SLG-WAM for non-LRD Stratified Programs

Presented by Spyros Hadjichristodoulou
shadjichrist@cs.stonybrook.edu

State University of New York at Stony Brook
Overview

Preliminaries

SLG-WAM for Non LRD-Stratified Programs

Detection and Handling of Negative Loops

Delaying and Handling Conditional Answers

Simplification

Answer Completion
When is a set of subgoals $S$ completely evaluated?
Tabling for LRD-Stratified Programs

- When is a set of subgoals $S$ completely evaluated?
- We need information about subgoal dependencies
• When is a set of subgoals $S$ completely evaluated?
• We need information about subgoal dependencies
• Subgoal Dependency Graph (SDG)
Tabling for LRD-Stratified Programs

- When is a set of subgoals \( S \) completely evaluated?
- We need information about subgoal dependencies
- Subgoal Dependency Graph (SDG)
- WAM ops for definite programs: NEW SUBGOAL, NEW ANSWER, ANSWER RETURN, COMPLETION, NEW INTERIOR
Tabling for LRD-Stratified Programs

- When is a set of subgoals $S$ completely evaluated?
- We need information about subgoal dependencies
- Subgoal Dependency Graph (SDG)
- WAM ops for definite programs: NEW SUBGOAL, NEW ANSWER, ANSWER RETURN, COMPLETION, NEW INTERIOR
- WAM ops added for LRD: FLOUNDERING, NEGATION FAILURE, NEGATION SUCCESS, NEGATION SUSPENSION
• Early Completion: Only one subgoal \((A)\) which has an answer that is a variant of \(A\)
On Early Completion

- Early Completion: Only one subgoal (A) which has an answer that is a variant of A
- A ground subgoal can be considered completely evaluated upon the addition of its first answer into its answer table
Early Completion: Only one subgoal (A) which has an answer that is a variant of A

- A ground subgoal can be considered completely evaluated upon the addition of its first answer into its answer table

- Extend the above with answer subsumption
On Early Completion

- Early Completion: Only one subgoal (A) which has an answer that is a variant of A
- A ground subgoal can be considered completely evaluated upon the addition of its first answer into its answer table
- Extend the above with answer subsumption
- When an answer is derived which is identical up to variable renaming to its associated subgoal S, S is marked as completely evaluated, and do COMPLETION with \{S\}
Tabling for Non LRD-Stratified Programs

- DELAYING transformation grounds negative literals involved in negative loops to proceed with resolution of remaining literals

• answer_template :- [delay_list] goal_list.
  • If delay_list is empty, node is an answer, added to table with NEW ANSWER operation.
  • answer_template subsumed by goal S associated with, answer substitution.
  • Answers with empty delay_list are called unconditional, others conditional.
  • Answers with no literals in delay_list being false are called supported, otherwise unsupported.
  • Subgoal S succeeds if it has an unconditional answer that is a variant of S, and fails if S is completely evaluated with no answers.
Tabling for Non LRD-Stratified Programs

- DELAYING transformation grounds negative literals involved in negative loops to proceed with resolution of remaining literals
- `answer_template :- [delay_list] goal_list.`

If `delay_list` is empty, node is an answer, added to table with `NEW ANSWER` operation

`answer_template` subsumed by goal `S` associated with, answer substitution

Answers with empty `delay_list` are called unconditional, others conditional

Answers with no literals in `delay_list` being false are called supported, otherwise unsupported

Subgoal `S` succeeds if it has an unconditional answer that is a variant of `S`, and fails if `S` is completely evaluated with no answers.
TABLING transformation grounds negative literals involved in negative loops to proceed with resolution of remaining literals

- \texttt{answer\_template} :- [\texttt{delay\_list}] \texttt{goal\_list}.
- If \texttt{delay\_list} is empty, node is an answer, added to table with \texttt{NEW ANSWER} operation
Tabling for Non LRD-Stratified Programs

- DELAYING transformation grounds negative literals involved in negative loops to proceed with resolution of remaining literals
- `answer_template :- [delay_list] goal_list.`
- If `delay_list` is empty, node is an answer, added to table with NEW ANSWER operation
- `answer_template` subsumed by goal S associated with, `answer substitution`
Tabling for Non LRD-Stratified Programs

- DELAYING transformation grounds negative literals involved in negative loops to proceed with resolution of remaining literals
- `answer_template :- [delay_list] goal_list.`
- If `delay_list` is empty, node is an answer, added to table with `NEW ANSWER` operation
- `answer_template` subsumed by goal S associated with, `answer substitution`
- Answers with empty `delay_list` are called `unconditional`, others `conditional`
Tabling for Non LRD-Stratified Programs

- DELAYING transformation grounds negative literals involved in negative loops to proceed with resolution of remaining literals
- `answer_template :- [delay_list] goal_list`.
- If `delay_list` is empty, node is an answer, added to table with NEW ANSWER operation
- `answer_template` subsumed by goal S associated with, `answer substitution`
- Answers with empty `delay_list` are called unconditional, others conditional
- Answers with no literals in `delay_list` being false are called supported, otherwise unsupported
Tabling for Non LRD-Stratified Programs

- DELAYING transformation grounds negative literals involved in negative loops to proceed with resolution of remaining literals
- answer_template :- [delay_list] goal_list.
- If delay_list is empty, node is an answer, added to table with NEW ANSWER operation
- answer_template subsumed by goal S associated with, answer substitution
- Answers with empty delay_list are called unconditional, others conditional
- Answers with no literals in delay_list being false are called supported, otherwise unsupported
- Subgoal S succeeds if it has an unconditional answer that is a variant of S, and fails if S is completely evaluated with no answers
Example 1

\[ t: \neg p. \quad p: \neg q. \quad q: \neg r. \quad r: \neg q, s. \quad s: \neg r. \]
Example 1

\[
\begin{align*}
t & : -\neg p. \\
p & : -q. \\
q & : -\neg r. \\
r & : -q, s. \\
s & : -r. \\
t : susp & : t : -\neg p. \\
p : act & : p : -q. \\
q : act & : q : -[\neg r]. \\
r : act & : r : -q, s.
\end{align*}
\]
Example 1

t: \neg p.
p: \neg q.
q: \neg r.
r: \neg q, s.
s: \neg r.

t: susp: t: \neg p.
p: act: p: \neg q.
q: ans: q: \neg[\neg r]. NA
r: act: r: \neg q, s.
Example 1

\[
\begin{align*}
t & : \neg p. \\
p & : \neg q. \\
q & : \neg r. \\
r & : \neg q, s. \\
s & : \neg r. \\
\end{align*}
\]

\[
\begin{align*}
t & : \text{ susp } t : \neg p. \\
p & : \text{ ans } p : \neg [q]. \text{ AR} \\
q & : \text{ ans } q : \neg [\neg r]. \\
r & : \text{ act } r : \neg [q], s. \text{ AR} \\
s & : \text{ act } s : \neg r. \\
\end{align*}
\]
Example 1

\[
\begin{align*}
\text{t:} & \neg p. & \text{t: susp: t:} & \neg p. \\
\text{p:} & \neg q. & \text{p: ans: p:} & \neg [q]. \\
\text{q:} & \neg r. & \text{q: ans: q:} & \neg [\neg r]. \\
\text{r:} & \neg q, s. & & \text{failed(r).} \\
\text{s:} & \neg r. & & \text{failed(s).}
\end{align*}
\]
Example 1

\[
\begin{align*}
\text{t:} & \neg \neg p. \\
\text{p:} & \neg q. \\
\text{q:} & \neg \neg r. \\
\text{r:} & \neg q, s. \\
\text{s:} & \neg r. \\
\text{t: susp:} & t: \neg \neg p. \\
\text{p: ans:} & p: \neg [q]. \\
\text{q: ans:} & q. \text{ SI} \\
\text{failed (r).} & \\
\text{failed (s).} & \\
\end{align*}
\]
Example 1

\[
\begin{align*}
  t &::= \neg p. \\
  p &::= \neg q. \\
  q &::= \neg r. \\
  r &::= \neg q, s. \\
  s &::= \neg r.
\end{align*}
\]

\[
\begin{align*}
  t &::= \text{susp} : t::= \neg p. \\
  p &::= \text{ans} : p. \text{ SI} \\
  q &::= \text{ans} : q. \text{ SI} \\
  \text{failed} (r). \\
  \text{failed} (s).
\end{align*}
\]
Example 1

t: \neg p.
p: \neg q.
q: \neg r.
r: \neg q, s.
s: \neg r.

failed(t). NF
p: ans :p. SI
q: ans :q. SI
failed(r).
failed(s).

 Spyros Hadjichristodoulou
 SLG-WAM
Handling Negative Loops

- Dynamically changing left-to-right computation rules
Handling Negative Loops

- Dynamically changing left-to-right computation rules
- Avert dependencies through negation

SIMPLIFICATION operations
true delayed literals are deleted from answers
false answers get deleted from the system
Variable bindings accumulated are propagated through uniﬁcation, but NOT delayed literals

Residual program

Spyros Hadjichristodoulou
Handling Negative Loops

- Dynamically changing left-to-right computation rules
- Avert dependencies through negation
- SIMPLIFICATION operations
Handling Negative Loops

- Dynamically changing left-to-right computation rules
- Avert dependencies through negation
- SIMPLIFICATION operations
- true delayed literals are deleted from answers
Handling Negative Loops

- Dynamically changing left-to-right computation rules
- Avert dependencies through negation
- SIMPLIFICATION operations
- True delayed literals are deleted from answers
- Answers with false delayed literals get deleted from the system
Handling Negative Loops

- Dynamically changing left-to-right computation rules
- Avert dependencies through negation
- SIMPLIFICATION operations
- true delayed literals are deleted from answers
- Answers with false delayed literals get deleted from the system
- Variable bindings accumulated are propagated through unification, but NOT delayed literals
Handling Negative Loops

- Dynamically changing left-to-right computation rules
- Avert dependencies through negation
- SIMPLIFICATION operations
- true delayed literals are deleted from answers
- Answers with false delayed literals get deleted from the system
- Variable bindings accumulated are propagated through unification, but NOT delayed literals
- Residual program
How to handle Non LRD-Stratified SLG Resolution?

- Detecting and resolving negative loops
How to handle Non LRD-Stratified SLG Resolution?

- Detecting and resolving negative loops
- Storing conditional answers in tables
How to handle Non LRD-Stratified SLG Resolution?

- Detecting and resolving negative loops
- Storing conditional answers in tables
- Manipulating delay lists
How to handle Non LRD-Stratified SLG Resolution?

- Detecting and resolving negative loops
- Storing conditional answers in tables
- Manipulating delay lists
- SIMPLIFICATION operations
Completion in Normal Logic Programs

• SDG of program-query pair is embedded in the choice point stack of SLG-WAM
Completion in Normal Logic Programs

(i)

- SDG of program-query pair is embedded in the choice point stack of SLG-WAM
- completion stack contains its stack-based approximation
Completion in Normal Logic Programs

(i)

- SDG of program-query pair is embedded in the choice point stack of SLG-WAM
- completion stack contains its stack-based approximation
- If no negative loops, approximation is OK
Completion in Normal Logic Programs

(i)

- SDG of program-query pair is embedded in the *choice point* stack of SLG-WAM
- *completion* stack contains its stack-based approximation
- If no negative loops, approximation is OK
- Else, break negative loops with DELAYING
SDG of program-query pair is embedded in the choice point stack of SLG-WAM

completion stack contains its stack-based approximation
If no negative loops, approximation is OK
Else, break negative loops with DELAYING
Performed by negation_resume
Completion in Normal Logic Programs

(ii)

- 4 steps of completion SLG-WAM instruction:

1. Lazily construct the SDG in order to find an independent SCC and find possible negative loop among subgoals in.
2. Perform left-to-right stratification check, classify the suspensions of subgoals in, remove subgoals participating in negative loop from the SDG and place them in NS.
3. Schedule negation resume instructions for all suspensions in NS, and if NS is empty complete every subgoal in.
4. Reclaim stack space of subgoals that were completed and compact completion stack.
Completion in Normal Logic Programs

(ii)

- 4 steps of completion SLG-WAM instruction:
  1. Lazily construct the SDG in order to find an independent SCC $I$ and find possible negative loop among subgoals in $I$
Completion in Normal Logic Programs
(ii)

- 4 steps of completion SLG-WAM instruction:
  1. Lazily construct the SDG in order to find an independent SCC \( I \) and find possible negative loop among subgoals in \( I \)
  2. Perform left-to-right stratification check, classify the suspensions of subgoals in \( I \), remove subgoals participating in negative loop from the SDG and place them in \( NS \)
Completion in Normal Logic Programs

(ii)

- 4 steps of completion SLG-WAM instruction:
  1. Lazily construct the SDG in order to find an independent SCC $I$ and find possible negative loop among subgoals in $I$
  2. Perform left-to-right stratification check, classify the suspensions of subgoals in $I$, remove subgoals participating in negative loop from the SDG and place them in $NS$
  3. Schedule negation-resume instructions for all suspensions in $NS$, and if $NS$ is empty complete every subgoal in $I$
Completion in Normal Logic Programs (ii)

- 4 steps of completion SLG-WAM instruction:
  1. Lazily construct the SDG in order to find an *independent SCC* $I$ and find possible negative loop among subgoals in $I$
  2. Perform left-to-right stratification check, classify the suspensions of subgoals in $I$, remove subgoals participating in negative loop from the SDG and place them in $NS$
  3. Schedule *negation resume* instructions for all suspensions in $NS$, and if $NS$ is empty complete every subgoal in $I$
  4. Reclaim stack space of subgoals that were completed and compact completion stack
\( t_{not}(S) \) (i)

- \( T = t_{not}(S) : \)
  - If \( S \) is completed:
    - If \( S \) fails then \( T \) succeeds
    - If \( S \) succeeds then \( T \) fails
  - If \( S \) has only conditional answers \( T \) delays
\[ tnot(S) \]  

- \( T = tnot(S) : \)
- If \( S \) is completed :

If \( S \) fails then \( T \) succeeds
If \( S \) succeeds then \( T \) fails
\( tnot(S) \)  

- \( T = tnot(S) \):
  - If \( S \) is completed:
  - If \( S \) fails then \( T \) succeeds
\( tnot(S) \)  

- \( T = tnot(S) \):
- If \( S \) is completed:
- If \( S \) fails then \( T \) succeeds
- If \( S \) succeeds then \( T \) fails
\( tnot(S) \)

- \( T = tnot(S) : \)
  - If \( S \) is completed :
  - If \( S \) fails then \( T \) succeeds
  - If \( S \) succeeds then \( T \) fails
  - If \( S \) has only conditional answers \( T \) delays
tnot(S) (ii)

- Else if S is not completed negation suspend suspends current computation and can:
  - Succeed: Negation suspension removed by early completion
  - Otherwise: Schedule negation resume when
    1. unsuspending on completion: fail (resolved away) or have conditional answers only (added to delay list)
    2. DELAYING to break negative loop: add to delay list
\text{tnot}(S) \quad (\text{ii})

- Else if \( S \) is not completed negation\_suspend suspends current computation and can:
  - Succeed: Negation suspension removed by early completion
  - Otherwise: Schedule negation resume when
    1. unsuspending on completion: fail (resolved away) or have conditional answers only (added to delay list)
    2. DELAYING to break negative loop: add to delay list
else if \( S \) is not completed, negation_suspend suspends current computation and can:

- Succeed: Negation suspension removed by early completion
- Otherwise: Schedule negation_resume when

\( \text{tnot}(S) \ (\text{ii}) \)
Else if $S$ is not completed, negation_suspend suspends current computation and can:

- **Succeed**: Negation suspension removed by early completion
- **Otherwise**: Schedule negation_resume when

1. unsuspending on completion: fail (resolved away) or have conditional answers only (added to delay list)
\text{tnot}(S) \quad (ii)

- Else if $S$ is not completed \texttt{negation_suspend} suspends current computation and can:
  - Succeed: Negation suspension removed by early completion
  - Otherwise: Schedule \texttt{negation_resume} when
    1. unsuspending on completion: fail (resolved away) or have conditional answers only (added to delay list)
    2. \texttt{DELAYING} to break negative loop: add to delay list
Delay Lists and Delay Elements

- Delay list literals:

  1. Undeclared under WFS
  2. Truth value yet undetermined

SIMPLIFICATION operations remove them from the delay list or dispose of the entire rule.

In SLG-WAM a delay list is \( \langle \text{Subgoal Id}, \text{Literal, Answer Subst Id} \rangle \)
Delay Lists and Delay Elements

- Delay list literals:
  1. Undefined under WFS
**Delay Lists and Delay Elements**

- Delay list literals:
  1. Undefined under WFS
  2. Truth value yet undetermined
Delay Lists and Delay Elements

- Delay list literals:
  1. Undefined under WFS
  2. Truth value yet undetermined

- SIMPLIFICATION operations remove them from the delay list or dispose of the entire rule
Delay Lists and Delay Elements

- Delay list literals:
  1. Undefined under WFS
  2. Truth value yet undetermined

- SIMPLIFICATION operations remove them from the delay list or dispose of the entire rule

- In SLG-WAM a delay list is \langle Subgoal_Id, Literal, Answer_Subst_Id \rangle
p(f(X)) :- p(Y), r(X,Y), ¬ q(X).
p(g(X)) :- q(X).
p(Y) :- r(X,Y), ¬ p(Y).
q(X) :- r(X,g(X)).
r(a,g(c)).
r(b,g(b)).
Example 2

\[
\begin{align*}
p(f(X)) & :- p(Y), r(X,Y), \neg q(X). \\
p(g(X)) & :- q(X). \\
p(Y) & :- r(X,Y), \neg p(Y). \\
q(X) & :- r(X,g(X)). \\
r(a,g(c)). \\
r(b,g(b)). \\
\end{align*}
\]

\[p(X) \quad g(b) \quad []\]
Example 2

\[
\begin{align*}
p(f(X)) & : \neg p(Y), r(X,Y), \neg q(X). \\
p(g(X)) & : q(X). \\
p(Y) & : r(X,Y), \neg p(Y). \\
q(X) & : r(X,g(X)). \\
r(a,g(c)). \\
r(b,g(b)). \\
p(X) & \\
g(b) & [] \\
g(c) & (\neg p(g(c)))] 
\end{align*}
\]
Example 2

\[
\begin{align*}
p(f(X)) & : - p(Y), r(X,Y), \neg q(X). \\
p(g(X)) & : - q(X). \\
p(Y) & : r(X,Y), \neg p(Y). \\
q(X) & : r(X,g(X)). \\
r(a,g(c)). \\
r(b,g(b)). \\
p(X) & \quad g(b) \quad [] \\
g(c) & \quad [\langle \neg p(g(c)) \rangle] \\
f(a) & \quad [\langle p(X), p(g(c)), p(X)_2 \rangle]
\end{align*}
\]
Example 2

\[
\begin{align*}
p(f(X)) & :- p(Y), r(X,Y), \neg q(X). \\
p(g(X)) & :- q(X). \\
p(Y) & :- r(X,Y), \neg p(Y). \\
q(X) & :- r(X,g(X)). \\
r(a,g(c)). \\
r(b,g(b)). \\
p(X) & : [g(b)] \\
g(c) & : [\langle \neg p(g(c)) \rangle] \\
f(a) & : [\langle p(X), p(g(c)), p(X)_2 \rangle] \\
p(g(c)) & : [\langle \neg p(g(c)) \rangle]
\end{align*}
\]
Example 2

\[
\begin{align*}
\text{p}(f(X)) & :\neg \text{p}(Y), \text{r}(X,Y), \neg \text{q}(X). \\
\text{p}(g(X)) & :\neg \text{q}(X). \\
\text{p}(Y) & :\neg \text{r}(X,Y), \neg \text{p}(Y). \\
\text{q}(X) & :\neg \text{r}(X,g(X)). \\
\text{r}(a,g(c)). \\
\text{r}(b,g(b)).
\end{align*}
\]

\[
\begin{align*}
p(X) & \quad g(b) \quad [] \\
g(c) & \quad \langle \neg \text{p}(g(c)) \rangle \\
f(a) & \quad \langle \text{p}(X), \text{p}(g(c)), \text{p}(X)_2 \rangle \\
p(g(c)) & \quad \langle \neg \text{p}(g(c)) \rangle \\
p(g(b)) & \quad []
\end{align*}
\]
Example 2 (cont’d)

```prolog
:- table pt/1, u/1.
pt(X) :- p(X).
pp(X) :- p(X).
u(X) :- pp(X), ~ pt(X).
```
Example 2 (cont’d)

:- table pt/1, u/1.
pt(X) :- p(X).
pp(X) :- p(X).
u(X) :- pp(X), ¬ pt(X).

\[ pt(X) \quad g(b) \quad [] \]
Example 2 (cont’d)

:- table pt/1, u/1.
pt(X) :- p(X).
pp(X) :- p(X).
u(X) :- pp(X), ¬ pt(X).

\[
\begin{align*}
pt(X) & \quad g(b) \quad [] \\
g(c) & \quad [\langle p(X), p(g(c)), p(X)_2 \rangle ]
\end{align*}
\]
Example 2 (cont’d)

:- table pt/1, u/1.
pt(X) :- p(X).
pp(X) :- p(X).
u(X) :- pp(X), ¬ pt(X).

\[
\begin{align*}
pt(X) & \quad g(b) \quad [] \\
g(c) & \quad [\langle p(X) , p(g(c)) , p(X)_2 \rangle ] \\
f(a) & \quad [\langle p(X) , p(f(a)) , p(X)_3 \rangle ]
\end{align*}
\]
Example 2 (cont’d)

:- table pt/1, u/1.
pt(X) :- p(X).
pp(X) :- p(X).
  u(X) :- pp(X), ¬ pt(X).

\[
\begin{align*}
pt(X) & \quad \text{g(b)} & [] \\
g(c) & \quad [\langle p(X), p(g(c)), p(X)_2 \rangle] \\
f(a) & \quad [\langle p(X), p(f(a)), p(X)_3 \rangle] \\
u(X) & \quad \text{g(c)} & [\langle p(X), p(g(c)), p(X)_2 \rangle, \langle \neg pt(g(c)) \rangle]
\end{align*}
\]
Example 2 (cont’d)

:- table pt/1, u/1.
pt(X) :- p(X).
pp(X) :- p(X).
u(X) :- pp(X), ¬ pt(X).

\[
\begin{align*}
pt(X) & \quad g(b) \quad [] \\
g(c) & \quad [\langle p(X), p(g(c)), p(X)_2 \rangle ] \\
f(a) & \quad [\langle p(X), p(f(a)), p(X)_3 \rangle ] \\
u(X) & \quad g(c) \quad [\langle p(X), p(g(c)), p(X)_2 \rangle , \langle ¬ pt(g(c)) \rangle ] \\
f(a) & \quad [\langle p(X), p(f(a)), p(X)_2 \rangle , \langle ¬ p(f(a)) \rangle ]
\end{align*}
\]
Delay List Representation and Maintenance

- Want to efficiently perform SIMPLIFICATION on delay lists

- Current value of delay list also maintained on the heap as a Prolog list

- When answer is derived, delay list is copied from the heap, interned and stored in the answer table

- Extending and interning delay lists

- Delay list maintenance using a new register $D$ and a new field $D_{\text{reg}}$ in choice point frames
Delay List Representation and Maintenance

- Want to efficiently perform SIMPLIFICATION on delay lists
- Current value of delay list also maintained on the heap as a Prolog list
Delay List Representation and Maintenance

- Want to efficiently perform SIMPLIFICATION on delay lists
- Current value of delay list also maintained on the heap as a Prolog list
- When answer is derived, delay list is copied from the heap, interned and stored in the answer table
Delay List Representation and Maintenance

- Want to efficiently perform SIMPLIFICATION on delay lists
- Current value of delay list also maintained on the heap as a Prolog list
- When answer is derived, delay list is copied from the heap, interned and stored in the answer table
- Extending and interning delay lists
Delay List Representation and Maintenance

• Want to efficiently perform SIMPLIFICATION on delay lists
• Current value of delay list also maintained on the heap as a Prolog list
• When answer is derived, delay list is copied from the heap, interned and stored in the answer table
• Extending and interning delay lists
• Delay list maintenance using a new register $D$ and a new field $D_{reg}$ in choice point frames
1. Conditions for simplification of delayed literals should be detected, and the SIMPLIFICATION operation should be applied as soon as possible.
Simplification Principles

1. Conditions for simplification of delayed literals should be detected, and the SIMPLIFICATION operation should be applied as soon as possible.

2. Derivation of an unconditional answer for a subgoal $S$ should immediately remove from the table for $S$ all conditional answers with the same answer substitution.
1. Conditions for simplification of delayed literals should be detected, and the SIMPLIFICATION operation should be applied as soon as possible.

2. Derivation of an unconditional answer for a subgoal $S$ should immediately remove from the table for $S$ all conditional answers with the same answer substitution.

3. Unsupported answer substitutions should be removed from the answer table as soon as possible.
Events that Trigger Simplification

1. Derivation of an unconditional answer for a subgoal
Events that Trigger Simplification

1. Derivation of an unconditional answer for a subgoal
   1.1 through program/answer resolution before the completion of a subgoal
Events that Trigger Simplification

1. Derivation of an unconditional answer for a subgoal
   1.1 through program/answer resolution before the completion of a subgoal
   1.2 through simplification of an existing conditional answer by deleting the last element of its delay list
Events that Trigger Simplification

1. Derivation of an unconditional answer for a subgoal
   1.1 through program/answer resolution before the completion of a subgoal
   1.2 through simplification of an existing conditional answer by deleting the last element of its delay list

2. Completion of a subgoal with no answers
Events that Trigger Simplification

1. Derivation of an unconditional answer for a subgoal
   1.1 through program/answer resolution before the completion of a subgoal
   1.2 through simplification of an existing conditional answer by deleting the last element of its delay list

2. Completion of a subgoal with no answers

3. Deletion of an unsupported answer substitution
Events that Trigger Simplification

1. Derivation of an unconditional answer for a subgoal
   1.1 through program/answer resolution before the completion of a subgoal
   1.2 through simplification of an existing conditional answer by deleting the last element of its delay list
2. Completion of a subgoal with no answers
3. Deletion of an unsupported answer substitution
   3.1 remove all answers positively delayed on it from the system
Events that Trigger Simplification

1. Derivation of an unconditional answer for a subgoal
   1.1 through program/answer resolution before the completion of a subgoal
   1.2 through simplification of an existing conditional answer by deleting the last element of its delay list

2. Completion of a subgoal with no answers

3. Deletion of an unsupported answer substitution
   3.1 remove all answers positively delayed on it from the system
   3.2 if the subgoal was completed and it was its last answer substitution, it has to fail
Simplification Procedures

- Deletion of a delay element from delay lists
Simplification Procedures

- Deletion of a delay element from delay lists
  1. `simplify_neg_fails(Subgoal)`
Simplification Procedures

• Deletion of a delay element from delay lists
  1. simplify_neg_fails(Subgoal)
  2. simplify_pos_unconditional(Answer_Substitution)
Simplification Procedures

• Deletion of a delay element from delay lists
  1. `simplify_neg_fails(Subgoal)`
  2. `simplify_pos_unconditional(Answer_Substitution)`

• Deletion of unsupported answers
Simplification Procedures

- Deletion of a delay element from delay lists
  1. `simplify_neg_fails(Subgoal)`
  2. `simplify_pos_unconditional(Answer_Substitution)`
- Deletion of unsupported answers
  1. `simplify_neg_succeeds(Subgoal)`
Simplification Procedures

- Deletion of a delay element from delay lists
  1. `simplify_neg_fails(Subgoal)`
  2. `simplify_pos_unconditional(Answer_Substitution)`

- Deletion of unsupported answers
  1. `simplify_neg_succeeds(Subgoal)`
  2. `simplify_pos_unsupported(Answer_Substitution)`
Example 3

\[ p: \neg r, s, q. \]
\[ q: \neg r. \]
\[ r: \neg p. \]
\[ s: \neg t. \]
\[ t: \neg p, \neg q. \]

\[ p: \text{susp} : p: \neg r, s, q. \]
\[ r: \text{act} : r: \neg p. \]
Example 3

\[
\begin{align*}
p &: \neg r, s, q. \\
q &: \neg r. \\
r &: \neg p. \\
s &: \neg t. \\
t &: \neg p, \neg q.
\end{align*}
\]
Example 3

\begin{align*}
\text{p:} & \neg r, s, q. & \text{p:act:} & p: \neg \neg r, s, q. \\
\text{q:} & \neg r. & \text{r:act:} & r: \neg p. \\
\text{r:} & \neg p. & \text{s:act:} & s: \neg \neg \neg t. \\
\text{s:} & \neg \neg t. & \text{t:act:} & t: \neg \neg \neg p, \neg q. \\
\text{t:} & \neg \neg p, \neg q.
\end{align*}
Example 3

\[
\begin{align*}
p &: \neg r, s, q. & p &: \text{act} : p: -[\neg r], s, q. \\
q &: \neg r. & r &: \text{act} : r: \neg p. \\
r &: \neg p. & s &: \text{cans} : s: -[\neg t]. \\
s &: \neg t. & t &: \text{act} : t: -[\neg p], \neg q. \\
t &: \neg p, \neg q.
\end{align*}
\]
Example 3

\[\begin{align*}
    p & : \neg r, s, q. & p & : \text{act} : p : \neg [\neg r], s, q. \\
    q & : \neg r. & r & : \text{act} : r : \neg p. \\
    r & : \neg p. & s & : \text{cans} : s : \neg [\neg t]. \\
    s & : \neg t. & t & : \text{susp} : t : \neg [\neg p], \neg q. \\
    t & : \neg p, \neg q. & q & : \text{act} : q : \neg r. \\
    \end{align*}\]
Example 3

\[ p : - r, s, q. \]
\[ q : - r. \]
\[ r : - p. \]
\[ s : - t. \]
\[ t : - p, q. \]

\[ \text{fail}(p). \]
\[ \text{fail}(r). \]
\[ \text{s: cans: } s : - [\neg t]. \]
\[ \text{t: susp: } t : - [\neg p], \neg q. \]
\[ \text{fail}(q). \]
Example 3

\[ p : \neg r, s, q. \quad \text{fail}(p). \]
\[ q : \neg r. \quad \text{fail}(r). \]
\[ r : \neg p. \quad \text{s: cans :s :} \neg t]. \]
\[ s : \neg t. \quad \text{t: ans :} t. \]
\[ t : \neg p, \neg q. \quad \text{fail}(q). \]
Example 3

p:-¬r,s,q. fail(p).
q:-r. fail(r).
r:-p. fail(s).
s:-¬t. t:ans :t.
t:-¬p,¬q. fail(q).
Example 4

\begin{align*}
  p & : \neg q. \\
  q & : \neg p, \neg r, q. \\
  r & : \neg p, \neg q. \\

  p : \text{susp} : p : \neg q. \\
  q : \text{susp} : q : \neg p, \neg r, q. 
\end{align*}
Example 4

\[ p: \neg q. \]
\[ q: \neg p, \neg r, q. \]
\[ r: \neg p, \neg q. \]

\[ p: \text{act} : p: - [\neg q]. \]
\[ q: \text{susp} : q: - [\neg p], \neg r, q. \]
\[ r: \text{susp} : \neg p, \neg q. \]
Example 4

\begin{align*}
p & : \neg q. \\
q & : \neg \neg p, \neg r, q. \\
r & : \neg \neg p, \neg q. \\
p: \text{cans} & : p : \neg [\neg q]. \\
q: \text{susp} & : q : \neg [\neg p], \neg r, q. \\
r: \text{act} & : [\neg p], \neg q. \\
\end{align*}
Example 4

\begin{align*}
p & : \neg q. \\
q & : \neg p, \neg r, q. \\
r & : \neg p, \neg q. \\
p & : \text{cans} \Rightarrow p : \neg q. \\
q & : \text{susp} \Rightarrow q : \neg p, \neg r, q. \\
r & : \text{susp} \Rightarrow r : [\neg p], \neg q.
\end{align*}
Example 4

\[ p : \neg q. \]
\[ q : \neg p, \neg r, q. \]
\[ r : \neg p, \neg q. \]

p: cans: p: -[\neg q].
q: susp: q: -[\neg p], [\neg r], q
r: susp: [\neg p], [\neg q].
Example 4

\begin{align*}
p &: \neg q. \\
q &: \neg p, \neg r, q. \\
r &: \neg p, \neg q.
\end{align*}

\begin{align*}
p &: \text{cans} : p : - [\neg q]. \\
\text{fail} (q). \\
r &: \text{susp} : [\neg p], [\neg q].
\end{align*}
Example 4

\begin{align*}
  p & : \neg q. \\
  q & : \neg p, \neg r, q. \\
  r & : \neg p, \neg q. \\
  p & : \text{ans} : p. \\
  & \text{fail}(q). \\
  r & : \text{susp} : \neg p, \neg q.
\end{align*}
Example 4

\[\begin{align*}
p &: \neg q. \\
q &: \neg p, \neg r, q. \\
r &: \neg p, \neg q. \\
p: \text{ans} &: p. \\
\text{fail}(q). \\
\text{fail}(r). \end{align*}\]
Example 5

\[ p: \neg q. \]
\[ q: \neg p, \neg r, q. \]
\[ r: \neg p. \]
\[ r: \neg q, \neg r, p. \]

\[ p: \text{susp} : p: \neg q. \]
\[ q: \text{susp} : q: \neg p, \neg r, q. \]
Example 5

\begin{align*}
p : & \neg q. \quad p : \text{act} : p : \neg [\neg q]. \\
q : & \neg p, \neg r, q. \quad q : \text{act} : q : \neg [\neg p], \neg r, q. \\
r : & \neg p. \\
r : & \neg q, \neg r, p.
\end{align*}
Example 5

\[ p: \neg q. \]
\[ q: \neg p, \neg r, q. \]
\[ r: \neg p. \]
\[ r: \neg q, \neg r, p. \]

\[ p: \text{act} : p: \neg [\neg q]. \]
\[ q: \text{susp}: q: \neg [\neg p], \neg r, q. \]
\[ r: \text{susp}: r: \neg p. \]
\[ r: \text{susp}: r: \neg q, \neg r, p. \]
Example 5

\[ p \leftarrow \neg q. \]
\[ q \leftarrow \neg p, \neg r, q. \]
\[ r \leftarrow \neg p. \]
\[ r \leftarrow \neg q, \neg r, p. \]

\[ p: \text{act} \ p: -[\neg q]. \]
\[ q: \text{susp} \ q: -[\neg p], [\neg r], q. \]
\[ r: \text{susp} \ r: -[\neg p]. \]
\[ r: \text{susp} \ r: -[\neg q], \neg r, p. \]
Example 5

\[
\begin{align*}
p &: \neg q. \\
q &: \neg p, \neg r, q. \\
r &: \neg p. \\
r &: \neg q, \neg r, p.
\end{align*}
\]

\[
\begin{align*}
p &: \text{act} : p : - [\neg q]. \\
q &: \text{sus} : q : - [\neg p], [\neg r], q. \\
r &: \text{can} : r : - [\neg p]. \\
r &: \text{sus} : r : - [\neg q], \neg r, p.
\end{align*}
\]
Example 5

\[ p : \neg q. \quad q : \neg p, \neg r, q. \quad r : \neg p. \quad r : \neg q, \neg r, p. \]

\[ p: act : p : \neg [\neg q]. \quad \text{fail}(q). \quad r: cans : r : \neg [\neg p]. \quad r: susp : r : \neg [\neg q], \neg r, p. \]
Example 5

\[
\begin{align*}
p &: \neg q. \\
q &: \neg p, \neg r, q. \\
r &: \neg p. \\
r &: \neg q, \neg r, p. \\
\end{align*}
\]

\[
\begin{align*}
p &: \text{ans} : p. \\
\text{fail}(q). \\
r &: \text{cans} : r : \neg [\neg p]. \\
r &: \text{susp} : r : \neg [\neg q], \neg r, p. \\
\end{align*}
\]
Example 5

\[ p: \neg q. \quad q: \neg p, \neg r, q. \quad r: \neg p. \quad r: \neg q, \neg r, p. \]

\[ p: \text{ans } p. \quad \text{fail}(q). \quad r: \text{unsu}: r: \neg \neg p. \quad r: \text{susp}: r: \neg \neg q, \neg r, p. \]
Example 5

\[
\begin{align*}
p & : \neg q. \\
q & : \neg p, \neg r, q. \\
r & : \neg p. \\
r & : \neg q, \neg r, p. \\
\end{align*}
\]

\[
\begin{align*}
\text{ans} & : p. \\
\text{fail} & (q). \\
\text{unsu} & : r: - [\neg p]. \\
\text{susp} & : r: - [\neg q], [\neg r], p.
\end{align*}
\]
Example 5

\[
\begin{align*}
p &: \neg q. \\
q &: \neg p, \neg r, q. \\
r &: \neg p. \\
r &: \neg q, \neg r, p. \\
\end{align*}
\]

\[
\begin{align*}
p &: \text{ans} : p. \\
\text{fail} (q). \\
r &: \text{unsu} : r : \neg [\neg p]. \\
r &: \text{susp} : r : \neg [\neg r], p. \\
\end{align*}
\]
Example 5

\[
\begin{align*}
  p & : \neg q. \\
  q & : \neg p, \neg r, q. \\
  r & : \neg p. \\
  r & : \neg q, \neg r, p. \\
  \text{plan} & : p. \\
  \text{fail} (q). \\
  \text{unsu} & : r : [\neg p]. \\
  \text{cans} & : [\neg r], p.
\end{align*}
\]
Currently, SLG-WAM detects positive loops between tabled subgoals
• Currently, SLG-WAM detects positive loops between tabled subgoals
• Answers are not added to a table unless true, or involved in a loop through negation (conditional answers)
• Currently, SLG-WAM detects positive loops between tabled subgoals
• Answers are not added to a table unless true, or involved in a loop through negation (conditional answers)
• There are pathological cases where certain conditional answers are later determined to be true or false
• Currently, SLG-WAM detects positive loops between tabled subgoals
• Answers are not added to a table unless true, or involved in a loop through negation (conditional answers)
• There are pathological cases where certain conditional answers are later determined to be true or false
• This is done by ANSWER COMPLETION
p(X): \neg s.
p(X): \neg p(X).
s: \neg r.
s: \neg p(X).
r: \neg s, r.
p(X): \neg s.
p(X): \neg p(X).
s: \neg r.
s: \neg p(X).
r: \neg s, r.

- WFS model: \{s\} = true, \{r, p(X)\} = false
\begin{itemize}
  \item \textbf{WFS model} : \{s\} = \text{true}, \{r, p(X)\} = \text{false}
  \item \textbf{XSB answer} : \{s\} = \text{true}, \{r\} = \text{false}, \{p(X)\} = \text{undefined}
\end{itemize}
Answer Completion

\begin{align*}
p(X) & : \neg s. \\
p(X) & : \neg p(X). \\
s & : \neg r. \\
s & : \neg p(X). \\
r & : \neg s, r.
\end{align*}

\begin{itemize}
  \item Why?
  \item Delay $\neg s$ in rule 1 producing conditional answer
  \item Return that to $p(X)$ of rule 2, producing conditional answer
  \item Detect positive loop for $r$, making it false
  \item Now $s$ becomes true, SIMPLIFICATION removes conditional answer 1
  \item Finally, conditional answer 2 becomes unsupported, hence undefined
\end{itemize}
Answer Completion (iii)

\[
\begin{align*}
p(X) & : -\neg s. \\
p(X) & : -p(X). \\
s & : -\neg r. \\
s & : -p(X). \\
r & : -\neg s, r.
\end{align*}
\]

- Why?
Why?

Delay \( \neg s \) in rule 1 producing conditional answer
Answer Completion (iii)

\[
\begin{align*}
p(X) & : \neg \neg s. \\
p(X) & : \neg p(X). \\
s & : \neg \neg r. \\
s & : \neg p(X). \\
r & : \neg \neg s, r.
\end{align*}
\]

• Why?
  • Delay \( \neg s \) in rule 1 producing conditional answer
  • Return that to \( p(X) \) of rule 2, producing conditional answer
\( p(X):=\neg s. \)
\( p(X):=-p(X). \)
\( s:=\neg r. \)
\( s:=-p(X). \)
\( r:=\neg s,r. \)

- Why?
- Delay \( \neg s \) in rule 1 producing conditional answer
- Return that to \( p(X) \) of rule 2, producing conditional answer
- Detect positive loop for \( r \), making it false
Answer Completion (iii)

\[\begin{align*}
p(X) & : \neg s. \\
p(X) & : \neg p(X). \\
s & : \neg r. \\
s & : \neg p(X). \\
r & : \neg s, r. \\
\end{align*}\]

- Why?
- Delay \( \neg s \) in rule 1 producing conditional answer
- Return that to \( p(X) \) of rule 2, producing conditional answer
- Detect positive loop for \( r \), making it false
- Now \( s \) becomes true, SIMPLIFICATION removes conditional answer 1
Answer Completion (iii)

\[ p(X) :- \neg s. \]
\[ p(X) :- p(X). \]
\[ s :- \neg r. \]
\[ s :- p(X). \]
\[ r :- \neg s, r. \]

- Why?
  - Delay \( \neg s \) in rule 1 producing conditional answer
  - Return that to \( p(X) \) of rule 2, producing conditional answer
  - Detect positive loop for \( r \), making it false
  - Now \( s \) becomes true, SIMPLIFICATION removes conditional answer 1
  - Finally, conditional answer 2 becomes unsupported, hence undefined
Answer Completion (iv)

- What does ANSWER_COMPLETION do?

• It makes false sets of answers that are not supported
• = conditional answers for completed subgoals that contain only positive dependencies on their delay lists
• To implement it, we search for each subgoal inside the SCC in respect whether every answer it has is supported
• If it’s not, we remove it and propagate its deletion’s simplifications
• Iteration stops after ALL answers of ALL subgoals are supported
• CHECK SUPPORTED ANSWER detects positive loops in answers already visited inside the SCC and returns false
• If already visited and NOT inside the SCC the answer was produced in another branch, returns true
What does ANSWER_COMPLETION do?
It makes false sets of answers that are not supported.
Answer Completion

(iv)

- What does ANSWER_COMPLETION do?
- It makes false sets of answers that are **not** supported
- = conditional answers for completed subgoals that contain only positive dependencies on their delay lists
Answer Completion (iv)

- What does ANSWER_COMPLETION do?
- It makes false sets of answers that are not supported
- = conditional answers for completed subgoals that contain only positive dependencies on their delay lists
- To implement it, we search for each subgoal inside the SCC in respect whether every answer it has is supported
Answer Completion (iv)

- What does ANSWER_COMPLETION do?
- It makes false sets of answers that are not supported
- = conditional answers for completed subgoals that contain only positive dependencies on their delay lists
- To implement it, we search for each subgoal inside the SCC in respect whether every answer it has is supported
- If it’s not, we remove it and propagate its deletion’s simplifications
What does ANSWER_COMPLETION do?

- It makes false sets of answers that are not supported
- Conditional answers for completed subgoals that contain only positive dependencies on their delay lists
- To implement it, we search for each subgoal inside the SCC in respect whether every answer it has is supported
- If it's not, we remove it and propagate its deletion's simplifications
- Iteration stops after ALL answers of ALL subgoals are supported

CHECK_SUPPORTED ANSWER detects positive loops in answers already visited inside the SCC and returns false

If already visited and NOT inside the SCC the answer was produced in another branch, returns true
Answer Completion (iv)

- What does ANSWER_COMPLETION do?
- It makes false sets of answers that are not supported
- = conditional answers for completed subgoals that contain only positive dependencies on their delay lists
- To implement it, we search for each subgoal inside the SCC in respect whether every answer it has is supported
- If it’s not, we remove it and propagate its deletion’s simplifications
- Iteration stops after ALL answers of ALL subgoals are supported
- CHECK_SUPPORTED_ANSWER detects positive loops in answers already visited inside the SCC and returns false
Answer Completion (iv)

- What does ANSWER_COMPLETION do?
- It makes false sets of answers that are not supported.
- = conditional answers for completed subgoals that contain only positive dependencies on their delay lists.
- To implement it, we search for each subgoal inside the SCC in respect whether every answer it has is supported.
- If it’s not, we remove it and propagate its deletion’s simplifications.
- Iteration stops after ALL answers of ALL subgoals are supported.
- CHECK_SUPPORTED_ANSWER detects positive loops in answers already visited inside the SCC and returns false.
- If already visited and NOT inside the SCC the answer was produced in another branch, returns true.
Thanks for your attention :-)