇓

  "**Prob(event)**" well-defined



Stochastic Games

- **Strategy**

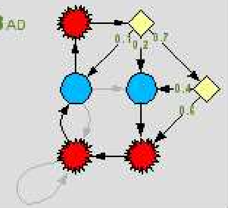
  = selection of outgoing transitions
- **Strategies for Good & Bad**

  ⇓

  Only probabilistic choices remain

  ⇓

  "**Prob(event)**" well-defined



Overview

- **Background**
- **Models**
- **Algorithm**
- **Conclusion**

Repeated Reachability for GPLCS

- **Input:**
  - GPLCS
  - Set F of final states
- **Output: Partition of states:**

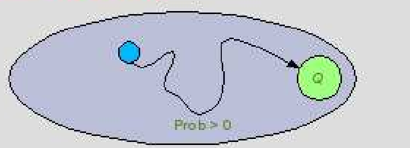
Winning for Bad

Winning for Good

Background Model Algorithm Conclusion

Subroutine: Force-set

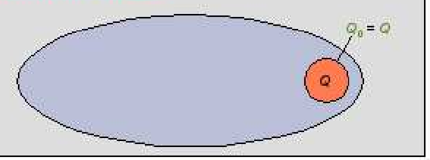
- Force-set ≈ “Reachability for games”
  - Given target set Q
  - Compute the set of states where GOOD can force Prob(reach Q) > 0
- Backward search



Background Model Algorithm Conclusion

Subroutine: Force-set Construction

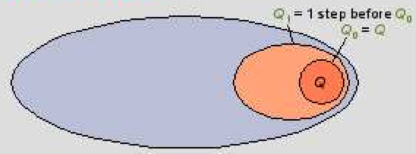
- Force-set ≈ “Reachability for games”
  - Given target set Q
  - Compute the set of states where GOOD can force Prob(reach Q) > 0
- Backward search



Background Model Algorithm Conclusion

Subroutine: Force-set Construction

- Force-set ≈ “Reachability for games”
  - Given target set Q
  - Compute the set of states where GOOD can force Prob(reach Q) > 0
- Backward search



Background Model Algorithm Conclusion

Subroutine: Force-set Construction

- Force-set ≈ “Reachability for games”
  - Given target set Q
  - Compute the set of states where GOOD can force Prob(reach Q) > 0
- Backward search

Background Model Algorithm Conclusion

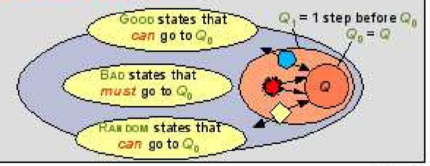
Subroutine: Force-set Construction

- Force-set ≈ “Reachability for games”
  - Given target set Q
  - Compute the set of states where GOOD can force Prob(reach Q) > 0
- Backward search

Background Model Algorithm Conclusion

Subroutine: Force-set Construction

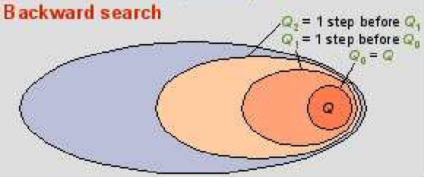
- Force-set ≈ “Reachability for games”
  - Given target set Q
  - Compute the set of states where GOOD can force Prob(reach Q) > 0
- Backward search



Background Model Algorithm Conclusion

Subroutine: Force-set Construction

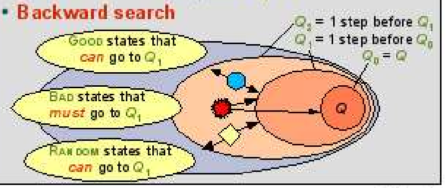
- Force-set ≈ “Reachability for games”
  - Given target set Q
  - Compute the set of states where GOOD can force Prob(reach Q) > 0
- Backward search



Background Model Algorithm Conclusion

Subroutine: Force-set Construction

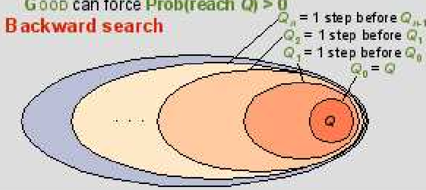
- Force-set ≈ “Reachability for games”
  - Given target set Q
  - Compute the set of states where GOOD can force Prob(reach Q) > 0
- Backward search



Background Model Algorithm Conclusion

Subroutine: Force-set Construction

- Force-set ≈ “Reachability for games”
  - Given target set Q
  - Compute the set of states where GOOD can force Prob(reach Q) > 0
- Backward search

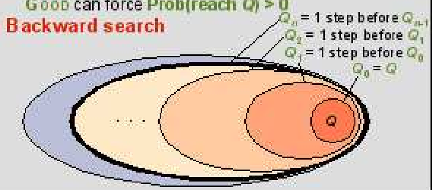


Background Model Algorithm Conclusion

Subroutine: Force-set Convergence

- Force-set … mes”
  - Given targ…
  - Compute the set of states where GOOD can force Prob(reach Q) > 0
- Backward search

Converges?



Background Model Algorithm Conclusion

Subroutine: Force-set Convergence

- Force-set … mes”
  - Given targ…
  - Compute the se… here
  GOOD can…
- Backward…

Converges?

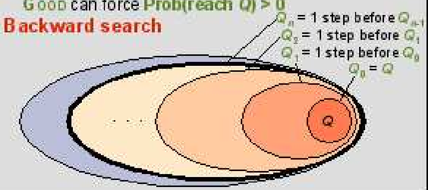
YES! for GPLCS (well quasi orders)

Background Model Algorithm Conclusion

Subroutine: Force-set Correctness

- Force-set … mes”
  - Given targ…
  - Compute the set of states where GOOD can force Prob(reach Q) > 0
- Backward search

Correct?



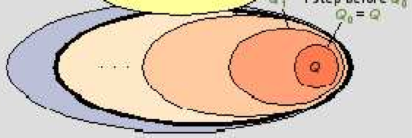
Background Model Algorithm Conclusion

Subroutine: Force-set Correctness

- Force-set "...mes"
  - Given targ...
  - Compute the se... ...here GOOD can ...
- Backward...

Correct?

YES! (and I'll show why)



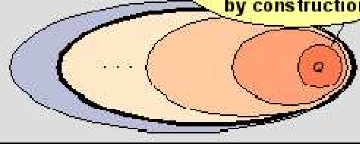
Background Model Algorithm Conclusion

Subroutine: Force-set Correctness

- Force-set "...mes"
  - Given targ...
  - Compute the set of states wher... GOOD can force Prob(re...
- Backward search

Correct?

In Q_n, GOOD can force Prob(reach Q) > 0 by construction



Background Model Algorithm Conclusion

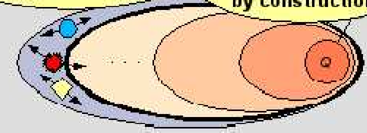
Subroutine: Force-set Correctness

- Force-set "...mes"
  - Given targ...
  - Compute the set of states wher...

Correct?

Outside Q_n, BAD can force Prob(reach Q) = 0

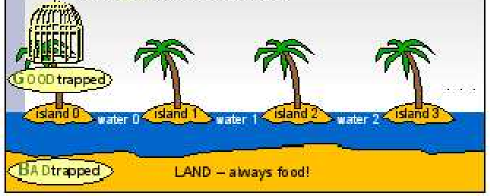
In Q_n, GOOD can force Prob(reach Q) > 0 by construction



Background Model Algorithm Conclusion

Algorithm Idea

- A world of islands and water
- No food (= F) on islands
- In water, may float to island (possibly elsewhere, too)
- Find BAD islands and water!



Background Model Algorithm Conclusion

Algorithm Idea

- A world of islands and water
- No food (= F) on islands
- In water, may float to island (possibly elsewhere, too)
- Find BAD islands and water!

Background Model Algorithm Conclusion

Algorithm Idea

- A world of islands and water
- No food (= F) on islands
- In water, may float to island (possibly elsewhere, too)
- Find BAD islands and water!

Background Model Algorithm Conclusion

Algorithm Idea

- A world of islands and water
- No food (= F) on islands
- In water, may float to island (possibly elsewhere, too)
- Find BAD islands and water!

Background Model Algorithm Conclusion

Algorithm Idea

- A world of islands and water
- No food (= F) on islands
- In water, may float to island (possibly elsewhere, too)
- Find BAD islands and water!

Background Model Algorithm Conclusion

Algorithm Idea

- A world of islands and water
- No food (= F) on islands
- In water, may float to island (possibly elsewhere, too)
- Find BAD islands and water!

Background Model Algorithm Conclusion

Algorithm Idea

- A world of islands and water
- No food (= F) on islands
- In water, may float to island (possibly elsewhere, too)
- Find BAD islands and water!

Background Model Algorithm Conclusion

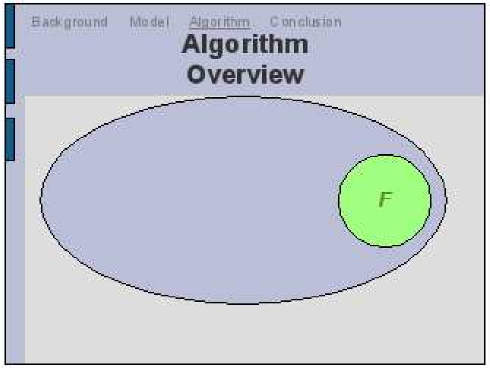
Algorithm Idea

- A world of islands and water
- No food (= F) on islands
- In water, may float to island (possibly elsewhere, too)
- Find BAD islands and water!

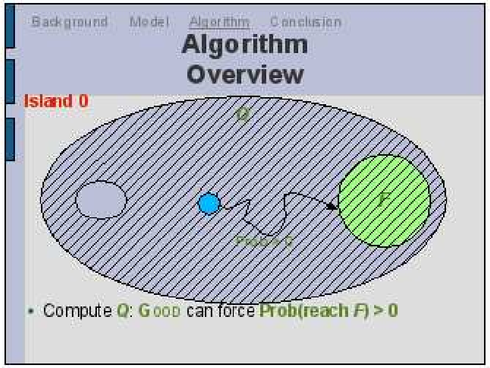
Background Model Algorithm Conclusion

Algorithm Idea

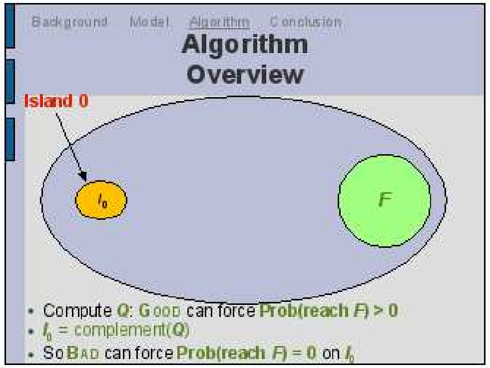
- A world of islands and water
- No food (= F) on islands
- In water, may float to island (possibly elsewhere, too)
- Find BAD islands and water!



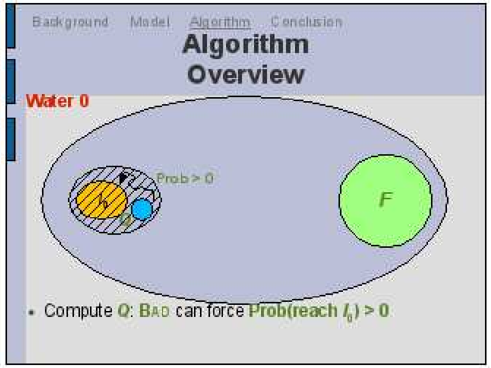
Slide 49



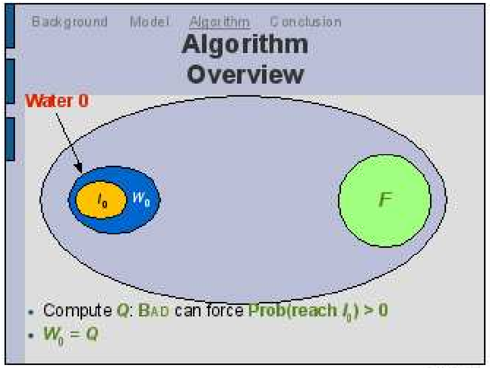
Slide 50



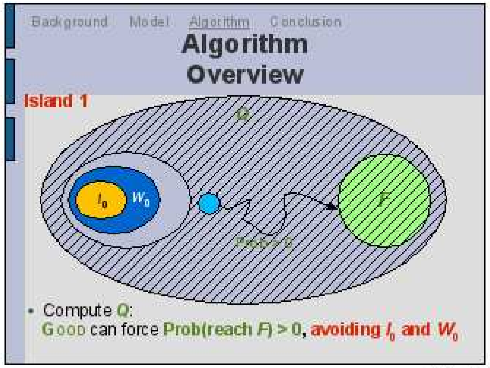
Slide 51



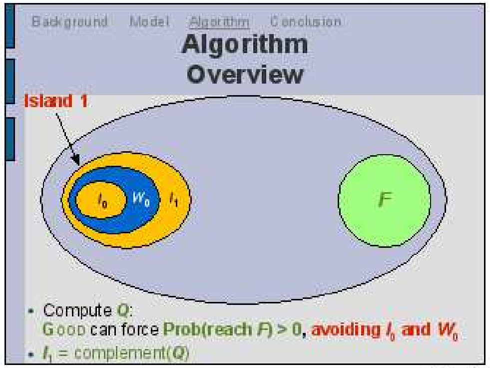
Slide 52



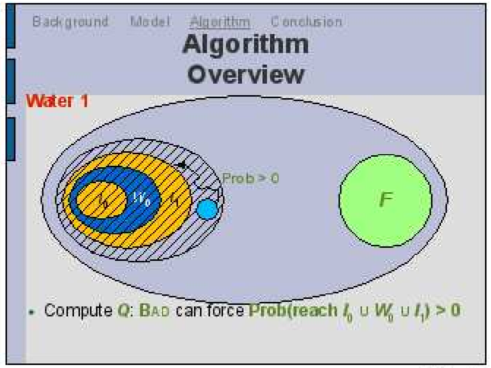
Slide 53



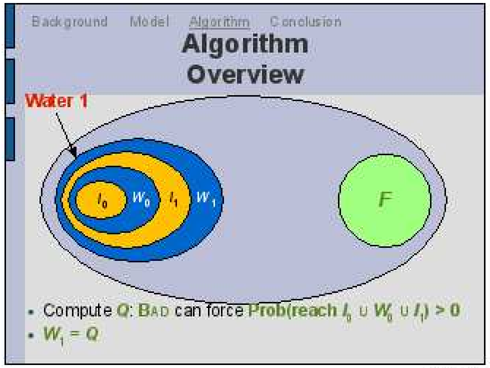
Slide 54



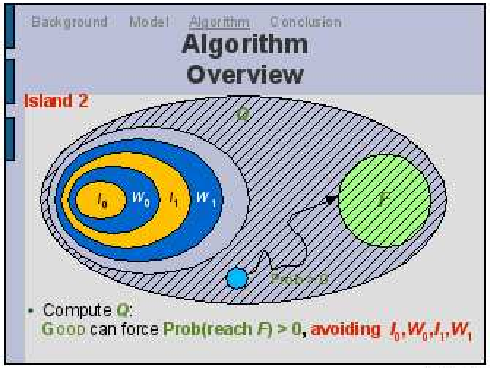
Slide 55



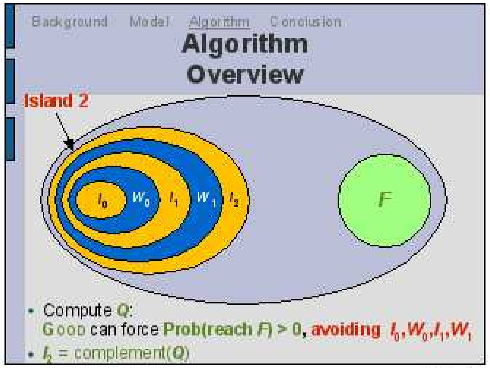
Slide 56



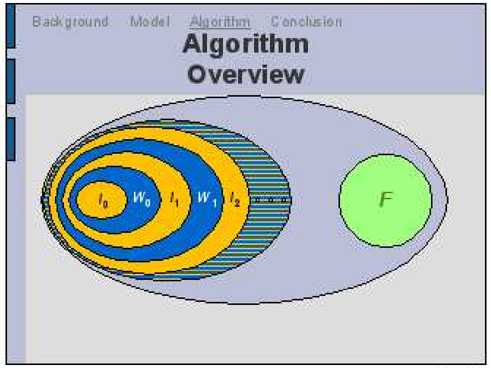
Slide 57



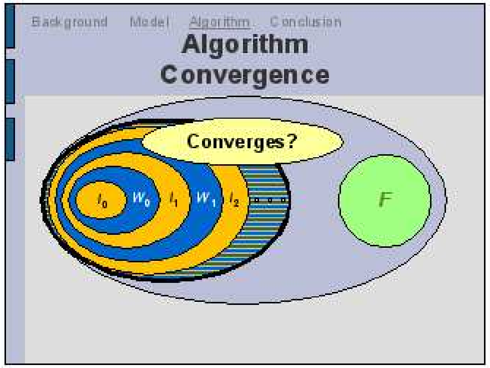
Slide 58



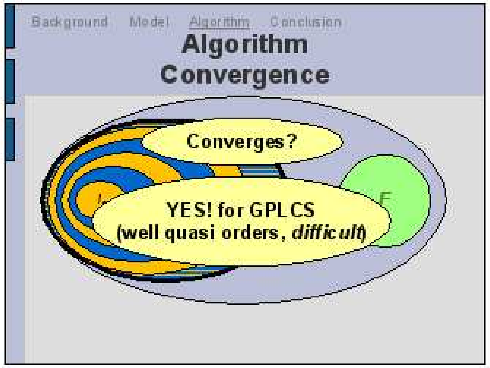
Slide 59



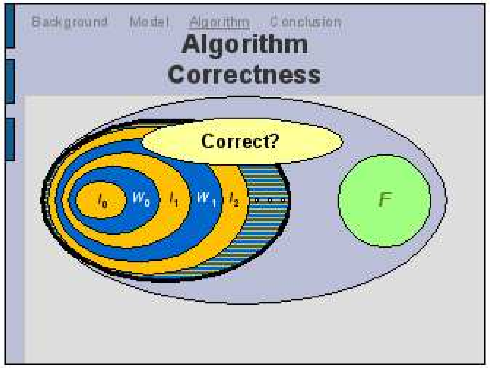
Slide 60



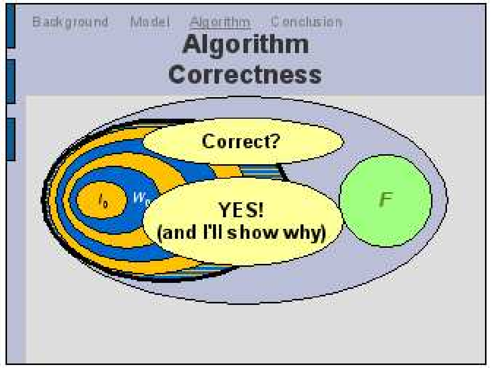
Slide 61



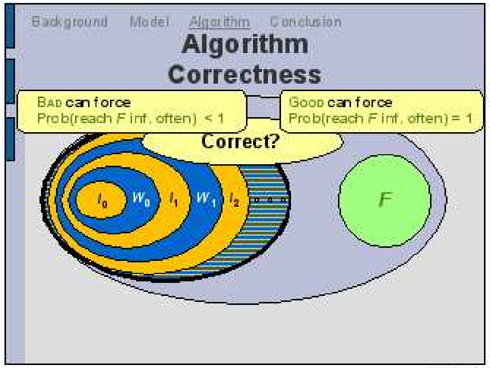
Slide 62



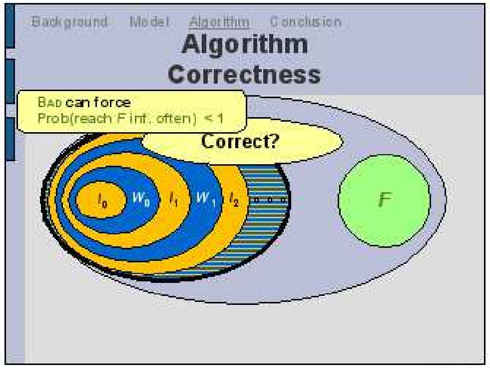
Slide 63



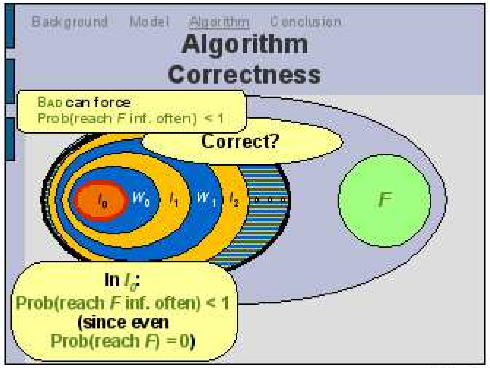
Slide 64



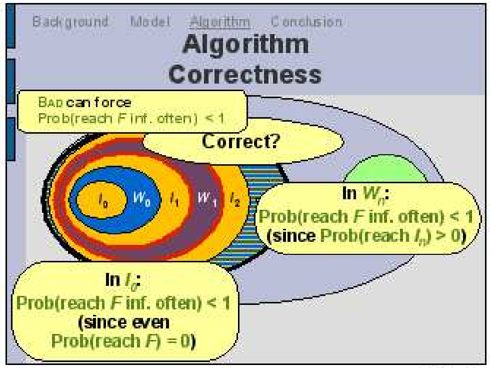
Slide 65



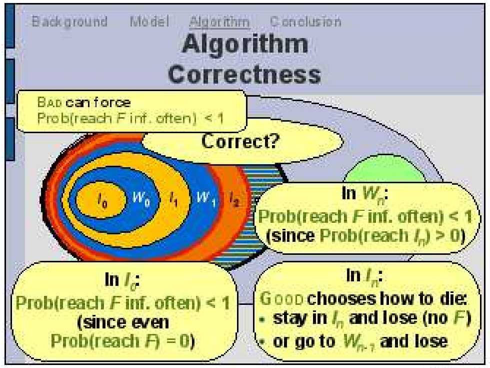
Slide 66



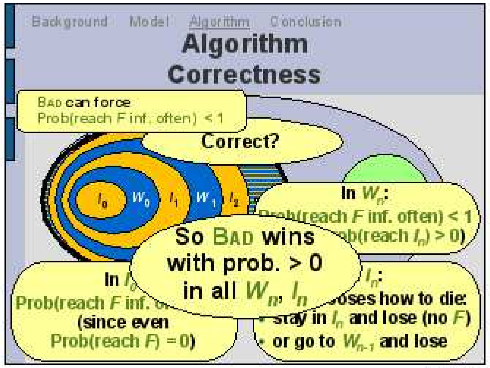
Slide 67



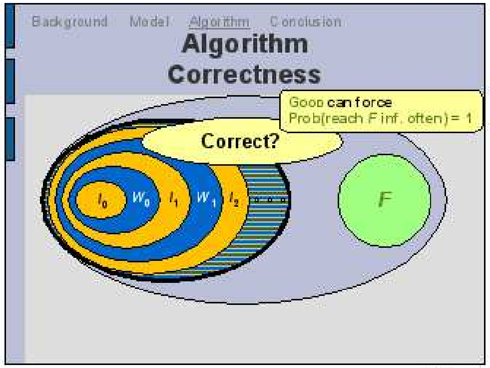
Slide 68



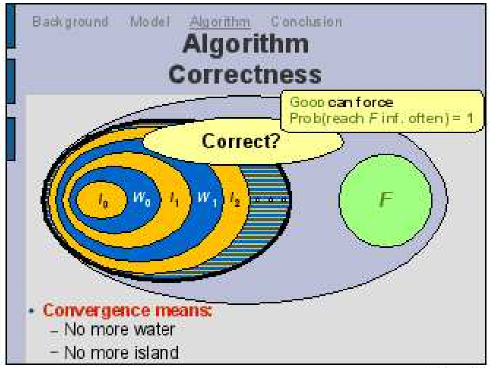
Slide 69



Slide 70

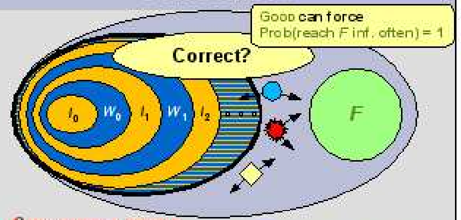


Slide 71



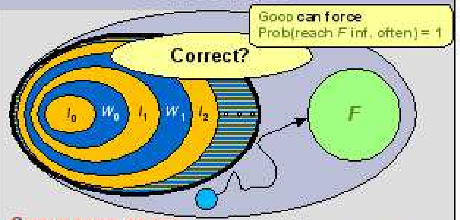
Slide 72

Algorithm Correctness



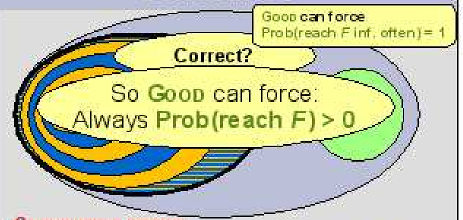
- Convergence means:
  - No more water $\Rightarrow$ Good can stay outside I_n, W_n
  - No more island

Algorithm Correctness



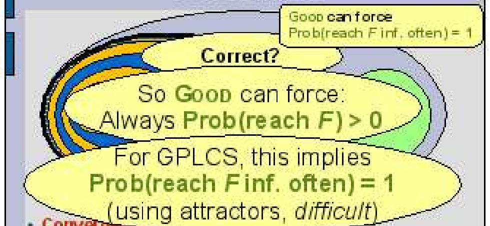
- Convergence means:
  - No more water $\Rightarrow$ Good can stay outside I_n, W_n
  - No more island $\Rightarrow$ Good can force Prob(reach F) > 0

Algorithm Correctness



- Convergence means:
  - No more water $\Rightarrow$ Good can stay outside I_n, W_n
  - No more island $\Rightarrow$ Good can force Prob(reach F) > 0

Algorithm Correctness



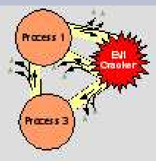
- Converg...
  - No more water $\Rightarrow$ Good can stay outside I_n, W_n
  - No more island $\Rightarrow$ Good can force Prob(reach F) > 0

Overview

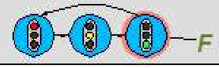
- Background
- Models
- Algorithm
- Conclusion

The Problem We Studied

- GPLCS:
  Protocol vs. Cracker

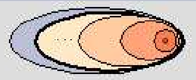


- Repeated reachability:
  reach "good" state infinitely many times



Algorithm

- Force-set:
  - Prob(reach Q) > 0
  - Backward search



- Winning sets for Bad:
  - Prob(reach F inf. often) < 1
  - Search for islands and water

Extensions

- Construct winning strategies
- Concurrent games
  Simultaneous moves
- Reachability games
  Can Good force Prob(reach F) = 1 ?

Can do

- Parity games
  Generalization of repeated reachability

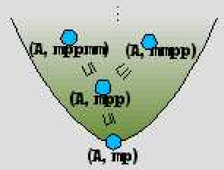
Unclear

The End

Questions?

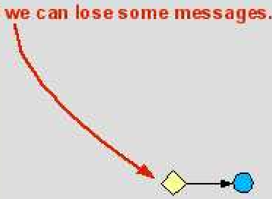
Convergence: Well Quasi Orders

- Recall: State: s = (A, mppp)
- Subword ordering:
  pmp $\sqsubseteq$ mpmmp
- Upward closure:
  all "bigger" states
- Message loss
  $\Rightarrow$ subword



Convergence: Well Quasi Orders

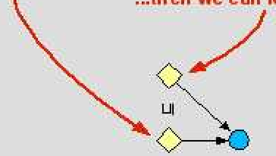
If we can lose some messages...



Convergence: Well Quasi Orders

If we can lose some messages...

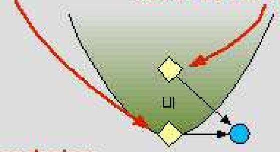
...then we can lose more!



Convergence: Well Quasi Orders

If we can lose some messages...

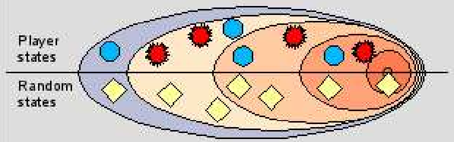
...then we can lose more!



Conclusion:
RANDOM states one step before are upward closed

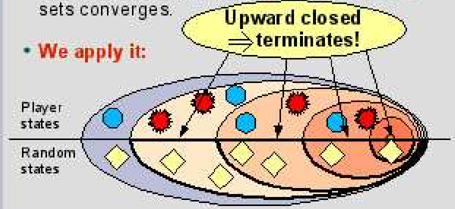
Convergence of Force-sets: Well Quasi Orders

- **Theorem (Higman 1952):** Any sequence of increasing, upward closed sets converges.
- **We apply it:**



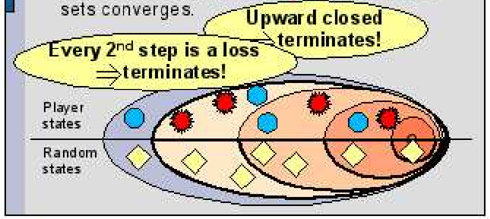
Convergence of Force-sets: Well Quasi Orders

- **Theorem (Higman 1952):** Any sequence of increasing, upward closed sets converges.
- **We apply it:**



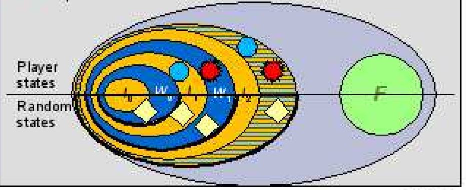
Convergence of Force-sets: Well Quasi Orders

- **Theorem (Higman 1952):** Any sequence of increasing, upward closed sets converges.



Convergence of Algorithm: Well Quasi Orders

- Argument is much more difficult!
- RANDOM states in water upward closed
- Sequence of waters terminates
- Sequence of islands terminates



The Fine Print: Finite-Memory Strategies

- **We assume BAD is restricted** to *finite-memory strategies*
- **BAD cannot remember** unbounded amount of history
- **Intuition:** Evil cracker has a finite hard disk
- Needed to prove correctness

Leftover

Repeated Reachability

- ***F* – set of "target states"**
- **Is *F* reached infinitely often?**
- **Example:**
  - Will it always return to the green state?